Chapter 1

INTRODUCTION

The Vehicular ad hoc networks (VANETs) have attracted a lot of attentions due to their interesting and promising functionalities including vehicular safety, traffic congestion avoidance, and location based services .We focus on safety driving application, where each vehicle periodically broadcasts messages including its current position, direction and velocity, as well as road information.

Privacy is an important issue in VANETs. As the wireless communication channel is a shared medium, exchanging messages without any security protection over the air can easily leak the information that users may want to keep private. Pseudonym based schemes have been proposed to preserve the location privacy of vehicles. However, those schemes require the vehicles to store a large number of pseudonyms and certifications, and do not support some important secure functionality such as authentication and integrity. The group signature is a promising security scheme to provide privacy in VANETs. All the existing group signature schemes in VANETs are based on centralized key management which preloads keys to vehicles off-line. The centralized key management has some disadvantages. For instance, the system maintenance is not flexible. Another issue regarding the centralized key management is that many existing schemes assume a tamper-proof device being installed in each vehicle. The tamper-proof device normally costs several thousand dollars, such as IBM 4764 card. The framework to be developed in this does not require the expensive tamper-proof device.

We propose and develop a secure distributed key management framework. In our framework, the road side units (RSUs) are responsible for secure group private keys distribution in a localized manner. When a vehicle approaches an RSU, it gets the group private key from the RSU dynamically. All vehicles which get the group private key from the same RSU form a group. A new issue induced by the distributed key management framework is that compromised RSUs may misbehave in the key distribution procedure.

We develop security protocols for the distributed key management framework, which are capable of detecting the compromised RSUs and their collusion with the malicious vehicles if any. In relation to we propose a more efficient and practical cooperative message authentication protocol (CMAP) with an assumption that each safety message carries the location information of the sender vehicle which can be generated by

a global positioning system (GPS) device. Verifiers of each message are defined according to their locations.

Problem statement

As the wireless communication channel is a shared medium, exchanging messages without any security protection over the air can easily leak the information that users may want to keep private. Worse, the hacker could modify the information and broadcast this modified information and cause catastrophes on the road. Any security mechanism which results in a very large communication overhead is also not recommended. Hence, it is of utmost importance to develop a security framework which offers protection for private messages from hacking and also offers very less communication overhead.

Objectives

- To develop a distributed key management framework which has advantages in the revocation of malicious vehicles, system maintenance, and the implementation of heterogeneous security policies.
- To develop a secure key distribution protocol with the capability of preventing RSUs from misbehaving. The protocol guarantees the traceability of compromised RSUs and malicious vehicles.
- An efficient cooperative message authentication protocol is to be developed, by which cooperative verifiers are intelligently selected to significantly reduce the computation and communication overhead in the group signature based implementation.

Chapter 2

PREAMBLE

2.1 Existing system

There have been several proposals for privacy preservation of VANETs. Using pseudonyms is a natural idea. It is preferable to preserve the location privacy of a vehicle by breaking the link ability between two locations, for which the vehicle can update its pseudonym after each transmission. Considering that a powerful adversary may still link the new and old pseudonyms by monitoring the temporal and spatial relations between new and old locations, the techniques of mix zone and silent period have been proposed to enhance the pseudonym scheme. Each vehicle in a mix zone will keep silent in transmission, and randomly update its pseudonyms when it travels out of the mix zone and becomes reactivated. Given a reasonable large mix zone, the location privacy can be well protected due to the untraceability of location and pseudonym updating in the silent period. In the AMOEBA, vehicles form groups. The messages of all group members are forwarded by the group leader, which implies that the privacy of group members is protected by sacrificing the privacy of group leader. Moreover, if a malicious vehicle is selected as a group leader, all group members' privacy may be leaked by the malicious leader.

Disadvantages:

- It doesn't guarantee privacy for the vehicular network.
- Cellular wimax based network is limited to single hop base station to vehicle communication.
- Communication overhead is increased.

2.2 Proposed system

We propose a more efficient and practical cooperative message authentication protocol (CMAP) with an assumption that each safety message carries the location information of the sender vehicle (which can be generated by a global positioning system (GPS) device). Verifiers of each message are defined according to their locations in relation to the sender. Only the selected verifiers check the validity of the message while other vehicles rely on verification results from these verifiers. Compared with our

protocol has smaller packet loss ratio, less computation and communication overhead, as well as better security performance. Hence, it is more efficient and practical in the real application.

Advantages:

- The proposed distributed key management framework which has advantages in the revocation of malicious vehicles, system maintenance, and the implementation of heterogeneous security policies.
- A secure key distribution protocol with the capability of preventing RSUs from misbehaving.
- Cooperative verifiers are intelligently selected to significantly reduce the computation and communication overhead in the group signature based implementation.

Chapter 3

LITERATURE SURVEY

3.1 Vehicular ad-hoc network

A Vehicular Ad-Hoc Network or VANET is a technology that uses moving cars as nodes in a network to create a mobile network. VANET turns every participating car into a wireless router or node, allowing cars approximately 100 to 300 meters of each other to connect and, in turn, create a network with a wide range. As cars fall out of the signal range and drop out of the network, other cars can join in, connecting vehicles to one another so that a mobile Internet is created. It is estimated that the first systems that will integrate this technology are police and fire vehicles to communicate with each other for safety purposes.

3.1.1 Applications

Most of the concerns of interest to MANets are of interest in VANets, but the details differ. Rather than moving at random, vehicles tend to move in an organized fashion. The interactions with roadside equipment can likewise be characterized fairly accurately. And finally, most vehicles are restricted in their range of motion, for example by being constrained to follow a paved highway.

In addition, in the year 2006 the term MANet mostly describes an academic area of research, and the term VANet perhaps its' most promising area of application. VANET offers several benefits to organizations of any size. While such a network does pose certain safety concerns (for example, one cannot safely type an email while driving), this does not limit VANET's potential as a productivity tool. GPS and navigation systems can benefit, as they can be integrated with traffic reports to provide the fastest route to work. A computer can turn a traffic jam into a productive work time by having his email downloaded and read to him by the on-board computer, or if traffic slows to a halt, read it himself. It would also allow for free, VoIP services such as Google Talk or Skype between employees, lowering telecommunications costs. Future applications could involve cruise control making automatic adjustments to maintain safe distances between vehicles or alerting the driver of emergency vehicles in the area.

3.1.2 Technology

In VANET, or Intelligent Vehicular Ad-Hoc Networking, defines an intelligent way of using Vehicular Networking. In VANET integrates on multiple ad-hoc networking technologies such as Wi-Fi IEEE 802.11p, WAVE IEEE 1609, Wimax IEEE 802.16, Bluetooth, IRA, and ZigBee for easy, accurate, effective and simple communication between vehicles on dynamic mobility. Effective measures such as media communication between vehicles can be enabled as well as methods to track the automotive vehicles.

In VANET helps in defining safety measures in vehicles, streaming communication between vehicles, infotainment and telematics. Vehicular Ad-hoc Networks are expected to implement a variety of wireless technologies such as Dedicated Short Range Communications (DSRC) which is a type of Wi-Fi. Other candidate wireless technologies are Cellular, Satellite, and Wimax. Vehicular Ad-hoc Networks can be viewed as component of the Intelligent Transportation Systems (ITS). As envisioned in ITS, vehicles communicate with each other via Inter-Vehicle Communication (IVC) as well as with roadside base stations via Roadside-to-Vehicle Communication (RVC). The optimal goal is that vehicular networks will contribute to safer and more efficient roads in the future by providing timely information to drivers and concerned authorities.

3.2 Data Encryption Standard

The Data Encryption Standard (DES) specifies a FIPS approved cryptographic algorithm as required by FIPS 140-1.Encrypting data converts it to an unintelligible form called cipher. Decrypting cipher converts the data back to its original form called plaintext. The algorithm described in this standard specifies both enciphering and deciphering operations which are based on a binary number called a key. A key consists of 64 binary digits ("O"s or "1"s) of which 56 bits are randomly generated and used directly by the algorithm. The other 8 bits, which are not used by the algorithm, are used for error detection. The 8 error detecting bits are set to make the parity of each 8-bit byte of the key odd, i.e., there is an odd number of "1"s in each 8-bit byte[1]. Authorized users of encrypted computer data must have the key that was used to encipher the data in order to decrypt it. The encryption algorithm specified in this standard is commonly known among those using the standard. The unique key chosen for use in a particular application makes the results of encrypting data using the algorithm unique. Selection of a different

key causes the cipher that is produced for any given set of inputs to be different. The cryptographic security of the data depends on the security provided for the key used to encipher and decipher the data.
Data can be recovered from cipher only by using exactly the same key used to encipher it. Unauthorized recipients of the cipher who know the algorithm but do not have the correct key cannot derive the original data algorithmically. However, anyone who does have the key and the algorithm can easily decipher the cipher and obtain the original data. A standard algorithm based on a secure key thus provides a basis for exchanging encrypted computer data by issuing the key used to encipher it to those authorized to have the data. Data that is considered sensitive by the responsible authority, data that has a high value, or data that represents a high value should be cryptographically protected if it is vulnerable to unauthorized disclosure or undetected modification during transmission or while in storage. A risk analysis should be performed under the direction of a responsible authority to determine potential threats. The costs of providing cryptographic protection using these standard as well as alternative methods of providing this protection and their respective costs should be projected. A responsible authority then should make a decision, based on these analyses, whether or not to use cryptographic protection and this standard.

The algorithm is designed to encipher and decipher blocks of data consisting of 64 bits under control of a 64-bit key. Deciphering must be accomplished by using the same key as for enciphering, but with the schedule of addressing the key bits altered so that the deciphering process is the reverse of the enciphering process. A block to be enciphered is subjected to an initial permutation IP, then to a complex key-dependent computation and finally to a permutation which is the inverse of the initial permutation IP^{-1}. The key-dependent computation can be simply defined in terms of a function *f*, called the cipher function, and a function KS, called the key schedule. A description of the computation is given first, along with details as to how the algorithm is used for encipherment. Next, the use of the algorithm for decipherment is described. Finally, a definition of the cipher function *f* is given in terms of primitive functions which are called the selection functions S_i and the permutation function P. S_i, P and KS of the algorithm are contained in this.

.

3.3 Elliptic curve

In mathematics, an **elliptic curve** is a smooth, projective algebraic curve of genus one, on which there is a specified point *O*. An elliptic curve is in fact an abelian variety — that is, it has a multiplication defined algebraically with respect to which it is a (necessarily commutative) group — and *O* serves as the identity element. Often the curve itself, without *O* specified, is called an elliptic curve.

Any elliptic curve can be written as a plane algebraic curve defined by an equation of the form:

$$y^2 = x^3 + ax + b \qquad [3.1]$$

Which is non-singular; that is, its graph has no cusps or self-intersections. (When the characteristic of the coefficient field is equal to 2 or 3, the above equation is not quite general enough to comprise all non-singular cubic curves; see below for a more precise definition.) The point *O* is actually the "point at infinity" in the projective plane.

If $y^2 = P(x)$, where *P* is any polynomial of degree three in *x* with no repeated roots, then we obtain a nonsingular plane curve of genus one, which is thus also an elliptic curve. If *P* has degree four and is squarefree this equation again describes a plane curve of genus one; however, it has no natural choice of identity element. More generally, any algebraic curve of genus one, for example from the intersection of two quadric surfaces embedded in three-dimensional projective space, is called an elliptic curve, provided that it has at least one rational point.

Using the theory of elliptic functions, it can be shown that elliptic curves defined over the complex numbers correspond to embeddings of the torus into the complex projective plane. The torus is also an abelian group, and in fact this correspondence is also a group isomorphism.

Elliptic curves are especially important in number theory, and constitute a major area of current research; for example, they were used in the proof, by Andrew Wiles (assisted by Richard Taylor), of Fermat's Last Theorem. They also find applications in cryptography (see the article elliptic curve cryptography) and integer factorization.

3.3.1 Elliptic curves over the real numbers

Although the formal definition of an elliptic curve is fairly technical and requires some background in algebraic geometry, it is possible to describe some features of elliptic curves over the real numbers using only high school algebra and geometry.

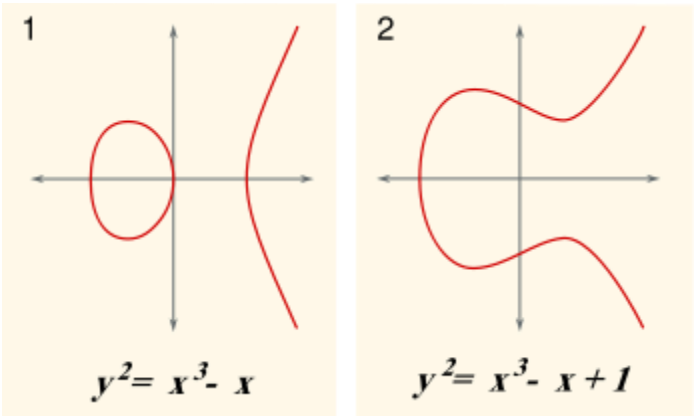


Figure 3.1: Graphs of curves $y^2 = x^3 - x$ and $y^2 = x^3 - x + 1$

The definition of elliptic curve also requires that the curve be non-singular. Geometrically, this means that the graph has no cusps, self-intersections, or isolated points. Algebraically, this involves calculating the discriminant

$$\Delta = -16(4a^3 + 27b^2). \qquad [3.2]$$

The curve is non-singular if and only if the discriminant is not equal to zero. (Although the factor −16 seems irrelevant here, it turns out to be convenient in a more advanced study of elliptic curves.)

The (real) graph of a non-singular curve has *two* components if its discriminant is positive, and *one* component if it is negative. For example, in the graphs shown in figure to the right, the discriminant in the first case is 64, and in the second case is −368.

3.4 Elliptic Curve Cryptography

In mathematics, an elliptic curve is a smooth, projective algebraic curve of genus one, on which there is a specified point *O*. An elliptic curve is in fact an abelian variety — that is, it has a multiplication defined algebraically with respect to which it is a

(necessarily commutative) group — and *O* serves as the identity element. Often the curve itself, without *O* specified, is called an elliptic curve. Any elliptic curve can be written as a plane algebraic curve defined by an equation of the form:

$$y^2 = x^3 + ax + b \qquad [3.3]$$

Which is non-singular; that is, its graph has no cusps or self-intersections. The point *O* is actually the "point at infinity" in the projective plane. If $y^2 = P(x)$, where *P* is any polynomial of degree three in *x* with no repeated roots, then we obtain a non-singular plane curve of genus one, which is thus also an elliptic curve. If *P* has degree four and is square free this equation again describes a plane curve of genus one; however, it has no natural choice of identity element. More generally, any algebraic curve of genus one, for example from the intersection of two quadric surfaces embedded in three-dimensional projective space is called an elliptic curve, provided that it has at least one rational point. Using the theory of elliptic functions, it can be shown that elliptic curves defined over the complex numbers correspond to embedding of the torus into the complex projective plane. The torus is also an abelian group, and in fact this correspondence is also a group isomorphism. Elliptic curves are especially important in number theory, and constitute a major area of current research; for example, they were used in the proof, by Andrew Wiles (assisted by Richard Taylor), of Fermat's Last Theorem. They also find applications in cryptography.

Elliptic curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. The use of elliptic curves in cryptography was suggested independently by Neal Koblitz and Victor S. Miller in 1985. Public-key cryptography is based on the intractability of certain mathematical problems. Early public-key systems, such as the RSA algorithm, are secure assuming that it is difficult to factor a large integer composed of two or more large prime factors. For elliptic-curve-based protocols, it is assumed that finding the discrete logarithm of a random elliptic curve element with respect to a publicly-known base point is infeasible. The size of the elliptic curve determines the difficulty of the problem.

For current cryptographic purposes, an elliptic curve is a plane curve which consists of the points satisfying the equation.

Along with a distinguished point at infinity, (The coordinates here are to be chosen from a fixed finite field of characteristic not equal to 2 or 3, or the curve equation will be somewhat more complicated.) This set together with the group operation of the elliptic group theory form an Abelian group, with the point at infinity as identity element. The structure of the group is inherited from the divisor group of the underlying algebraic variety. As for other popular public key cryptosystems, no mathematical proof of security has been published for ECC as of 2009. However, the U.S. National Security Agency has endorsed ECC by including schemes based on it in its Suite B set of recommended algorithms and allows their use for protecting information classified up to top secret with 384-bit keys. While the RSA patent expired in 2000, there are patents in force covering certain aspects of ECC technology, though the Federal elliptic curve digital signature standard (ECDSA; NIST FIPS 186-3) and certain practical ECC-based key exchange schemes (including ECDH) can certainly be implemented without infringing them. Several discrete logarithm-based protocols have been adapted to elliptic curves, replacing the group $\mathbb{Z}_{pq}$ with an elliptic curve:

- the elliptic curve Diffie–Hellman key agreement scheme is based on the Diffie–Hellman scheme,
- the Elliptic Curve Digital Signature Algorithm is based on the Digital Signature Algorithm,
- The ECMQV key agreement scheme is based on the MQV key agreement scheme.

At the RSA Conference 2005, the National Security Agency (NSA) announced Suite B which exclusively uses ECC for digital signature generation and key exchange. The suite is intended to protect both classified and unclassified national security systems and information. Recently, a large number of cryptographic primitives based on bilinear mappings on various elliptic curve groups, such as the Weil and Tate pairings, have been introduced. Schemes based on these primitives provide efficient identity-based encryption as well as pairing-based signatures, sign cryption, key agreement, and proxy re-encryption.

To use ECC all parties must agree on all the elements defining the elliptic curve, that is, the domain parameters of the scheme. The field is defined by p in the prime case and the pair of m and f in the binary case. The elliptic curve is defined by the constants a and

b used in its defining equation. Finally, the cyclic subgroup is defined by its *generator* (aka. *base point*) G. For cryptographic application the order of G, that is the smallest non-negative number n such that nG = O, must be prime. Since n is the size of a subgroup of $E(\mathbb{F}_p)$ it follows from Lagrange's theorem that the number $h = \frac{|E|}{n}$ is an integer. In cryptographic applications this number h, called the *cofactor*, must be small ($h \leq 4$) and, preferably, h = 1. Let us summarize: in the prime case the domain parameters are (p,a,b,G,n,h) and in the binary case they are (m,f,a,b,G,n,h). Unless there is an assurance that domain parameters were generated by a party trusted with respect to their use, the domain parameters *must* be validated before use. The generation of domain parameters is not usually done by each participant since this involves counting the number of points on a curve which is time-consuming and troublesome to implement. As a result several standard bodies published domain parameters of elliptic curves for several common field sizes.

3.5 Java

One characteristic of Java is portability, which means that computer programs written in the Java language must run similarly on any supported hardware/operating-system platform. This is achieved by compiling the Java language code to an intermediate representation called Java byte code, instead of directly to platform-specific machine code. Java byte code instructions are analogous to machine code, but are intended to be interpreted by a virtual machine (VM) written specifically for the host hardware. End-users commonly use a Java Runtime Environment (JRE) installed on their own machine for standalone Java applications, or in a Web browser for Java applets. Standardized libraries provide a generic way to access host-specific features such as graphics, threading and networking. A major benefit of using byte code is porting. However, the overhead of interpretation means that interpreted programs almost always run more slowly than programs compiled to native executable would. Just-in-Time compilers were introduced from an early stage that compiles byte codes to machine code during runtime. Over the years, this JVM built-in feature has been optimized to a point where the JVM's performance competes with natively compiled C code. Java was conceived by James Gosling, Patrick Naughton, Chris Wrath, Ed Frank, and Mike Sheridan at Sun Micro system. It is an platform independent programming language that extends its features

wide over the network.Java2 version introduces an new component called "Swing" – is a set of classes that provides more power & flexible components than are possible with AWT.

- It's a light weight package, as they are not implemented by platform-specific code.
- Related classes are contained in javax.swing and its sub packages, such as javax.swing.tree.
- Components explained in the Swing have more capabilities than those of AWT.

Java is a high-level programming language that is all of the following:

- Simple
- Object-oriented
- Distributed
- Interpreted
- Robust
- Secure
- Architecture-neutral
- Portable

Java is also unusual in that each Java program is both compiled and interpreted. With a compiler, you translate a Java program into an intermediate language called *Java byte codes*-the platform-independent codes interpreted by the Java interpreter. With an interpreter, each Java byte code instruction is parsed and run on the computer. Compilation happens just once; interpretation occurs each time the program is executed. This figure illustrates how this works.

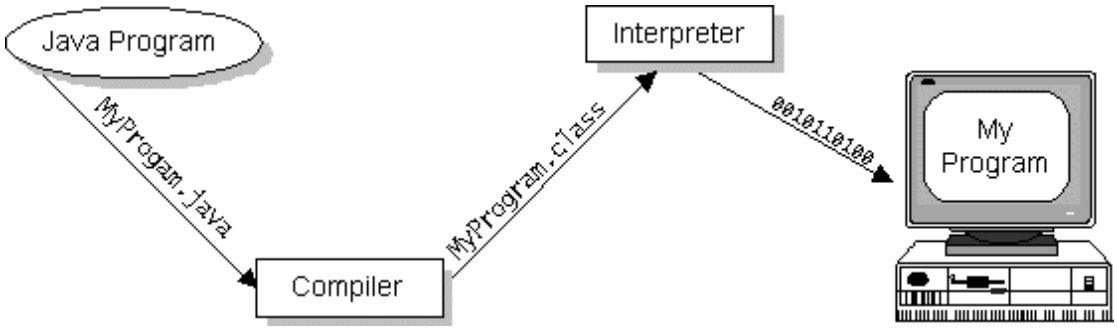


Figure 3.2: Execution of a java program

Java byte codes can be considered as the machine code instructions for the *Java Virtual Machine* (Java VM). Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM. The Java VM can also be implemented in hardware. Java byte codes help make "write once, run anywhere" possible. The Java program can be compiled into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. For example, the same Java program can run on Windows NT, Solaris, and Macintosh.

3.5.1 Java Platform

A platform is the hardware or software environment in which a program runs. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other, hardware-based platforms. Most other platforms are described as a combination of hardware and operating system. The Java platform has two components:

- The *Java Virtual Machine* (Java VM)
- The *Java Application Programming Interface* (Java API)

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries (*packages*) of related components. The following figure depicts a Java program, such as an application or applet, that's running on the Java platform. As the figure shows, the Java API and Virtual Machine insulates the Java program from hardware dependencies.

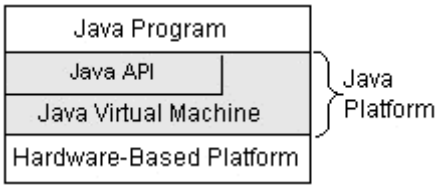


Figure 3.3: Java platform

As a platform-independent environment, Java can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring Java's performance close to that of native code without threatening portability.

3.5.3Java support

Probably the most well-known Java programs are *Java applets*. An applet is a Java program that adheres to certain conventions that allow it to run within a Java-enabled browser. However, Java is not just for writing cute, entertaining applets for the World Wide Web ("Web"). Java is a general-purpose, high-level programming language and a powerful software platform. Using the generous Java API, we can write many types of programs. The most common types of programs are probably applets and applications, where a Java application is a standalone program that runs directly on the Java platform.

With packages of software components that provide a wide range of functionality. The *core API* is the API included in every full implementation of the Java platform. The core API gives you the following features:

- **The Essentials**: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- **Applets**: The set of conventions used by Java applets.
- **Networking**: URLs, TCP and UDP sockets, and IP addresses.
- **Internationalization**: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
- **Security**: Both low-level and high-level, including electronic signatures, public/private key management, access control, and certificates.
- **Software components**: Known as JavaBeans, can plug into existing component architectures such as Microsoft's OLE/COM/Active-X architecture, OpenDoc, and Netscape's Live Connect.

- **Object serialization**: Allows lightweight persistence and communication via Remote Method Invocation (RMI).

3.5.4 Java Swing

The AWT is a crucial part of Java, but its component set is no longer widely used to create graphic user interface. Today, most programmers use swing for this purpose. Swing is a set of classes that provides more powerful and flexible GUI components than does the AWT. Swing provides the look and feel of the modern Java GUI.

Two Key Swing Features:

Swing was created to address the limitations present in the AWT. It does this through two key features:

Swing components are lightweight: With very few exceptions, swing components are lightweight. This means that they are written entirely in java and do not map directly to platform-specific peers. Because lightweight components are rendered using graphics primitives, they can be transparent, which enables nonrectangular shapes. Thus, lightweight components are more efficient and more flexible. Furthermore, because lightweight components do not translate into native peers, the look and feel of each component is determined by swing, not by the underlying operating system.

Swing supports a pluggable look and feel: Swing supports a pluggable look and feel. Because each swing component is rendered by java code rather than by native peers, the look and feel of the component is under the control of swing. This fact means that it is possible to separate the look and feel of a component from the logic of the component, and this is what swing does.

Separating out the look and feel provides a significant advantage: it becomes possible to change the way that a component is rendered without affecting any of its other aspects. Pluggable look- and- feels offer several important advantages. It is possible to define a look and feel that is consistent across all platforms. Conversely, it is possible to create a look and feel that acts like a specific platform. For eg, if you know that an application will be running only in a windows environment, it is possible to specify the windows look and feel. It is also possible to design a custom look and feel. Finally, the look and feel can be changed dynamically at run time. Swing is a set of classes that provides more powerful and flexible components that are possible with AWT. In addition

to the familiar components, such as button checkboxes and labels, swing supplies several exciting additions, including tabbed panes, scroll panes, trees and tables.

3.5.5 Java Development Toolkit

Java development tool kit comes with a collection of tools that are used for developing and running java programs. They include:

Tools	Description
Applet viewer	Enables us to run Java applets (without actually using a Java-compatible browser).
Java	Java interpreter, which runs applets and application by reading and interpreting byte code files.
Javac	The Java complier, which translates Java source code to byte code files that the interpreter can understand.
Javadoc	Creates HTML-format documentation from Java source code files.
Javah	Produces header files for use with native methods.
Javap	Java disassemble, which enables us to convert byte code files into a program description.
Jdb	Java debuggers which helps us to find errors in our program.

Table 3.1: Java Development tool kit

To create a java program, we need to create a source code file using a text editor. The sourcecode is then complied using the java compiler **javac** and executed using the Java interpreter **java**. The Java debugger **jdb** is used to find errors, if any, in the sourcecode.

A compiled Java program can be converted into a source code with the help of Java disassembles **javap**.

3.5.6 Socket Connection

Before an application program (client or server) can transfer any data, it must first create an end point for communication by calling socket. Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.

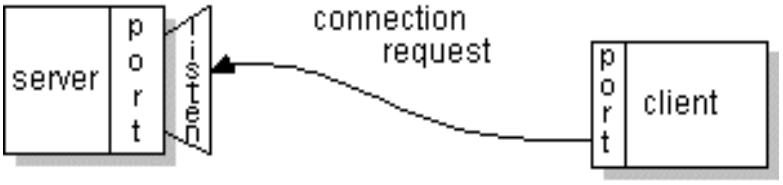


Fig 3.4: Socket Connection Request

If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.

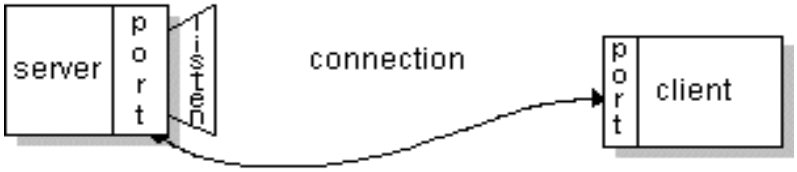


Fig 3.5: Socket Connection

On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.

Chapter 4

SYSTEM REQUIREMENTS SPECIFICATION

4.1 Software Requirement Specification

Requirements analysis is done in order to understand the problem the software system is to solve. The emphasis in requirements analysis is on identifying what is needed from the system, not how the system will achieve its goals. For complex systems, even determining what is needed is a difficult task. The goal of the requirements activity is to document the requirements in a software requirements specification document. There are two major activities in this phase: problem understanding or analysis and requirement specification. In problem analysis, the aim is to understand the problem and its context, and the requirements of the new system that is to be developed. Understanding the requirements of a system that does not exist is difficult and requires creative thinking. Once the problem is analyzed and the essentials understood, the requirements must be specified in the requirement specification document. The goal of the requirements activity is to produce the Software Requirements Specification (SRS) that describes what the proposed software should do without describing how the software will do it.

A Software Requirements Specification (SRS) is a complete description of the behaviour of the system to be developed. It includes a set of use cases that describe the interactions that the users will have with the software. Use cases are also known as functional requirements. In addition to use cases, the SRS also contains non-functional (or supplementary) requirements. Non-functional requirements are requirements which impose constraints on the design or implementation (such as performance engineering requirements, quality standards, or design constraints).

The software requirement specification is produced at the culmination of the analysis task. The function and performance allocated to software as part of system engineering are refined by establishing a complete information description as functional representation, a representation of system behavior, an indication of performance requirements and design constraints, appropriate validation criteria.

4.2 Functional Requirement Specification

Functional requirements precisely state the functions of the system, what it should do and what is should not do. Functions are provided to the users in a GUI. All the functionalities are provided in the main interface.

- **Set up a Road Side Unit:** We need a key distribution agent to distribute the keys to the nodes which is done by setting up a Road Side Unit. The RSU should manage the key generation to the nodes and also distribution of keys when two registered nodes communicate.
- **Key generation:** We need to implement some public key cryptographic scheme which generates the keys for each node. This should be installed in the RSU which acts as an authority and generates the keys for the node asking for a registration in it.
- **Message Encryption:** The messages sent over the network to other nodes must be encrypted to ensure confidentiality of the message. Any conventional cryptographic scheme need to be used to convert the plaintext into a cipher. Authorized users of encrypted computer data must have the key that was used to encipher the data in order to decrypt it.
- **Managing network:** There need to be a mechanism which intelligently selects the neighbours in a co-operative manner to reduce overhead. The node must be able to periodically update the cost in the routing table and also the route to reach the nodes.
- **Restricting communication with nodes which are not registered with RSU:** We also need to ensure that the sender and the receiver node must be registered with the RSU to communicate with each other. Any attempts made by the nodes to communicate with the nodes without registration must be prohibited.
- **Ensuring key distribution via RSU to the sender and the receiver:** During the message encryption by sender, the node needs the RSU to distribute the public key of the receiver and during the decryption of the receiver; the RSU should distribute the key of the sender.

4.3 Non Functional Requirements Specification

The non-functional requirements arise through user needs, because of budget constraints, because of organizational politics, with other software or software systems. The non-functional requirements may come from required characteristics of the software, the organization developing the software or from external sources. The following are the types of non-functional requirements.

- Usability requirements.
- Efficiency requirements.
- Reliability requirements.
- Hardware Reliability.
- Software Reliability.
- Operator Reliability.
- Portability Requirements.
- Organizational Requirements.
- Delivery Requirements.
- Implementation Requirements.
- External Requirements.
- Legislative Requirements.
- Safety Requirements.

4.4 Domain Requirement Specification

Domain requirements basically mean the workspace or the range or the environment in which the product will work properly. Basically, the domain requirements include both the functional and non-functional requirements.

So, for proper working of the project, it is required that the product should meet the functional and non-functional requirements. It includes things like the full modules and their working and technology used. The product should be fully functional without any kind of errors and any kind of difficulties. This may also cover the hardware requirements as the products will also need the good and efficient hardware for the working of the software product.

If the product does not work properly, it merely means that the analyst hasn't gathered proper information for the correct working of the product. So, it is really a big and difficult task for the system analyst to find the full domain requirement so that the

software product will be fully functional and should work in most efficient manner producing great results with less time consumption. The analyst should also see the security, robustness and efficiency. All types of testing should be done so that when the product is live, it should not produce any kind of error or problem.

4.5 Hardware Requirements

Processor	: Any Processor above 500 MHz
Ram	: 128Mb.
Hard Disk	: 10 GB.
Compact Disk	: 699 Mb.
Input device	: Standard Keyboard and Mouse.
Output device	: VGA and High Resolution Monitor.

4.6 Software Requirements

Operating System	: Windows Family.
Language	: Java
Development kit	: JDK 1.7
Front End	: Java Swing
IDE	: Net Beans

Chapter 5

SYSTEM DESIGN

5.1 Introduction

Design is the first step in the development phase for any engineering product or system. It may be defined as "the process of applying various techniques and principles for the purpose of defining a device, a process or a system in sufficient detail to permit its physical realization".

Design is a meaningful engineering representation of something that is to be built. Software design is an iterative process through which requirements are translated into a "blueprint" for constructing the software.

When designing the system the points to be taken care of are:

- Identify the data to be stored.
- Identify the user requirement.
- Need to maintain data and retrieve them whenever wanted.
- Identification of the inputs and arriving at the user defined outputs screens.
- System specification.
- Security specification.
- View of the future implications of the project.

The potential objects are thoroughly analyzed. Class hierarchies are identified to check whether the system is behaving the way it has to. There after the classes are individually tested and subsequently they are integrated to form the overall system.

The detailed design or low-level design involves recording of designers thought and decisions and to represent the design so that one can view it. For this design notation are used. Design notations are largely meant to be used during the process of design and are used to represent design or design decisions .They are meant largely for the designer that he can quickly represent decisions in a compact manner that can evaluate and modify.

Once the designer is satisfied with the design, he has produced; the design is to be precisely specified in the form of document. While the design represented using the design notation is largely used by the designer, a design specification has to be so precise and that other programmers can use as it as basis further development design specification uses textual structures, with design notation helping understanding.

5.2 Modules

Registration: When a node wants to send messages to another node, both the sender and the receiver must be registered with the RSU. Therefore, each node would send a register request to the RSU. The RSU would then verify the node, generate keys for them and register the nodes in its network. The RSU will reply back the keys which are generated. Now that the nodes are registered, the RSU would then be used for key retrieval by the nodes while sending and receiving messages.

Message transfer: When a node wants to communicate with another node and both are registered with the RSU, it would proceed to encrypt the message. The sender will send a getkey request to the RSU requesting for the public key of the receiver node. The RSU would process the getkey request sent by the node and reply back with the receiver's public key. Now the message is converted into a cipher and broadcasted into the network. The message would pass upon the intermediate nodes and reach the receiver. The receiver would send the getkey request to the RSU requesting for the public key of the sender. The RSU, upon receiving the request, would send the public key of the sender. The node would now proceed to decrypt the message.

Mobile Agent: The mobile agent consists of a forward agent and a reverse agent. This module is responsible for forwarding the message from the source to the destination in the shortest path possible. The mobile agent would continuously check on its neighboring nodes by forwarding the cost and dictionary information to them. Since the nodes are mobile, the agent looks for changes in the cost of its neighbors, updates them accordingly and also restructure the path of message path if a shorter path exists.

Road Side Units: The RSU generates and assigns the keys to every node that registers to it. For nodes to communicate among themselves, they should be first registered to the RSU. While sending messages, the sender node would request the public key of the receiver from the RSU. To decrypt the message received, the receiver node would request for the public key of the sender node. Thus RSU plays a very important role in both the key distribution and the communication phase.

Nodes: The nodes are nothing but the vehicles that form the network for communication. In this system, the nodes would register themselves with the nearest RSU and start communicating with other nodes. The nodes would contain a forward agent and a reverse

agent which finds and routes the path from the source to the destination. The message will be encrypted in the node by using the public key of the receiver (sent by the RSU) and it is sent via intermediate nodes.

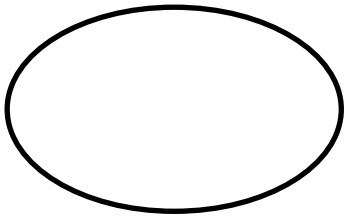
Key generation and message encryption: We employ the Elliptic Curve Cryptography to generate keys to each node. For encrypting the message to be sent, we use the Data Encryption Standard (DES). The DES requires a key to encrypt the message and without the key on the receiver side, it is impossible to decrypt the message algorithmically. We arrive at the key on the sender side based on key distribution from the RSU and the node's private key. On the receiver's side, the sender's public key and the receiver's private key is used to arrive at the same key which was used to encrypt the message. This is possible because the public and the private keys of the nodes are generated using the points on the curve. It is interesting to note that the shared key will only be the same if the intended destination calculates it. Any other malicious node trying to decipher the message would not be able to arrive at the common shared key and hence cannot decrypt the message.

5.3 Data Flow Diagram

Data Flow Diagrams are the intuitive ways of showing how data is processed by the system. At the analysis level, they should be used to model the way in which data is processed in the existing system. The notations used in these models represent functional processing, data stores and data movements between functions.

5.3.1 Symbols used in DFD

Process: Here flow of data is transformed. Example: Searching, Key generating etc.



External Entity: A source or destination of data which is external to the system.

Data Flow: It is a packet of data. It may be in the form of a document, plaintext etc.

5.3.2 Registration

Both sender and the receiver nodes should register to the Road Side Unit before exchanging messages. The nodes would send a register request to the Road Side Unit and the Distributed Key Authority would generate keys for the respective nodes after checking the authenticity of the node.

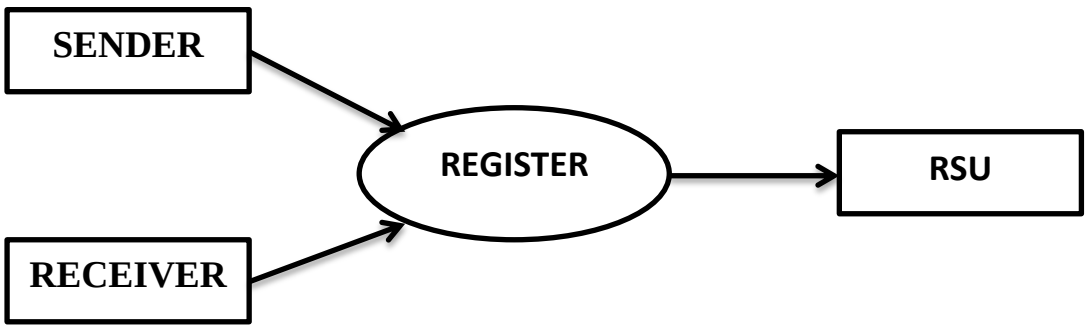


Figure 5.1: DFD of Registration Phase

5.3.3 Communication

Both the sender and receiver nodes must be registered with the RSU before communicating with each other. Without registration, communication is not possible between nodes. The sender would check whether the receiver is registered with RSU and retrieve the public key of the Receiver. The message is broadcasted in the network after encryption. The receiver to decrypt the message and to check the authenticity of the node, would check in with the RSU and retrieve the public key of the sender if the node is authenticated. Figure 5.2 shows the data flow diagram of the communication between nodes.

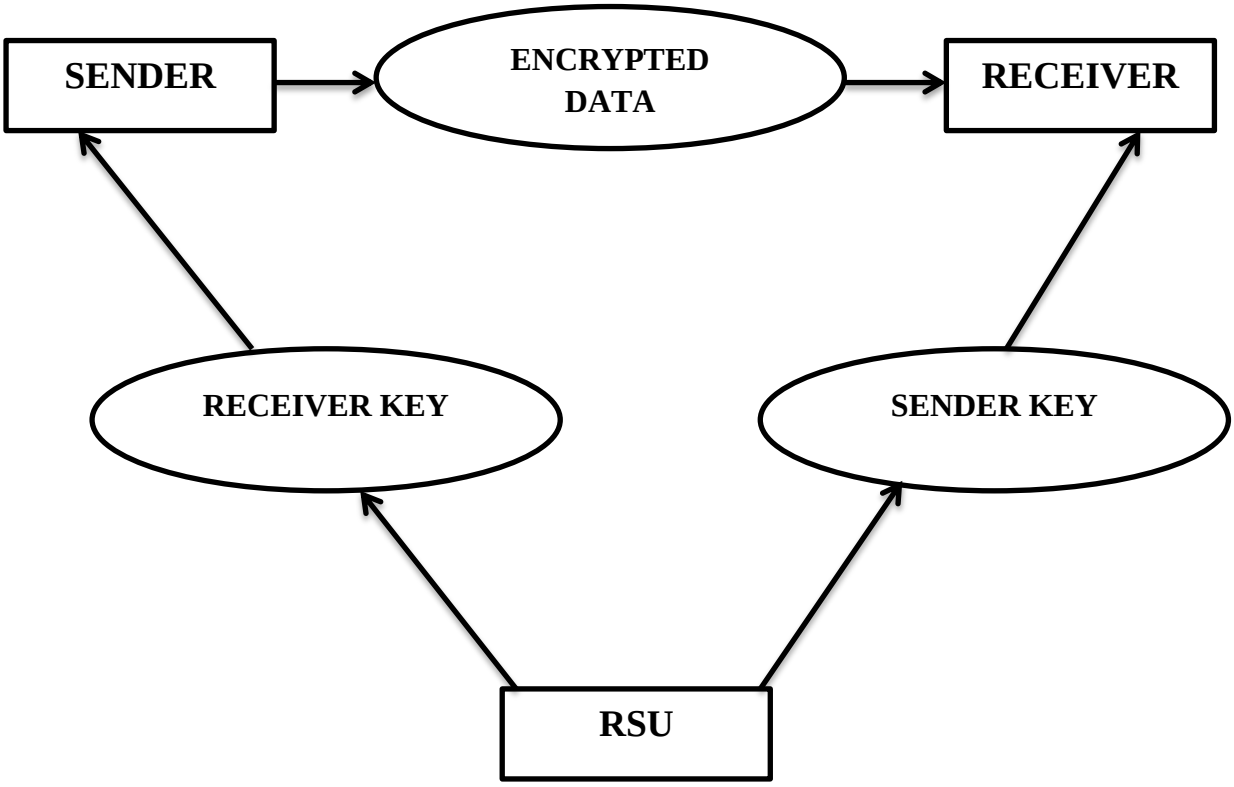


Figure 5.2: Communication between sender and the receiver.

5.3.4 Mobile Agent

The mobile agent is responsible for routing the VANETs and uses the AntNet principle in detecting the path to the destination. The mobile agent consists of two sub-

modules called forward agents and reverse agents. Together, they are responsible for finding the shortest path from the source node to the destination node.

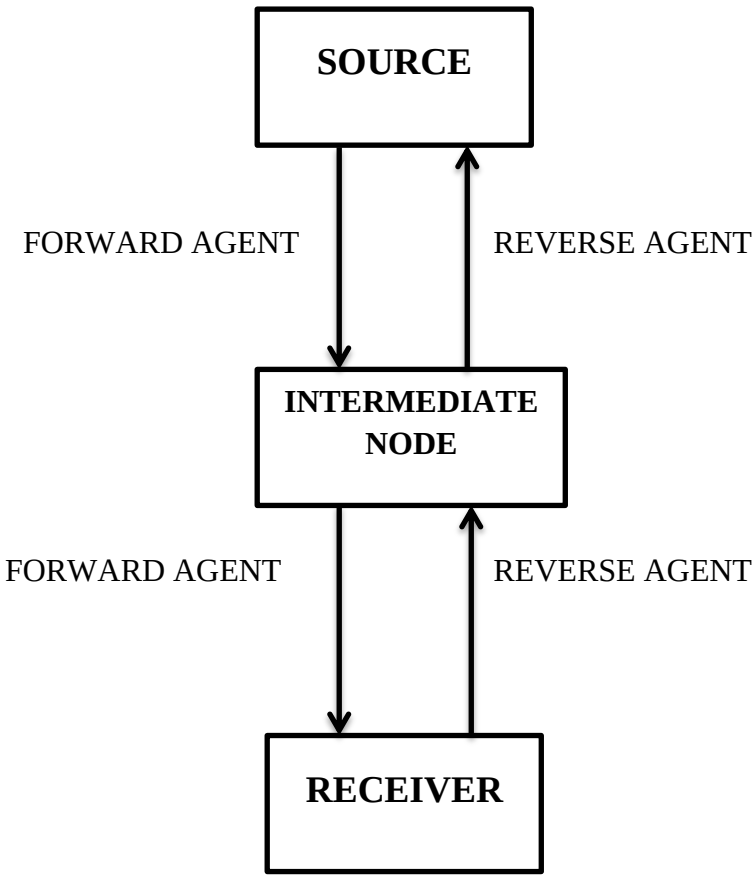


Figure 5.3: Working of Mobile Agent

5.4 Architecture

System architecture is the conceptual model that defines the structure, behavior and more views of the system. An architecture description is a formal description and representation of a system, organized in the way that supports reasoning about the structuring of the system which comprises system components, the externally visible properties of those components, the relationships (ex. behavior) between them, and provides a plan from which products can be procured and system development, that will work together to implement the overall system. The language for the architecture description is called the architecture description language.

Nodes are ordinary vehicles on the road that can communicate with each other and RSUs. The network consists of 4 nodes all being related to the same Road Side Unit.

RSUs deployed at the road sides which are in charge of key management in our framework. These four nodes form an ad-hoc network and are dependent on the RSU for the key distribution. However, the message from the sender to the receiver is broadcasted in the network using intermediate nodes if necessary. Figure 5.4 shows the network architecture of the system consisting of RSU and four nodes.

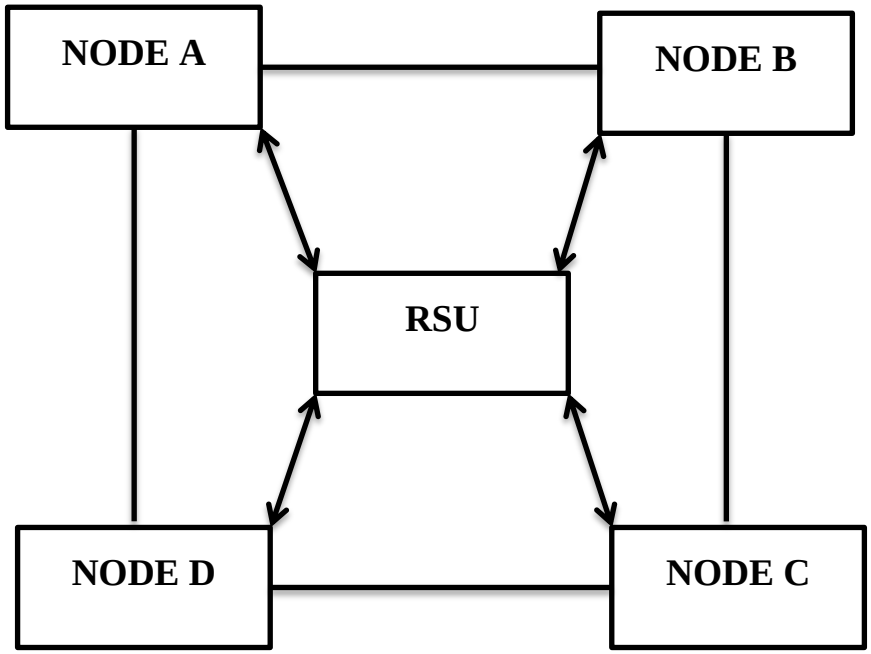


Figure 5.4: System Architecture

5.5 Use Case Diagram

A use case diagram in the UML is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases.

The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

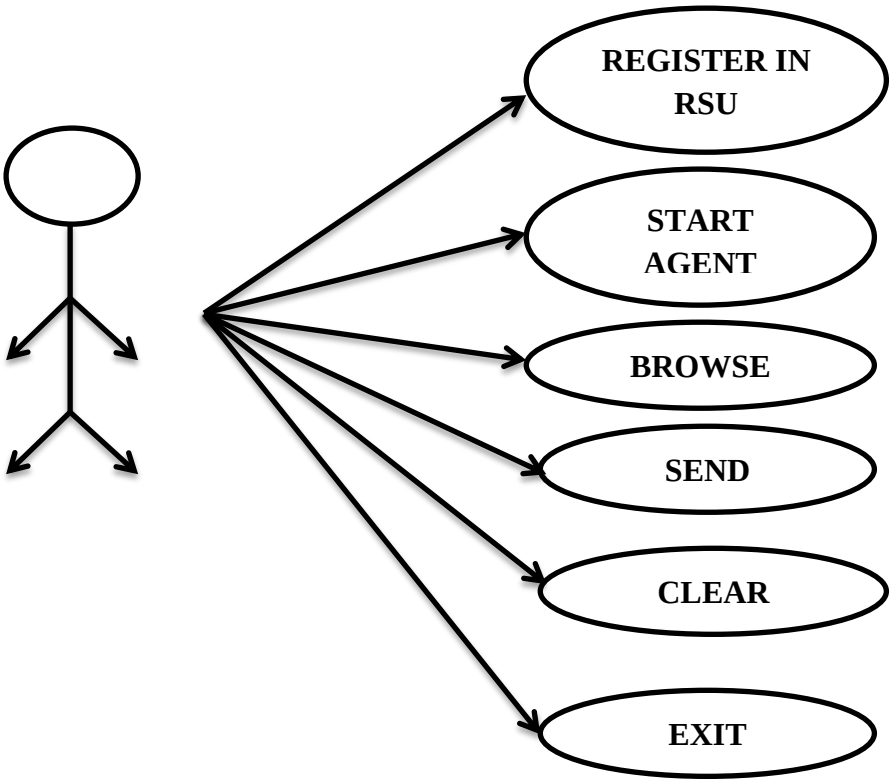


Figure 5.5: Use Case Diagram

5.6 Sequence Diagram

A sequence diagram shows the participants in an interaction and the sequence of messages among them; it shows the interaction of a system with it actors to perform all or part of use case. Each actor as well as the system is represented by a vertical line called a life line and each message by a horizontal arrow from the sender to the receiver. Time proceeds from top to bottom, but the spanning is irrelevant, the diagram shows only the sequence of messages, not their exact timing. Figure 5.6 shows the sequence diagram of the system.

- The sender and the receiver register themselves with the RSU and obtain their respective keys.
- The sender would request the RSU to give the public key of the receiver node which is very much necessary for encryption.
- The message is encrypted and then passed through the intermediate node and reaches the receiver.

- The receiver would then request the RSU for the public key of the sender and would decrypt the message.

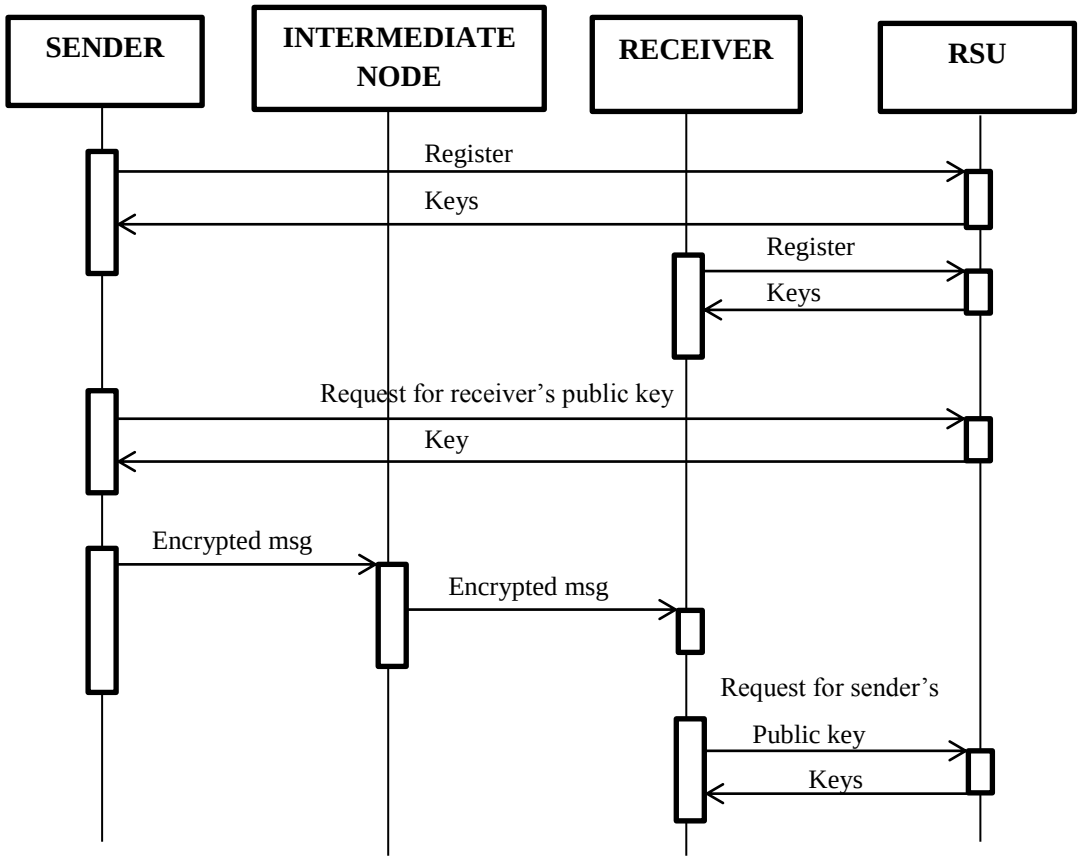
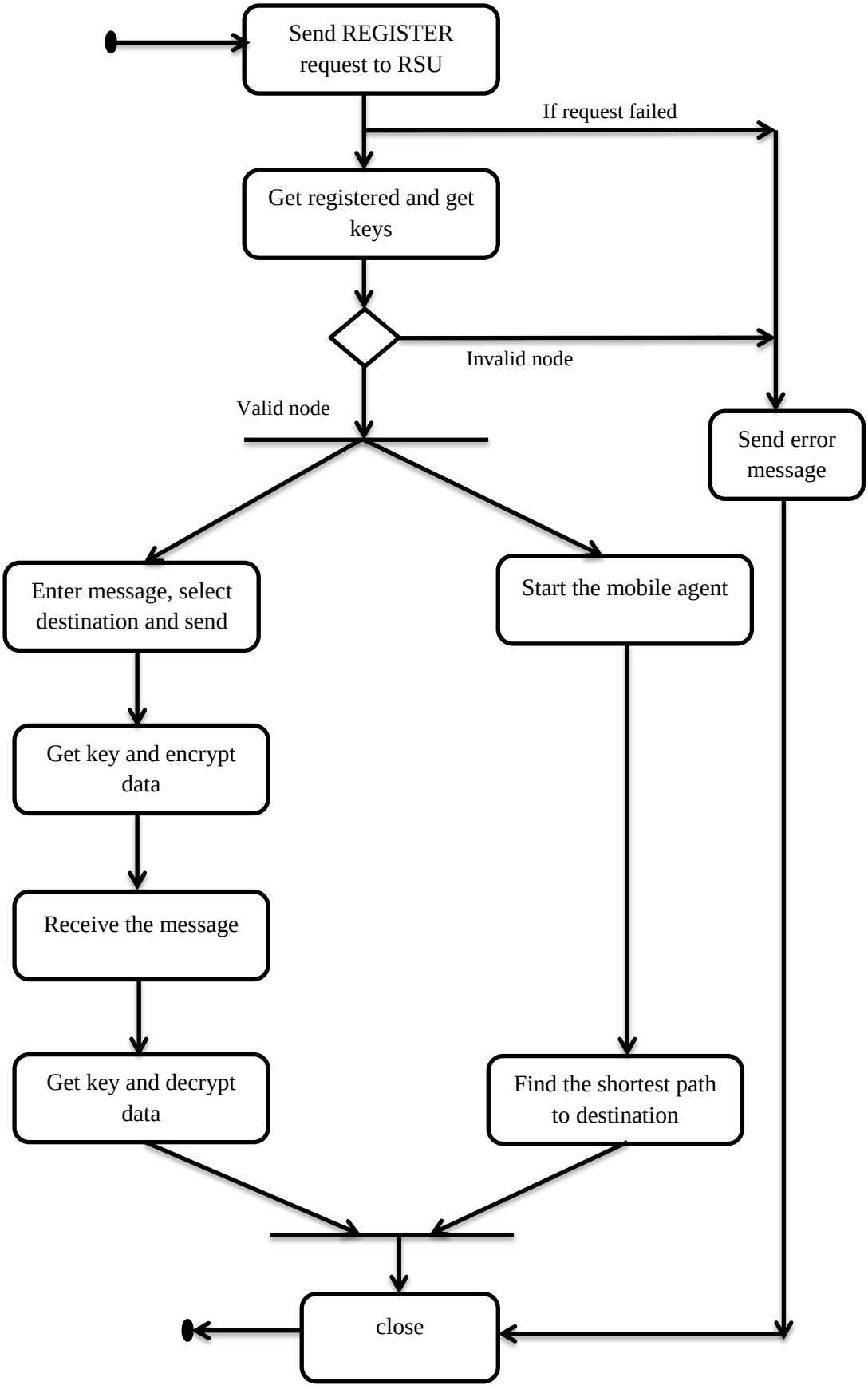


Figure 5.6: Sequence Diagram.

5.7 Activity Diagram

Activity diagram are graphical representation of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In UML, activity diagram can be used to describe operational step by step workflows of components in a system. Activity diagram shows overall flow of control. It mainly concentrates on operations rather than on objects.


Send REGISTER request to RSU
If request failed
Get registered and get keys
Invalid node
Valid node
Send error message
Enter message, select destination and send
Start the mobile agent
Get key and encrypt data
Receive the message
Get key and decrypt data
Find the shortest path to destination
close

Chapter 6

IMPLEMENTATION

6.1 Introduction

Implementation is the stage in the project where the theoretical design is turned into a working system and is giving confidence on the new system for the users, which it will work efficiently and effectively. It involves careful planning, investigation of the current System and its constraints on implementation, design of methods to achieve the changeover, an evaluation, of change over methods. Apart from planning major task of preparing the implementation are education and training of users. The more complex system being implemented, the more involved will be the system analysis and the design effort required just for implementation.

An implementation co-ordination committee based on policies of individual organization has been appointed. The implementation process begins with preparing a plan for the implementation of the system. According to this plan, the activities are to be carried out, discussions made regarding the equipment and resources and the additional equipment has to be acquired to implement the new system.

Implementation is the final and important phase, the most critical stage in achieving a successful new system and in giving the users confidence. That the new system will work be effective .The system can be implemented only after through testing is done and if it found to working according to the specification. This method also offers the greatest security since the old system can take over if the errors are found or inability to handle certain type of transactions while using the new system.

User Training

After the system is implemented successfully, training of the user is one of the most important subtasks of the developer. For this purpose user manuals are prepared and handled over to the user to operate the developed system. Thus the users are trained to operate the developed systems successfully in future .In order to put new application system into use, the following activities were taken care of:

- Preparation of user and system documentation

- Conducting user training with demo and hands on
- Test run for some period to ensure smooth switching over the system.

The users are trained to use the newly developed functions. User manuals describing the procedures for using the functions listed on menu and circulated to all the users .it is confirmed that the system is implemented up to user need and expectations.

6.2 Module: Registration

As seen in the network architecture, there will be four nodes and one Road Side Unit. The primary requirement for all the nodes to communicate is that they should be registered with the Road Side Unit. To implement this module, the nodes will contain a function which would read the text file named RSU.txt for the IP address of the RSU and calls a socket to communicate with the RSU module. The node would then set up data input and output streams and send REGISTER request to the RSU and waits for the reply.

The class readreq in the RSU handles the requests from the nodes. It would passively listen to the socket and perform according to the requests from the node. In this case, the node would send the REGISTER request which is read by the input stream in the RSU. The RSU then verifies the source from which the request is coming from and proceeds to key generation. It would call the functions of class ECC (which implements the Elliptic Curve Cryptography) and generate a public key and the private key for the node which sent the request. The RSU would then send the SUCCESS signal and subsequently send the keys back to the node. The public key is logged in RSU for the key distribution phase.

6.3 Module: Message Transfer and key distribution

When a node wants to communicate with another, the name of the node is selected; the message is either typed or retrieved from a file in the system. If either of the nodes isn't registered with the RSU while sending, the node would get an error message stating the problem and the message will not be sent. If both the sender and the receiver are registered with RSU, the sender node would then proceed to encrypt the message. The sender will send the GETKEY request and the name of the receiver node to the RSU. The RSU, upon processing the request, would reply back with the public key of the receiver. Using the sender's private key and the receiver's public key, the node would generate a one-time secret key called the shared key. Using that shared key, the message is

encrypted using the DES algorithm by calling the encrypt function of the DES class. Now that the message is ready to travel along the network, the forward agent would traverse through the nodes and would find out the shortest path to the receiver. The message travels from the sender node to the receiver node via intermediate nodes. Upon receiving the encrypted message, the receiver would check if the message is intended to it and then proceeds to decrypt the message. The receiver would then send the GETKEY request and the name of the sender node to the RSU. The RSU processes the request and replies back with the public key of the sender. The node would then generate a shared key using the sender's public key and the receiver's private key. If the message was truly intended to that node, both the sender and the receiver would arrive at the same shared key and thus the receiver would successfully decrypt the message by calling the decrypt function of the DES class. If any intermediate node tries to decrypt the message which wasn't intended to it, it would not be possible for that node to successfully decrypt the message since it is not possible to arrive at the same shared key the sender used to encrypt the message. Thus the messages are immune to any hacking attempts made to read the message and successfully reach the receiver.

6.4 Module: Road Side Unit

The Road Side Unit is the special unit we have included in our project which acts as a key generation and key distribution agent. Every node has to be registered with the RSU. The RSU will receive requests for registration and key distribution when the nodes want to communicate with one another. The RSU implements the Elliptic Curve Cryptography which is used to generate the keys for the nodes requesting for registration with the RSU. The RSU also maintains a log of all the nodes which are registered to it and their keys so that whenever the registered nodes request for key distribution, the RSU checks whether the requested node and the node which it is requesting for the keys are registered with the RSU by going through the logs and then would proceed to reply with the keys as requested. The RSU forms a very important and crucial part of the project and it is assumed that the RSUs are heavily immune to any hacking attempts as it is protected by a very thick firewall.

6.5 Module: Nodes

The nodes are nothing but the vehicles which are the part of the network of VANets. In this project, four nodes are implemented. The nodes are enclosed with methods implementing sending and receiving messages, forward and reverse agents, a class which encrypts/decrypts the message using DES algorithm and an intuitive GUI for the node enabling user to instantiate the functions of the node. The user can register with the RSU, start the mobile agent and even send messages to other nodes just by the press of a button and the subsequent function calls are made. Care has been taken to keep the GUI simple and efficient and also powerful enough to accommodate all the functionalities to be offered by the node.

6.6 Module: Mobile Agent

The mobile agent module is implemented as two sub-modules namely: the forward agent and the reverse agent. The role of the mobile agent is to run on an infinite loop finding the shortest path from the source node to the destination and also constantly update the cost of the neighboring nodes in the network.

The forward agent is implemented in the agent class in each node. The forward agent performs the route discovery process to the destination. The agent would connect to the IP address of its neighbors, connect to them one after the other and find the route to the destination node. The forward agent would create a stack and forward to the neighboring node and the cost and the dictionary stacks are updated till it reaches the destination. After the traversal, it would read the replies sent by the nodes, compare them and decide upon the shortest path to the destination.

The reverse agent is responsible for updating the cost from each node to its neighbors. In this project, the reverse agent is implemented in the ragent class. The ragent would run in an infinite loop, running continuously and thus be constantly aware of the changes in costs of the node. This is very important because, the VANets are mobile nodes and the costs between the nodes are constantly changing and ragent makes sure that it lives competent to the changes in the costs of the nodes. It would connect to its neighbors by reading its IP addresses and broadcast its cost and dictionary information. The other nodes would read the information packet send by this node, update their routing tables, cost and the dictionary stack and reply back with the updated information in the

stack. This goes till all the four nodes are traversed. Thus, every node will be updated with the number of nodes participating in the network and their costs from the current node.

6.7 Key generation and message encryption

The Road Side Unit assigns keys to every node that registers to it. To implement the key generation, we employ the Elliptic Curve Cryptography (ECC)**.** ECC is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. To implement the Elliptic Curve Cryptographic scheme, we create a new class ecc, which consists of methods necessary to implement this scheme.

Key Generation

Key generation is an important part, generating both public key and private key. The sender will be encrypting the message with receiver's public key and the receiver will decrypt its private key.

We have to select a prime number 'd' within the range of **'**n**'**.
Using the following equation we can generate the public key

Q = d * P **[6.1]**

d = the random number that we have selected within the range of (1 **to n-1**). **P** is the point on the curve.
'Q' is the public key and 'd' is the private key.

Encryption

The messages to be sent from the sender to the receiver are encrypted using Data Encryption Standard (DES). The algorithm described in this standard specifies both enciphering and deciphering operations which are based on a binary number called a key. Data can be recovered from cipher only by using exactly the same key used to encipher it. Unauthorized recipients of the cipher who know the algorithm but do not have the correct key cannot derive the original data algorithmically.

In this project, we generate a "shared key" which acts as a key for encryption and decryption using DES. However, by using ECC, we employ a mechanism wherein the shared key is generated in the sender node using the public key of the receiver and the

private key of the sender. On the receiver side, the key is again calculated by taking the public key of the sender and the private key of the receiver and arriving at the same key which decrypts the message. This is possible because the keys are generated using a point on the elliptic curve. Any unauthorized node will not be successful in deciphering the message since it cannot arrive at the common shared key.

Let 'M' be the message that is being sent. Let Pk_S be the private key of the sender and Pu_R be the public key of the receiver node. If the shared key is denoted by S, then the generated key for encryption is given by

$$\mathbf{S = Pk_S * Pu_R} \quad \textbf{[6.2]}$$

The encrypted message E is given by

$$\mathbf{E = M + Pk_S * Pu_R} \quad \textbf{[6.3]}$$

Decryption

In the receiver side, message decryption happens as shown below.

Let E be the encrypted message received by the receiver which is generated by the sender using [6.3]. To decrypt the message, the receiver node arrives upon the shared key by taking the public key of the sender Pu_S and the private key of the receiver Pr_R which is given by,

$$\mathbf{S = Pu_S * Pr_R} \quad \textbf{[6.4]}$$

$$\mathbf{M = E - (Pu_S * Pr_R)} \quad \textbf{[6.5]}$$

Thus we get back the message M. From this, the encryption and decryption of messages to be sent over the network is achieved without actually transmitting the key alongside the message and also it is immune to any hacking attempts made to decrypt the message.

6.8 Socket Connection

Before an application program (client or server) can transfer any data, it must first create an end point for communication by calling socket. Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

The client and server interactions occur as follows:

- The server, when willing to offer its advertised services, binds a socket to a well-known address associated with the service, and then passively listens on its socket. It is then possible for an unrelated process to rendezvous with the server.
- The server process socket is marked to indicate incoming connections are to be accepted on it.
- The client requests services from the server by initiating a connection to the server's socket. The client process uses a **connect** subroutine to initiate a socket connection.
- If the client process socket is unbound at the time of the **connect** call, the system automatically selects and binds a name to the socket if necessary. This is the usual way that local addresses are bound to a socket.
- The system returns an error if the connection fails (any name automatically bound by the system, however, remains). Otherwise, the socket is associated with the server and data transfer can begin.

No.	Constructor/ Method	Description
	ServerSocket Class	
1	ServerSocket(int port)	Constructs an object of class ServerSocket and binds the object to the specified port – to which all clients attempt to connect.
2	accept( )	This is a blocking method call – the server listens (waits) for any incoming client connection request and cannot proceed further unless contacted by a client. When a client contacts, the method is unblocked and returns a Socket object to the server program to communicate with the client.
3	close( )	Closes the ServerSocket object
4	void setSoTimeout(int timeout)	The ServerSocket object is set to listen for an incoming client request, under a particular invocation of the accept ( ) method on the object, for at most the milliseconds specified in "timeout". When the timeout expires, a java.net.SocketTimeoutException is raised. The timeout value must be > 0; a timeout value of 0 indicates infinite timeout.
	Socket Class	
5	Socket(InetAddress host, int port)	Creates a stream socket and connects it to the specified port number at the specified IP address

6	InetAddress getInetAddress( )	Returns the IP address at the remote side of the socket
7	InetAddress getLocalAddress( )	Returns the IP address of the local machine to which this socket is bound.
8	int getPort( )	Returns the remote port number to which this socket is connected.
9	int getLocalPort()	Returns the local port number to which this socket is bound
10	InputStream getInputStream( )	Returns an input stream for this socket to read data sent from the other end of the connection.
11	OutputStream getOutputStream( )	Returns an output stream for this socket to send data to the other end of the connection
12	close( )	Closes this socket
13	void setSoTimeout(int timeout)	Sets a timeout value to block on any read( ) call on the Input Stream associated with this socket object. When the timeout expires, a java.net.SocketTimeoutException is raised. The timeout value must be > 0; a timeout value of 0 indicates infinite timeout.

Table 5.1 Key Commonly Used Methods of the Stream-Mode Socket API

CHAPTER 7

TESTING

Testing is an important phase in the development life cycle of the product; this was the phase where the error remaining from all the phases was detected. Hence testing performs a very critical role for quality assurance and ensuring the reliability of the software. The testing determines the program reliability of the software. During the testing, the program to be tested was executed with a set of test cases and the output of the program for the test cases was evaluated to determine whether the program is performing as expected. Errors were found and corrected by using the following testing steps and correction was recorded for future references. Thus, a series of testing was performed on the system before it was ready for implementation.

It is the process used to help identify the correctness, completeness, security, and quality of developed computer software. Testing is a process of technical investigation, performed on behalf of stake holders, i.e. intended to reveal the quality-related information about the product with respect to context in which it is intended to operate. This includes, but is not limited to, the process of executing a program or application with the intent of finding errors. The quality is not an absolute; it is value to some person. With that in mind, testing can never completely establish the correctness of arbitrary computer software; Testing furnishes a 'criticism' or comparison that compares the state and behavior of the product against specification. An important point is that software testing should be distinguished from the separate discipline of software quality assurance (SQA), which encompasses all business process areas, not just testing.

There are many approaches to software testing, but effective testing of complex products is essentially a process of investigation not merely a matter of creating and following routine procedure. One definition of testing is "The process of questioning a product in order to evaluate it", where the "questions" are operations the tester attempts to execute with the product, and the product answers with its behavior in reaction to the probing of the tester. Although most of the intellectual processes of testing are nearly identical to that of review or inspection, the word testing is connoted to mean the dynamic analysis of the product-putting the product through its paces.

Some of the common quality attributes include capability, reliability, efficiency, portability, maintainability, compatibility and usability. A good test is sometimes

described as one, which reveals an error; however, more recent thinking suggest that a good test is one which reveals information of interest to someone who matters within the project community.

7.1 Types of Tests

7.1.1 Unit Testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produces valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

7.1.2 Functional Testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input	: identified classes of valid input must be accepted.
Invalid Input	: identified classes of invalid input must be rejected.
Functions	: identified functions must be exercised.
Output	: identified classes of application outputs must be exercised.
Systems/Procedures	: interfacing systems or procedures must be invoked.

7.1.3 System Testing

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

7.1.4 Performance Testing

The Performance test ensures that the output is produced within the time limits and the time taken by the system for compiling, giving response to the users and request being send to the system for to retrieve the results.

7.1.5 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Integration testing for Server Synchronization:

- Testing the IP Address for nodes to communicate with the other Nodes.
- Check the request is sent from peer node to RSU.
- Check the key parts are sent from RSU to the sender and the receiver.
- Check the mobile agent to traverse through its neighbors and find the route to the destination.
- Check the encrypted message is forwarded to receiver.

7.1.6 Validation Testing

The validation testing can be defined in many ways, but a simple definition is that, validation succeeds when the software functions in a manner that can be reasonably expected by the end user.

a) Black Box Testing

Black box testing is done to find the following

- Incorrect or missing functions
- Interface errors
- Errors in external database access
- Performance error
- Initialization and termination error

b) White Box Testing

This allows the tester to

- Check whether all independent paths within a module have been exercised at least once.
- Exercise all logical decisions on their and false sides
- Execute all loops and their boundaries and within their bounds
- Exercise the internal data structure to ensure their validity
- Ensure whether all the possible validity checks and validity lookups have been provided to validate data entry.

7.1.7 Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

MODULE	GIVEN INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	RESULT
Sender	Register in RSU	Registered	Registered	OK
RSU	REGISTER request from node	Keys updated successfully	Keys updated successfully	OK
Sender	GETKEY request to RSU	Public key of the receiver node from RSU	Public key of the receiver node from RSU	OK
Mobile Agent	Find the shortest path to the destination	Destination reached via shortest path	Destination reached via shortest path	OK
Receiver	GETKEY request to RSU	Public key of the sender from RSU	Public key of the sender from RSU	OK

Mobile Agent	Update the routing table	Routing table updated with changing costs of node	Routing table updated with changing costs of node	OK
Sender	Message	Encrypt using shared key	Message encrypted	OK
Receiver	Encrypted Message	Decrypt using shared key	Message Decrypted	OK

Table 7.1: Testing Results

7.2 Test Cases

In order to fully test that all the requirements of an application are met, there must be at least two test cases for each requirement: one positive test and one negative test. If a requirement has sub-requirements, each sub-requirement must have at least two test cases. Keeping track of the link between the requirement and the test is frequently done using a traceability matrix. Written test cases should include a description of the functionality to be tested, and the preparation required to ensure that the test can be conducted.

A formal written test-case is characterized by a known input and by an expected output, which is worked out before the test is executed. The known input should test a precondition and the expected output should test a post condition.

TEST CASE ID: UT_1	TESTTITLE: Socket Connection
The following steps have to be followed to carry out test: • Open command prompt. • Type ipconfig (to get the IP address). • Ping address.	
Expected Result: Socket connection established.	Results: Pass.

Table 7.2: Socket Connection

TEST CASE ID: UT_2	TESTTITLE: Start the nodes and RSU
The following steps have to be followed to carry out test: • Start the nodes and RSU's batch file. • Start.	
Expected Result: The nodes and RSU window should start.	Results: Pass.

Table 7.3: Running the module

TEST CASE ID: UT_3	TESTTITLE: Register in RSU
The following steps have to be followed to carry out test. • Press the 'Register with RSU' button in the each node. • After which keys are generated using elliptic curve cryptography and updated in the RSU table. Press ok.	
Expected Result: Nodes get registered and keys are displayed.	Results: Pass.

Table 7.4: Registering with RSU

TEST CASE ID: UT_4	TEST TITLE: Starting Mobile Agent
The following steps have to be followed to carry out test. • Press the 'Start Agent' button in the respective nodes. • It starts finding the shortest path and is displayed in the stack area.	
Expected Result: Messages will be sent from the sender using shortest path to the destination	Results: Pass.

Table 7.5: Mobile Agent

TEST CASE ID: UT_5	TEST TITLE: Sending the message
The following steps have to be followed to carry out test. • Write the text in the message area or select the files using browse option. • Select the destination node. • Press 'send' button. • Message processing can be seen in the process field.	
Expected Result: Message sent successfully	Results: Pass.

Table 7.6: Message Sending

TEST CASE ID: UT_6	TEST TITLE: Key Distribution by RSU
The following steps have to be followed to carry out test. • When 'send' button is pressed, the sender verifies the receiver node with RSU. • If the node is valid then RSU replies back with the public key of receiver and then the data is sent. • At the receiver end, receiver verifies the source node with RSU. If valid it gets the public key of the sender.	
Expected Result: Valid nodes which get the keys should be able to communicate.	Results: Pass.

Table 7.7: Key Distribution

TEST CASE ID: UT_7	TEST TITLE: Encryption and decryption
The following steps have to be followed to carry out test. • When the sender gets the public key of receiver, it generates a shared key to encrypt the data. This data will be send. • When this encrypted data reaches the receiver, it uses the public key of the sender and generates the shared key. Using which the data is decrypted. • For encryption and decryption, Data Encryption Standard (DES) is used.	
Expected Result: Encrypted message is sent on the network without having to send the key alongside the message	Results: Pass.

Table 7.8: Encryption and decryption

TEST CASE ID: UT_8	TEST TITLE: Communication with an unregistered node
The following steps have to be followed to carry out test. • Choose a node which is not yet registered with the RSU as the destination. • Type in the message and press the send button.	
Expected Result: Error message stating the node is not registered with RSU.	Results: Pass.

Table 7.9: Node not registered with RSU

CONCLUSION

The convergence of computing, telecommunications (fixed and mobile), and various kinds of services are enabling the deployment of different kinds of VANET technologies. In the past decade, many VANET projects around the world have been undertaken and several VANET standards have been developed to improve vehicle-to-vehicle or vehicle-to infrastructure communications. We reviewed some of the main areas that researchers have focused on in the last few years and these include security, routing, QoS, and broadcasting techniques and we highlighted the most salient results achieved to date.

In this project, an attempt has been made to provide security for communication among nodes in VANets. A Road Side Unit, which acts as the key distribution agent, registers new nodes entering the network, distributing keys for nodes trying to communicate and also restricting malicious nodes and ones which tries to establish an unauthorized connection. Care has been taken to ensure that the security does not come with a price of high communication overhead. Powerful cryptographic algorithms are used to make cryptanalysis difficult. VANET security is an emerging area. As different VANET protocols and applications are based on different assumptions, a common evaluation framework is needed to compare different security research contributions.

FUTURE ENHANCEMENTS

As with other applications, there is certainly a scope for improvement in this application too. New modules are in pipeline for to increase the compatibility of the project. Once these improvements have been done, the majority of the features that make an application an excellent one would be there and the usage would become wider and more expensive. Here, there are some of decisions for to make our project effectively and efficiently in the future.

- Improvise the Co-operative Message Authentication protocol to further reduce the overhead in communication.
- Implement an online facility which retrieves information from the RSU to monitor the vehicular movements.
- Implement an intelligent protocol in which one RSU informs the other about malicious nodes and completely disbands it from the network.
- Compress the messages being sent and further optimize the message transmission.

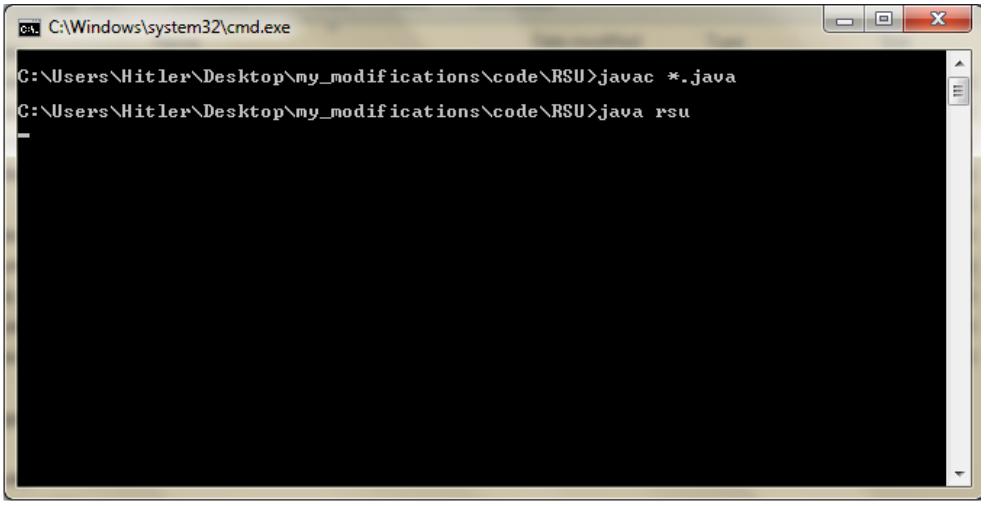
REFERENCES

- J. Blum and A. Eskandarian, "The threat of intelligent collisions", IT Professional, vol. 6, no. 1, pp. 24–29, 2004.
- R. Chen, D. Ma, and A. Regan, "TARI: Meeting delay requirements in VANETs with efficient authentication and revocation", In Proceedings of WAVE, 2009.
- X. Lin, X. Sun, P. Ho, and X. Shen, "GSIS: A secure and privacy preserving protocol for vehicular communications", IEEE Trans. Veh. Technol., vol. 56, no. 6, pp. 3442–3456, 2007.
- Perrig, R. Canetti, D. Tygar, D. Song, "The TESLA broadcast authentication protocol", CryptoBytes, vol. 5, 2002.
- M. Raya and J. Hubaux, "Securing vehicular ad hoc networks", J. Comput. Secur., vol. 15, no. 1, pp. 39–68, 2007.
- Studer, E. Shi, F. Bai, and A. Perrig, "TACKing Together efficient authentication revocation, and privacy in VANETs", In Proceedings of SECON, pp. 22–26, 2009.
- Zhang, X. Lin, R. Lu, P.-H. Ho, and X. Shen, "An efficient message authentication scheme for vehicular communications", IEEE Trans. Veh. Technol., vol. 57, no. 6, pp. 3357–3368, 2008.
- Zhang, R. Lu, X. Lin, P.-H. Ho and X. Shen, "An efficient identity-based batch verification scheme for vehicular sensor networks", In Proceedings of INFOCOM, 2008.
- L. Zhang, Q. Wu, A. Solanas,D.-F. Joseph, "A scalable robust authentication protocol for secure vehicular communications", IEEE Trans. Veh. Technol., vol. 59, no. 4, pp. 1606–1617, 2010.

Appendix-A

SCREENSHOTS

- **Running the RSU:**



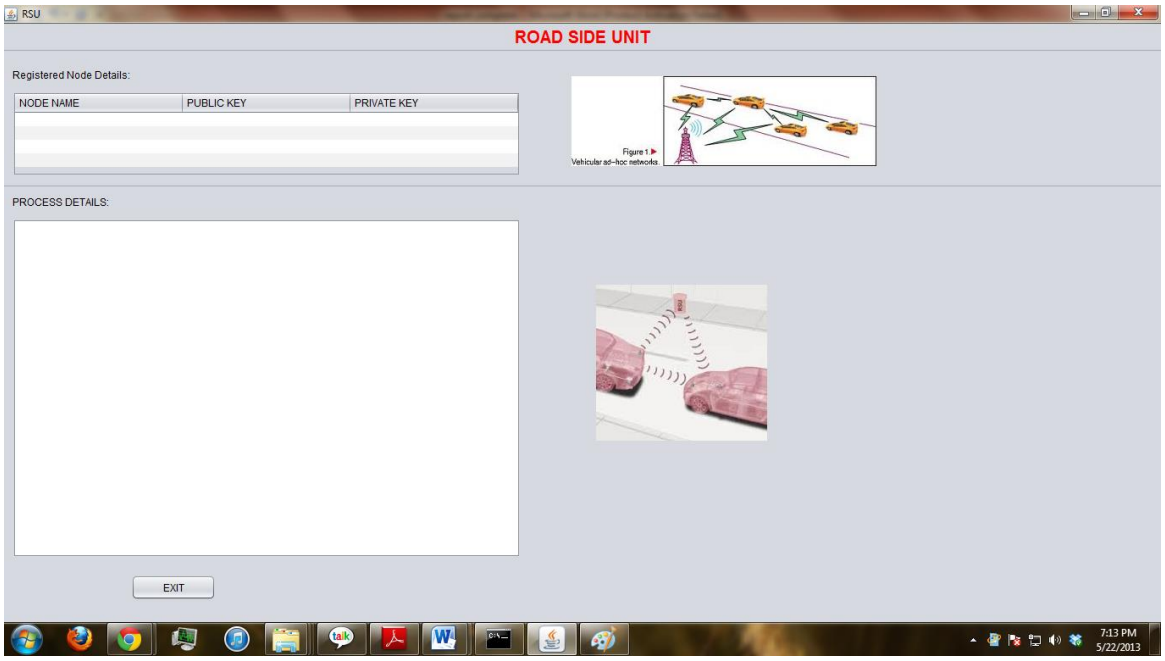


Figure A.1: Road Side Unit (RSU)

➢ Running the nodes

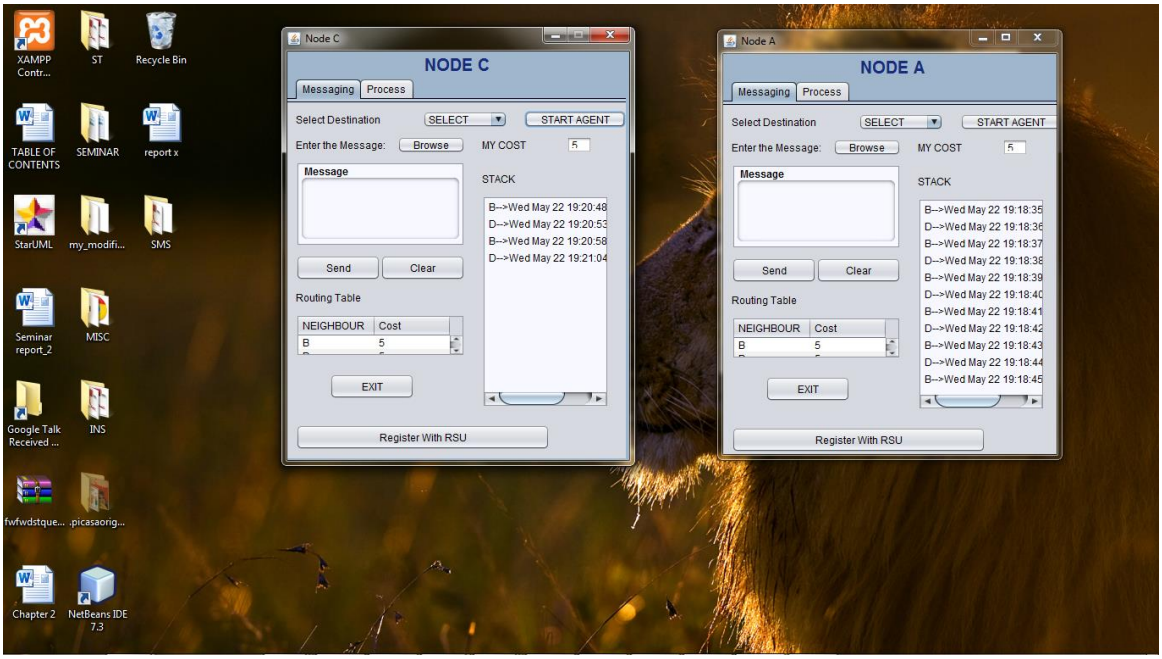


Figure A.2: Node A and Node C

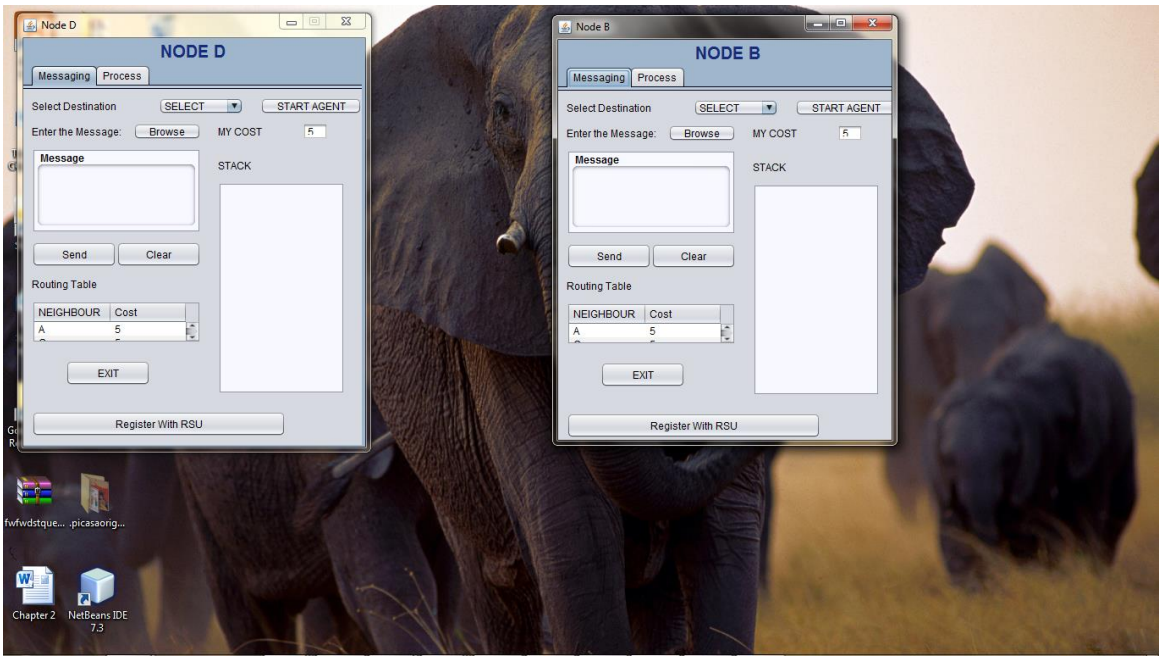
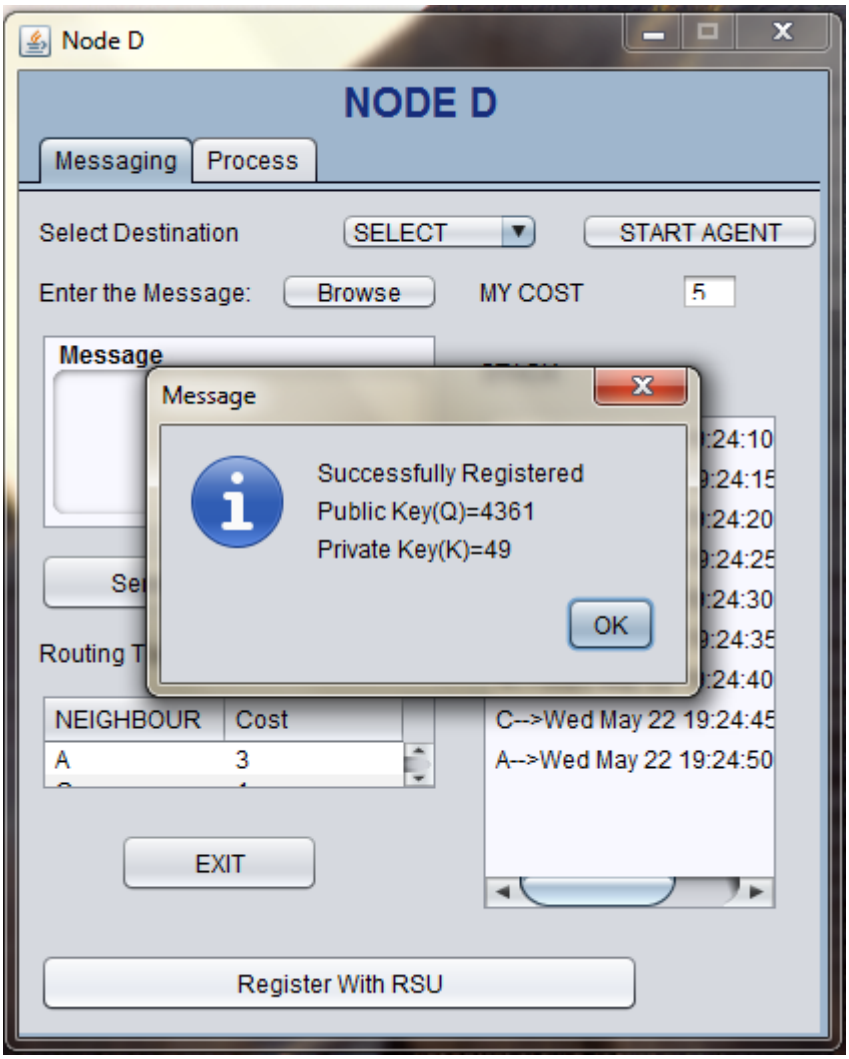


Figure A.3: Node B and Node D

- **Node Registration**



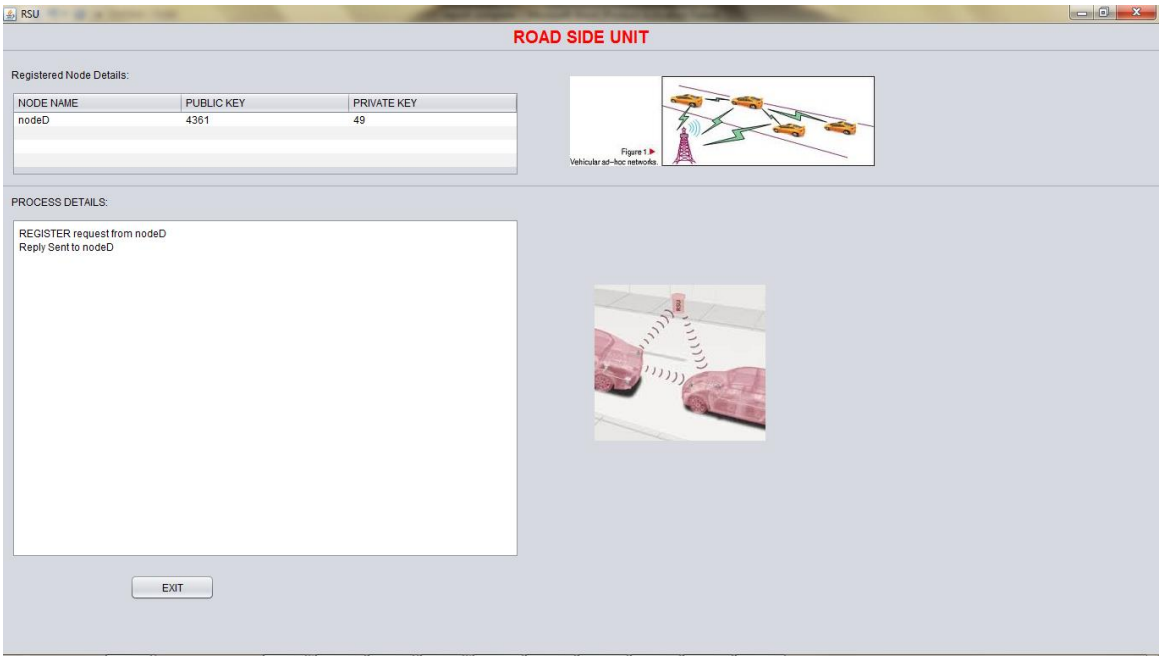


Figure A.4: Successful Registration of Node D

- **Message sending**

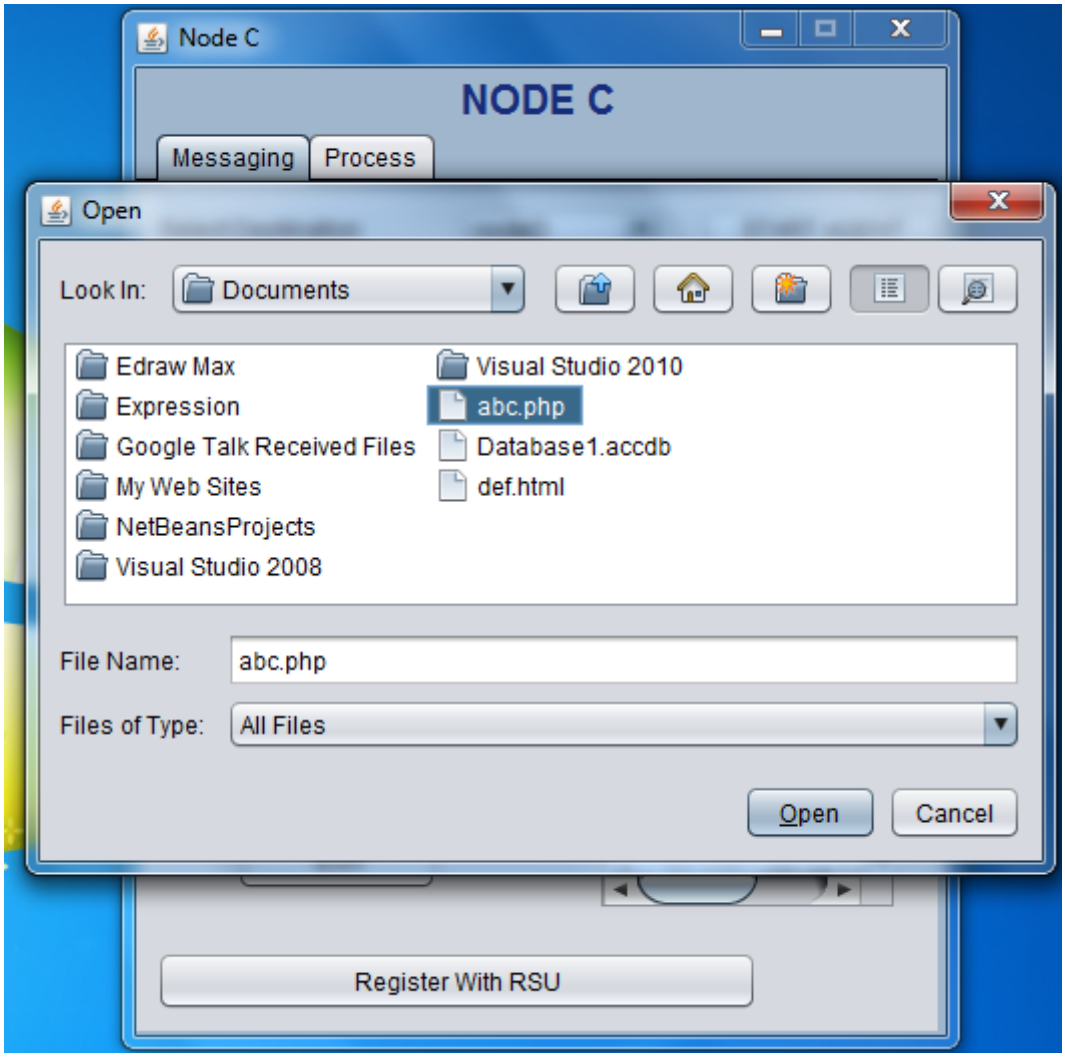


Figure A.5: Choosing a file content to be sent

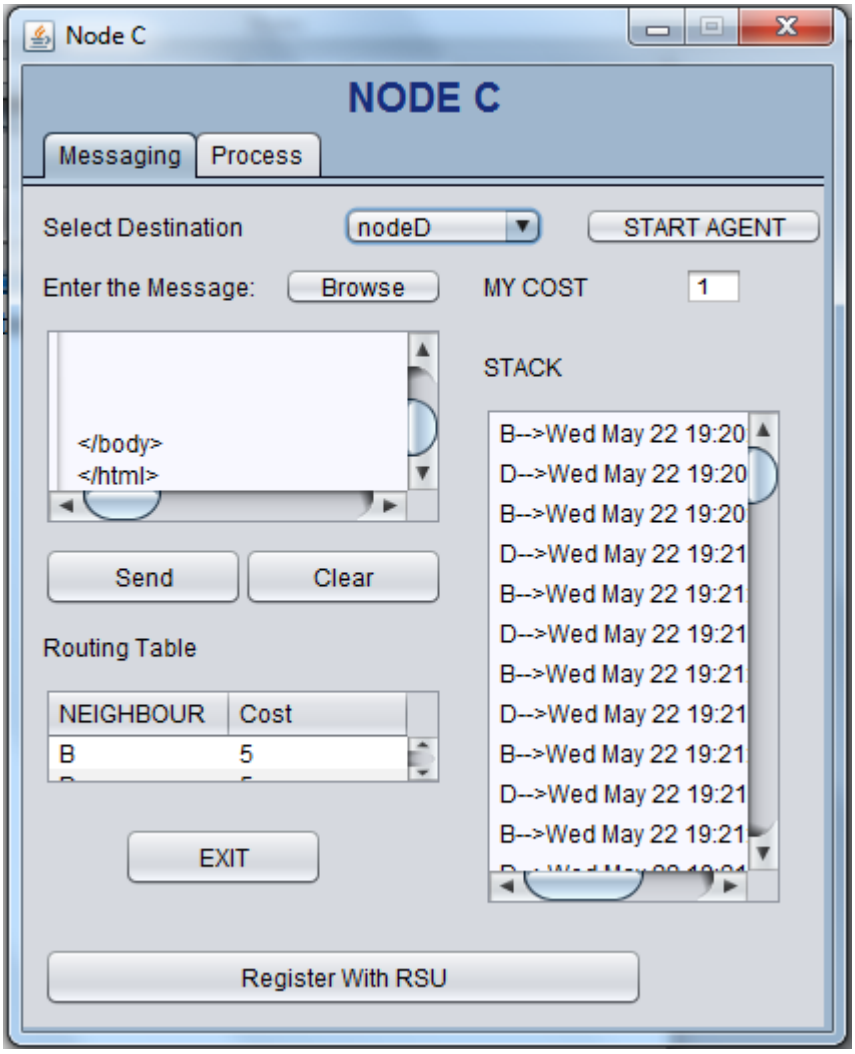


Figure A.6: Choosing the destination

- **Message Sending and Key Distribution (Sender node)**

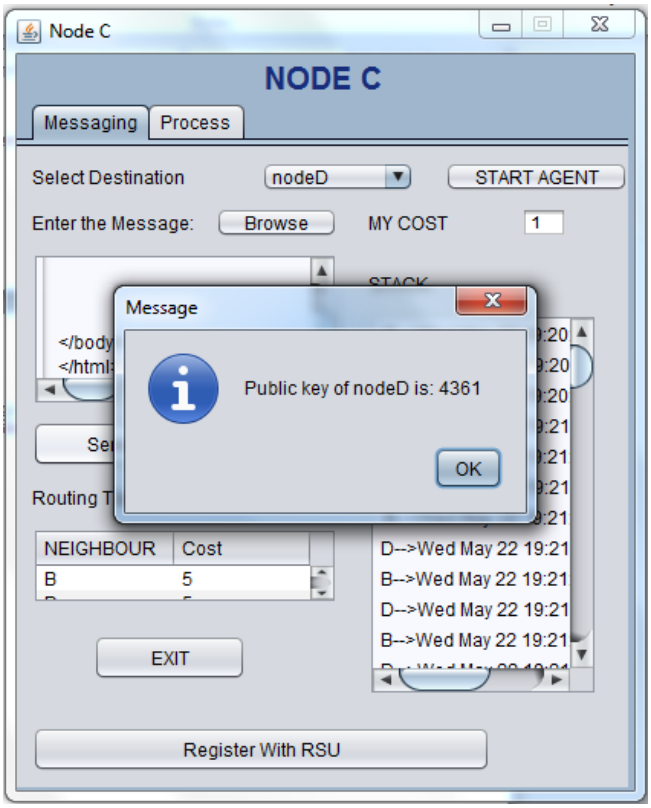


Figure A.7: Retrieving key from RSU

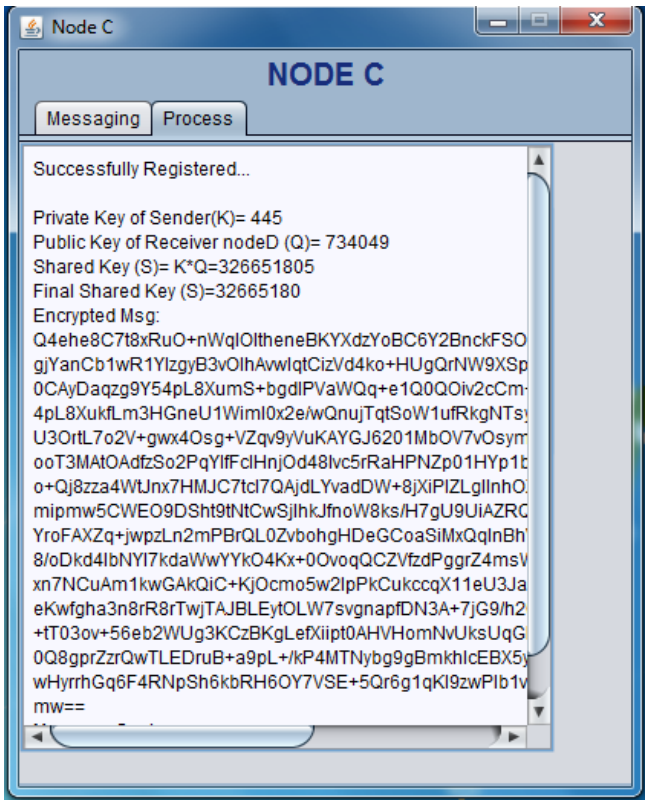


Figure A.8: Encrypted message sent over the network

- **Message Sending and Key Distribution (Receiver node)**

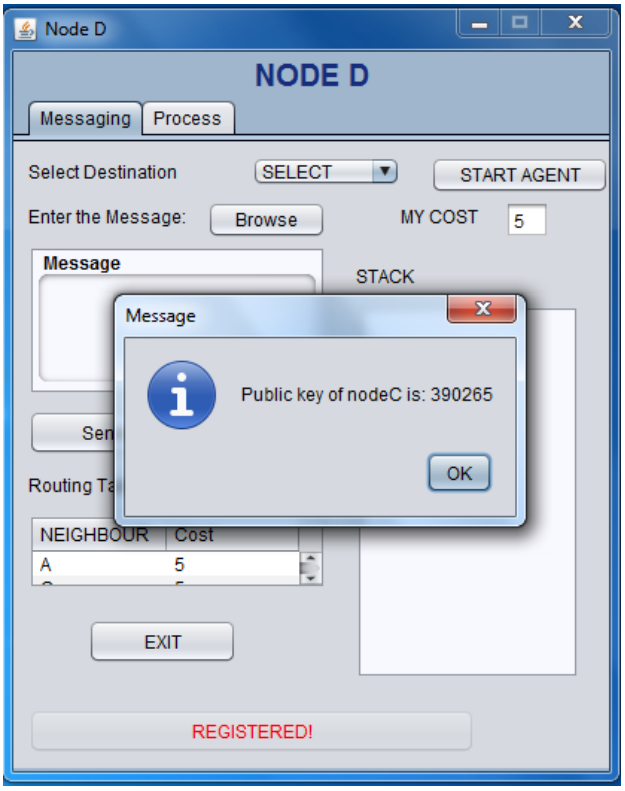


Figure A.9: Retrieving sender's key from RSU

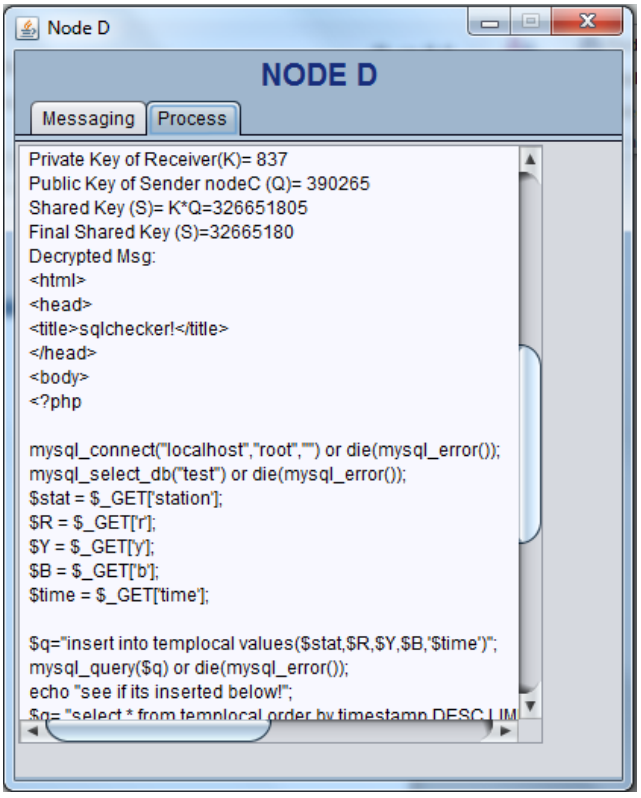


Figure A.10: Decrypted message

Appendix-B

CODE SNIPPETS

- **Key generation using ECC**

```
import java.lang.Math.*;

class ecc
{
int G=0;
int Ka=0,Kb=0;

ecc()
{

        G=generateG();
        Ka=generateK(G);
        Kb=generateK(G);
        int Qa=Ka*G;
        int Qb=Kb*G;

        int KaQb=Ka*Qb;
        int KbQa=Kb*Qa;
        int KaKbG=Ka*Kb*G;

}

int generateG()
{
int Gtemp=0;
        try
        {
                while(true)
                {

                        Gtemp=(int)(Math.random()*1000);

                        if (prime(Gtemp) && Gtemp>20)
                        break;
                }
```

```
        }
        catch(Exception e)
        {
                System.out.println(e);
        }

        return Gtemp;
}

boolean prime(int num)
{
        boolean result=true;

        int i=0;
        for (i=2;i<num;i++)
        if ((num%i)==0)
        break;

        if (i==num)
        result=true;
        else
        result=false;



        return result;

}


int generateK(int Gtemp)
{
        int ktemp=0;
        try
        {
                while(true)
                {
                        ktemp=(int)(Math.random()*Gtemp);
                        if (ktemp>1)
                        break;
                }


        }
```

```
            catch(Exception e)
            {
                  System.out.println(e);
            }

            return ktemp;
      }


      public static void main(String args[])
      {
            ecc e=new ecc();

      }

}
```

- **Registration request to RSU from the node**

```
      void register()
      {
            try
            {
                  Socket soc=new Socket(rsuaddr,10000);
                  DataOutputStream dout=new
                  DataOutputStream(soc.getOutputStream());
                  DataInputStream din=new DataInputStream(soc.getInputStream());

                  dout.writeUTF("REGISTER");
                  dout.writeUTF("nodeA");
                  String reply=din.readUTF();

                  if (reply.equals("SUCCESS"))
                  {
                        k=din.readInt();
                        q=din.readInt();
                        JOptionPane.showMessageDialog(this,"Successfully
                        Registered\nPublic Key(Q)="+q+"\nPrivate Key(K)="+k);
                  }
                  else
                  JOptionPane.showMessageDialog(this,"Registration Failed");
            }
            catch(Exception e)
            {
                  System.out.println(e);
```

```
        }
    }
```

- **Nodes sending GETKEY request to RSU**

```
    boolean getkey(String dest)
        {
            boolean reply=false;
            try
            {
                if (k==0)
                JOptionPane.showMessageDialog(this,"U R Not Registered
with RSU!");
                else
                {
                    Socket soc=new Socket(rsuaddr,10000);
                    DataOutputStream dout=new
                    DataOutputStream(soc.getOutputStream());
                    DataInputStream din=new
                    DataInputStream(soc.getInputStream());

                    dout.writeUTF("GETKEY");
                    dout.writeUTF(dest);

                    reply=(Boolean)din.readBoolean();
                    if (reply)
                    {
                        qdest=(int)din.readInt();

        JOptionPane.showMessageDialog(this,"Public key of "+dest+" is:
"+qdest);
                    }
                    else
                    JOptionPane.showMessageDialog(this,dest+" is Not
Registered in RSU ");
                }
            }
            catch(Exception e)
            {
                System.out.println(e);
            }

            return reply;
    }
```

- **RSU processing the REGISTER and GETKEY requests from nodes**

```
public void run()
        {
                try
                {
                        ServerSocket ss=new ServerSocket(10000);

                        while (true)
                        {
                                Socket soc=ss.accept();
                                DataInputStream din=new
                                DataInputStream(soc.getInputStream());
                                DataOutputStream dout=new
                                DataOutputStream(soc.getOutputStream());

                                String req=din.readUTF();

                                if (req.equals("REGISTER"))
                                {
                                        String node=din.readUTF();
                                        obj.jta.append(req+" request from
                                        "+node+"\n");
                                        int k=obj.e.generateK(obj.g);
                                        int q=k*obj.g;
                                        addinfo(node,q,k);
                                        dout.writeUTF("SUCCESS");
                                        dout.writeInt(k);
                                        dout.writeInt(q);
                                        obj.jta.append("Reply Sent to "+node+"\n");
                                }
                                else
                                if (req.equals("GETKEY"))
                                {
                                        String node=(String) din.readUTF();
                                        int  key=0;
                                        boolean exist=false;
                                        for (int i=0;i<obj.row;i++)
                                        {
                                                String
                                                n=obj.dft.getValueAt(i,0).toString().t
                                                rim();
                                                if (n.equals(node))
                                                {
```

```
                key=Integer.parseInt(obj.dft.getValueAt(i,1).toString().trim());
                                                exist=true;
                                                break;
                                        }
                                }

                                if (exist)
                                {
                                        dout.writeBoolean(true);
                                        dout.writeInt(key);
                                }
                                else
                                dout.writeBoolean(false);
                        }


                        din.close();
                        dout.close();
                        soc.close();

                }
        }
        catch(Exception e)
        {
                System.out.println(e);
        }
}
```