ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

FACULTY OF TECHNOLOGY

ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT

THREATS AND TRUSTED COUNTERMEASURES, USING A SECURITY PROTOCOL, IN THE AGENT SPACE

By

Tinbit Admassu

Advisor

Prof. Dr. Sayed Nouh

A thesis submitted to the school of Graduate studies of Addis Ababa University in partial fulfillment of the requirements for the degree of

Masters of Science in Computer Engineering

April 2008

Addis Ababa, Ethiopia

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

FACULTY OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

THREATS AND TRUSTED COUNTERMEASURES, USING A SECURITY PROTOCOL, IN THE AGENT SPACE

By

Tinbit Admassu

Advisor
Prof. Dr. Sayed Nouh

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

THREATS AND TRUSTED COUNTERMEASURES, USING A SECURITY PROTOCOL, IN THE AGENT SPACE

By

Tinbit Admassu

FACULTY OF TECHNOLOGY

APPROVAL BY BOARD OF EXAMINERS

Dr. Mengesha Mamo	_____
Chairman Dept. of Graduate Committee	Signature
Prof. Dr. Sayed Nouh	_____
Advisor	Signature
_____	_____
Internal Examiner	Signature
_____	_____
External Examiner	Signature

Declaration

I, the undersigned, declare that this thesis work is my original work, has not been presented for a degree in this or any other universities, and all sources of materials used for the thesis work have been fully acknowledged.

Name: Tinbit Admassu Signature: _____

Place: Addis Ababa

Date of submission:

This thesis has been submitted for examination with my approval as a university advisor.

Prof. Dr. Sayed Nouh Signature: _____

Advisor's Name

ACKNOWLEDGEMENT

First of all I thank The Almighty God for letting me finish this thesis. This thesis would not have been successful without His help.

I would like to express my appreciation to my Advisor Prof. Dr. Sayed Nouh for his invaluable support, starting from his willingness to take on this research idea from its inception to the overall support, guidance, suggestions and encouragement throughout the course of the work. I would also like to give my gratitude to Dr. Mesfin Belachew for his helpful ideas during the implementation phase of the work.

I am grateful to all my friends and other people who have been around encouraging me to successfully finish this thesis. Special thanks goes to my families, without whom things would have been so different.

TABLE OF CONTENTS

ACKNOWLEDGEMENT i
TABLE OF CONTENTS ii
LIST OF FIGURES v
LIST OF ACRONYMS vi
ABSTRACT vii
Chapter 1 1
INTRODUCTION 1
1.1. Background 2
1.2. Statement of the Problem 3
1.3. Objectives 3
1.4. Scope 4
1.5. Methodology 4
1.6. Organization of the Thesis 5
Chapter 2 6
AGENTS AND MOBILE AGENTS CONCEPTS 6
2.1. Background 6
2.2. Network Computing Paradigms 9
2.2.1. Client – Server 9
2.2.2. Remote Evaluation 10
2.2.3. Code on Demand 10
2.2.4. Mobile Agent 11
2.3. Java Support for Mobile Agent Computing Paradigm 12

Chapter 3 14
THREATS AND COUNTERMEASURES 14
3.1. Introduction 14
3.2. Hostile Host Threats 15
3.2.1. Masquerading 15
3.2.2. Denial of Service 15
3.2.3. Eavesdropping 15
3.2.4. Alteration 16
3.3. Countermeasures for Malicious Host Threats 16
3.3.1. Trusted Hardware 16
3.3.2. Trusted Execution Environment 17
3.3.3. Computing with Encrypted Functions 17
3.3.4. Code Obfuscation 18
3.3.5. Path Histories 19
3.3.6. Mutual Itinerary Recording 19
3.3.7. Itinerary Recording with Replication and Voting 20
Chapter 4 21
PROPOSED COUNTERMEASURE 21
4.1. Introduction 21
4.2. Design Guidelines 22
4.3. The Proposed Countermeasure 23
4.4. Components of the Proposed Countermeasure 27
4.5. Security Protocol 29
4.6. Proposition Evaluation 35
Chapter 5 37
IMPLEMENTATION AND RESULTS 37
5.1. Overview 37
5.2. Introduction to Tools Used 37
5.2.1. MAS Familiarization 37

5.2.2. Cryptographic Service Providers 42
5.3. Implementation 43
5.3.1. Utility Classes 43
5.3.2. Components of the Prototype Application 45
5.4. Results and Performance Comparison 60
Chapter 6 64
CONCLUSION AND FUTURE WORK 64
REFERENCES 66
APPENDIX A: 68
INSTALLING AGLETS PLATFORM 68
APPENDIX B: 69
INSTALLING PROVIDERS 69
APPENDIX C: 70
SAMPLE SOURCE CODES 70

LIST OF FIGURES

Figure 2.1 A mobile agent system 8
Figure 2.2 Client-server computing model 9
Figure 2.3 Remote evaluation 10
Figure 2.4 Code on demand computing model 11
Figure 2.5 Mobile agent computing model 11
Figure 2.6 Detailed view of mobile agent platform 13
Figure 4.1 Original mobile agent computing model 24
Figure 4.2 Proposed mobile agent computing model. 25
Figure 4.3 Security protocol at Home 30
Figure 4.4 Security protocol at TS. 31
Figure 4.5 Security protocol at the i^{th} host. 32
Figure 4.6 Security protocol back at TS. 33
Figure 4.7 Security protocol back at Home. 34
Figure 5.1 The Aglet architecture 38
Figure 5.2 Depicting the aglet's life cycle 40
Figure 5.3 Methods of Destn class 43
Figure 5.4 Methods of ProxyH class 44
Figure 5.5 Methods of CipherCls class 45
Figure 5.6 The path dialog box 55
Figure 5.7 Results dialog box 56
Figure 5.8 Test environment set up 61
Figure 5.9 Captured packet analysis for DS MA and Normal MA. 61
Figure 5.10 Captured packet analysis for Proposed MA. 62
Figure 5.11 Simulating nodes using ports 63
Figure 5.12 Detection of alteration threat 63

LIST OF ACRONYMS

MAS	Mobile Agent System
MAP	Mobile Agent Platform
MA	Mobile Agent
ABA	Agent Based Application
TS	Trusted Server
JVM	Java Virtual Machine
SA	Stationary Agent
JCA	Java Cryptography Architecture
ASE	Active Storage Element

ABSTRACT

Mobile agent technology is a new paradigm of distributed computing, that models well some of services available in the net than other distributed computing approaches. Although this technology has generated considerable excitement in the research community, it has not been translated into a significant number of real-world applications due to a new dimensionality of security problems it brings along with it.

Mobile agents are programs that can autonomously migrate from nodes to nodes in a computer network and perform a computation on the behalf of the user. It has a unique ability to transport itself from one system in a network to another. They work in conjunction with a mobile agent platform which provides appropriate execution environment for the mobile agent. This platform needs to be available at each potential host willing to entertain a visiting mobile agent. In this thesis all the background information on the concept of mobile agent is given.

The security problem of mobile agent technology is of multidimensional. In this thesis the issue of hostile host towards a visiting agent is given due diligence: Malicious host problem. It is one of the most difficult problems to address. Threats emanating from malicious hosts are identified and a modified mobile agent computing model is proposed to prevent some of the threats. The proposed system design consists of components that provide support to the mobile agent while it is touring hosts in the agent space. It also protects the confidentiality and integrity of parts of the mobile agent.

Keywords: agents, mobile agents, malicious host problem, trusted nodes.

Chapter 1

INTRODUCTION

Internet which has been started merely as scientific experimental project at the department of defense (DOD) in US has now changed virtually every aspect of our life. Internet Based Electronic mails, news papers, dictionaries, e-learning centers are some to mention. In short it has changed radically the way we communicate, do business, learn and access information.

The deployment of these electronic versions of the real world scenario services, have stride on their own right: offering comfort and conveniences; do the job anywhere any time; and at last saving time compared to their real world counterparts. Nevertheless, it is unfortunate that all of these services are based on the traditional client server approach. An alternative distributed computing paradigm has emerged in late nineties, which models well some of the services available on the net.

The distributed paradigm is modeled much like what top business men, famous coaches and sporting personalities used to. As a single person could not manage or handle many tasks at a time, he would delegate part of his authority to his agent, who could autonomously make a decision on his behalf. Think of a program or a software component, instead of the real person, which you delegate a task to; indeed this is what mobile agents are meant to be.

Mobile agent computing is quite a novel and powerful paradigm for building distributed applications that it has attracted attentions of many researchers around the world in the mid and late nineties. Many researches have been carried out during the same period that has resulted in a various mobile agent computing platforms. The release of Java with features that facilitates the development and deployment of mobile agent systems in the same era has played a greater role in the mobile agent computing field. Java has a tremendous support that by the end of late nineties

around 80% of the mobile agent platform available on the market were based on and used Java as their programming language [1].

However, as fascinating as the idea of mobile agent computing is, it is not without a glitch when it comes to the implementations and real world deployment. Mobile agents could be subjected to attacks launched by hostile hosts while visiting different hosts in the net. The thesis work deals with familiarization of mobile agents' technology along with their protection in hostile environments, in relation to data collecting agents. Based on the study, a modified mobile agent computing model is proposed. The proposal consists of components that help the mobile agent system as a whole to be defiant of some of the possible hostile host attacks. The detail of the proposal is presented in the thesis report.

1.1. Background

A mobile agent can be thought of as a program, which can autonomously migrate between various nodes of a network and perform computation on the behalf of a user [2]. It has a unique ability to transport itself from one system in a network to another. The ability to travel allows a mobile agent to move to a system that contains the object with which the agent wants to interact and then take advantage of being in the same host or network as the object. Hence it reduces the network communication and latency. Mobile agents are promising paradigms for the design and implementation of distributed applications [2].

Mobile agent technology is not entirely based on mobile agents only, there is another complementary component called mobile agent platform. It provides appropriate execution environment and services to mobile agents while they are on the move. Accordingly, the platform needs to be available at each potential node in the network which is going to be visited by the mobile agent. Thorough background information, generally about software agents and in particular about mobile agent systems is presented in the next chapter.

1.2. Statement of the Problem

Mobile agent systems need an open environment to operate and do their job. This characteristic has subjected them to various kinds of attacks. More specifically mobile agents and mobile agent platforms are subjected to attacks from malicious hosts and malicious mobile agents respectively. Hence, security problems in the mobile agent paradigm may be classified into two broad categories:

- Mobile agent platform protection
- Mobile agent protection

The second category, mobile agent protection, is dealt in this thesis, in relation to data collecting agents.

To prevent a malicious host from attacking incoming mobile agent is difficult and some researchers even claim that it is impossible [3]. To works around this problem, it seemed to have to use closed system or only use trustworthy machines, which severely restricts system openness. However, some research has still been done to protect mobile agents to some extent, and detect the attacks if they are done.

1.3. Objectives

The main objectives of thesis work is to study mobile agent systems in general, analyze countermeasures suggested so far to malicious hosts threats and come up with a proposition which could provide a mechanism to avert some of the hostile threats.

The second objective of thesis work is to develop a prototype that implements the proposed concept and analyze its impact on the performance of the overall mobile agent system using a number of parameters.

1.4. Scope

The thesis work targets mobile agents that are sent to agent space for information retrieval. The term agent space refers to all nodes in the internet that are capable of entertaining a visiting mobile agent.

1.5. Methodology

Different methodologies are used for the various phases of the thesis work. The first phase involves the study of the issues and areas closely related to the thesis work in order to get a deeper understanding of the problem. This is accomplished mainly through reading journal papers, articles, book and other reading materials. Major activities performed in this phase are:

- Studying about software agents in general, their characteristics and about mobile agent systems.
- Studying the various types of distributed computing paradigms which include client-server, remote evaluation, code on demand and mobile computing.

Overall the major tasks performed during this phase focused on gaining all of the necessary background information that are needed to understand the broader picture of the problem as well as its surrounding.

The second phase involves the study of the problem. During this phase threats posed on mobile computation as a whole and specifically to that of mobile agents and various propositions available also far are studied and analyzed, which helped us to come up with our own perception of the problem.

The last phase of the work deals with the development and realization of a concept that will address some of the threats posed to mobile agents. The major activities carried out include:

- Development of a concept to mitigate some of the problems posed on mobile agents, specifically to that of data collecting agents
- Evaluation and selection of different mobile agent platforms that will be used to realize the concept
- Further refining the concept developed
- Realization of the concept using the mobile agent platform selected
- Collection of data

1.6. Organization of the Thesis

The thesis is organized as follows. Chapter two presents all the necessary background information needed to become familiarized with the concept of mobile agent systems, which comprises of mobile agents and mobile agent platforms and other issues related to mobile agent computing. Chapter three presents first the various kinds of threats posed by malicious host on mobile agents and at the later section, a thorough analysis, some of the already proposed countermeasure is presented. Chapter four discusses the details of the proposed countermeasure. The last two chapters discuss about the development of a prototype that implements the concepts, gathering of results from the prototype developed and finally a conclusion of the thesis work and suggestions for future works are outlined.

Chapter 2

AGENTS AND MOBILE AGENTS CONCEPTS

2.1. Background

A Software agent, from here onward will be called simply agent, is a program that assists people and acts on their behalf. Agents function by allowing people to delegate work to them. This is very light end user perspective of an agent. A more advanced system perspective definition to an agent is: it is a software object that is situated within execution environment and posses a number of mandatory properties [4].

An agent can be seen as just another kind of software abstraction much like methods, functions and objects that are forms of software abstraction [5]. Objects are defined abstractly in terms of classes, methods and attributes. We can think of an agent as an abstraction that embodies behavior and active properties. An agent based definition for a system can thus focus on specifying agent behavior. Being active entities, in contrast to software objects of object oriented programming, agents work according to the so called Hollywood principle: “Don’t call us, we will call you” [4].

Numerous agent systems are made available to the public by research labs, commercial organizations and others .But all of these systems share common properties: the mandatory properties that all agents need to posses. Agents are Reactive (sense changes in their environment and act accordingly to those changes), Autonomous (has control over its own actions), Pro-active (goal driven); and temporarily continuous. Beside these mandatory properties, an agent could possess orthogonal properties, which would lay the foundation for further classification of the agent system. Some of the orthogonal properties include: Communicative (able to communicate with other agents), Mobile (can travel from one host to another) and Learning (adapts in accordance with previous experience).Based on the orthogonal property of *Learning,* we could

have Intelligent agent, and on *Mobility,* we could have mobile and stationary agents which this thesis is all about.

As already defined in the introductory section of the previous chapter, a mobile agent is a program which represents a user in a distributed computer network and is capable of migrating autonomously from node to node to perform some computation on the behalf of the user. Its task ranges from online shopping to real-time device control as determined by the agent's application. Mobile agent (MA) based applications inject mobile agents into a network allowing them to roam inside the network either on a predetermined path or on that the agent themselves determined based on dynamically gathered information. Having accomplished their goals, the agents return to their home in order to report the results to the user. At the heart of the Mobile agent technology, there are two fundamental components: Mobile agent platform and mobile agents themselves.

The Mobile Agent Platform provides a secure and appropriate distributed execution environment for the mobile agents. It gives the MA the following basic services: creation, execution, transportation and termination. The platform must be run at each network site willing to accept visiting agents. In other words we can think of the platform as a distributed execution environment for the MA. Mobile agents are executed as a process or a thread in the context of agent run time environment, agent server. The conceptual model of the mobile agent system is shown in figure 2.1 [1].

Mobile Agent Systems differ in many aspects and can be roughly divided based on the programming language by which the mobile agent system (MAS) is developed and used: Java and non Java based (using languages like C/C++ and scripting languages like Tcl/Tk).Analysis of Java Based MAS is given in section 2.3.

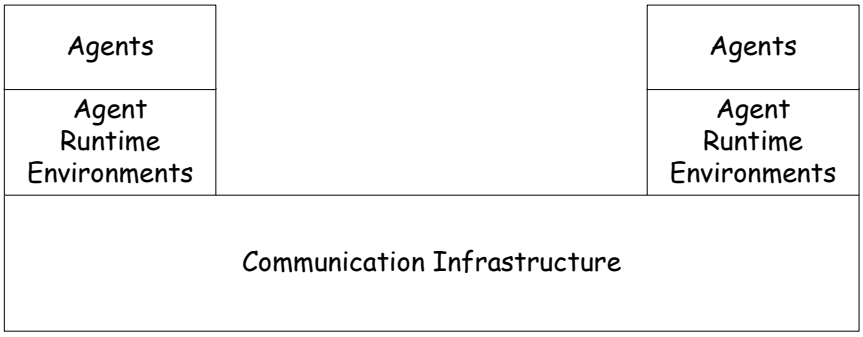

Figure 2.1 A mobile agent system

A closer look at the make up of mobile agents themselves reveals that they are indeed composed of three components: code, data and execution state. The code defines the range of tasks that the mobile agent will perform while visiting different nodes in the network. To put it in other words; it is simply a list of functions or methods defined inside the MA. The second component is the data. It is place where the mobile agent stores the information that it has collected while visiting nodes. The last component is the execution state. Based on this component, the MA could have strong or weak mobility. A strong mobility means that the attributes of the execution states are transferred via the network. So the mobile agent may not even be aware that it has been moved from one place to another. Hence, the agent continues execution at exactly the same position as it was before being transferred to the new host. Examples of MAS that implements strong mobility includes: AgentTCL and WASP. On the other hand weak mobility means only transporting the data, so the mobile agent is aware of being moved and will start executing from the beginning. MAS that implement weak mobility include. Grasshopper, Aglets and Jumping Beans.

Mobile agents provide many advantages. First, they reduce network overload, by dispatching the code to the data location, instead of bringing remote data to the code's emplacement. They also reduce network latency when used in real time device control; as they can be executed locally where the control is required. The MAs are ideal to work within environments with expensive and fragile links using their asynchronous and autonomous execution behavior. Heterogeneity is another feature inherent in mobile agents.

2.2. Network Computing Paradigms

Mobile agent computing is just only one of the four network computing paradigms used to develop distributed applications. Each one paradigm has its own mechanism of performing the computation as well as area of applicability. It should not be felt strange to face with a single application utilizing all the paradigms at the same time, in a hybrid mode. Some of the paradigms are well established, like client server, while others are brand new and not widely adopted, like mobile agent. These paradigms are:

- Client-Server,
- Remote Evaluation,
- Code on Demand and
- Mobile Agent.

2.2.1. Client – Server

This paradigm is the first one for implementing distributed applications, using remote procedure call approach. It utilizes centralized servers and distributed clients. The servers publish methods to be used by clients to access data and services it offers. Whenever the client wants to get some kind of service from the server, it calls the right published interface with appropriate arguments. The server after processing the call returns the result back to the client. Figure 2.2 shows a typical client-server environment, e.g. HTTP and FTP.

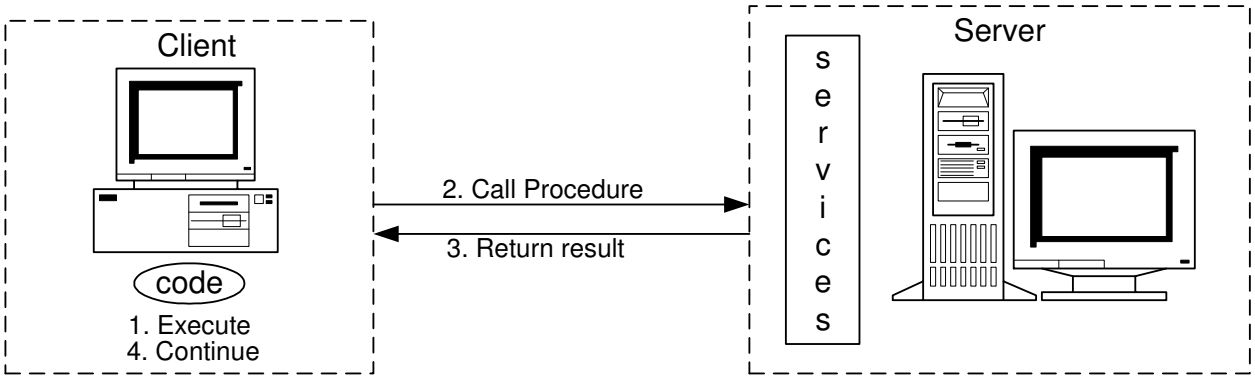

Figure 2.2 Client-server computing model

2.2.2. Remote Evaluation

Unlike the client-server paradigm, this paradigm not only transfer data between the client and the server, but also code. Once this code is transferred to the server it will be executed there and only the result will be returned to the client. In contrast to client – server paradigm, only one connection to the server has to be made to transfer the code to the server instead of several, as in client – server [4]. Figure 2.3 shows typical remote evaluation environment, e.g. rsh command in Unix.

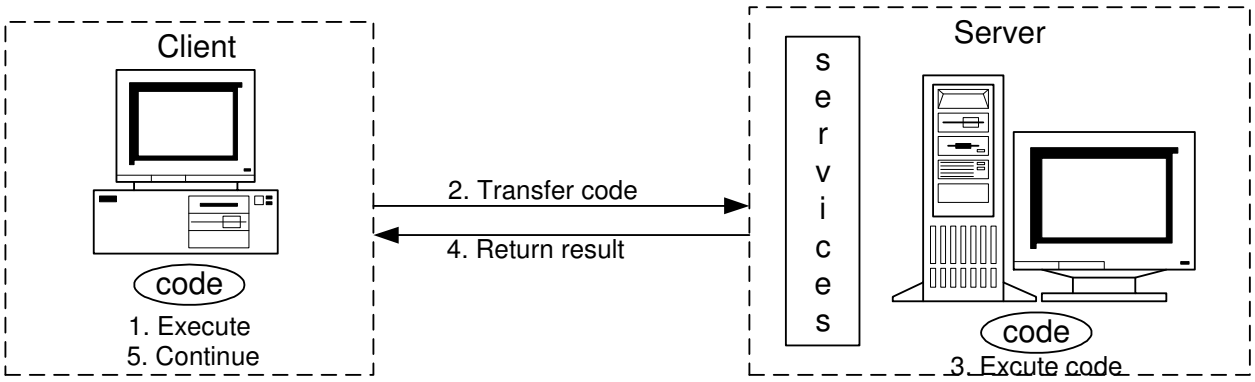

Figure 2.3 Remote evaluation

2.2.3. Code on Demand

In this paradigm the code is stored on the server, which will be downloaded by the client to perform the task. Java Applets are a famous implementation of this paradigm. Servlets are also good example of code on demand computing, but here in contrast to applets the code is uploaded to the server from the client side. A simple code on demand environment is shown in figure 2.4.

One advantage of this paradigm is that, as the execution takes place locally and use the server merely as a repository of the code, it has the capability of decentralized computing. As a result, servers do not have to be as powerful as in the case of client-server paradigm, where all the processing takes place at the server side.

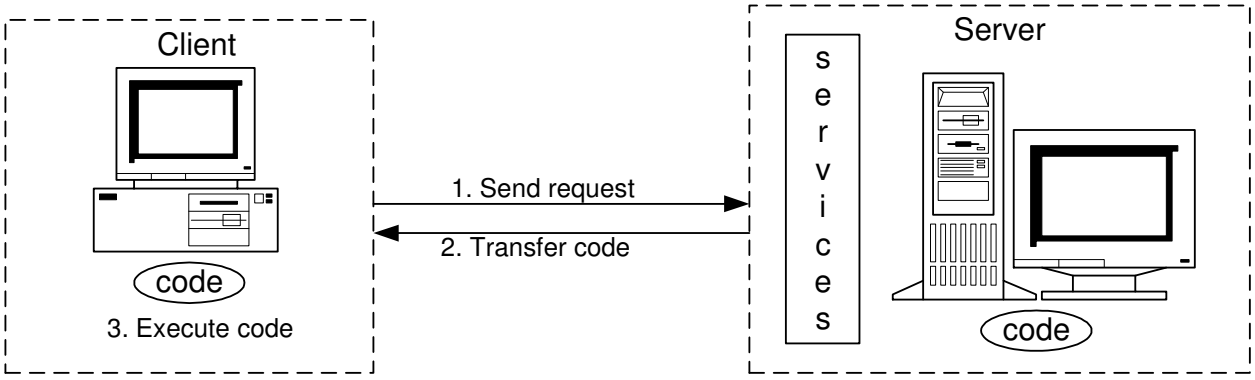

Figure 2.4 Code on demand computing model

2.2.4. Mobile Agent

In this paradigm, the code (agent) moves from place to place to perform computation on the behalf of its owner. It is some what similar to remote evaluation except that the code has autonomy and could visit multiple nodes before returning to its home. Figure 2.5 shows typical mobile agent computing environment.

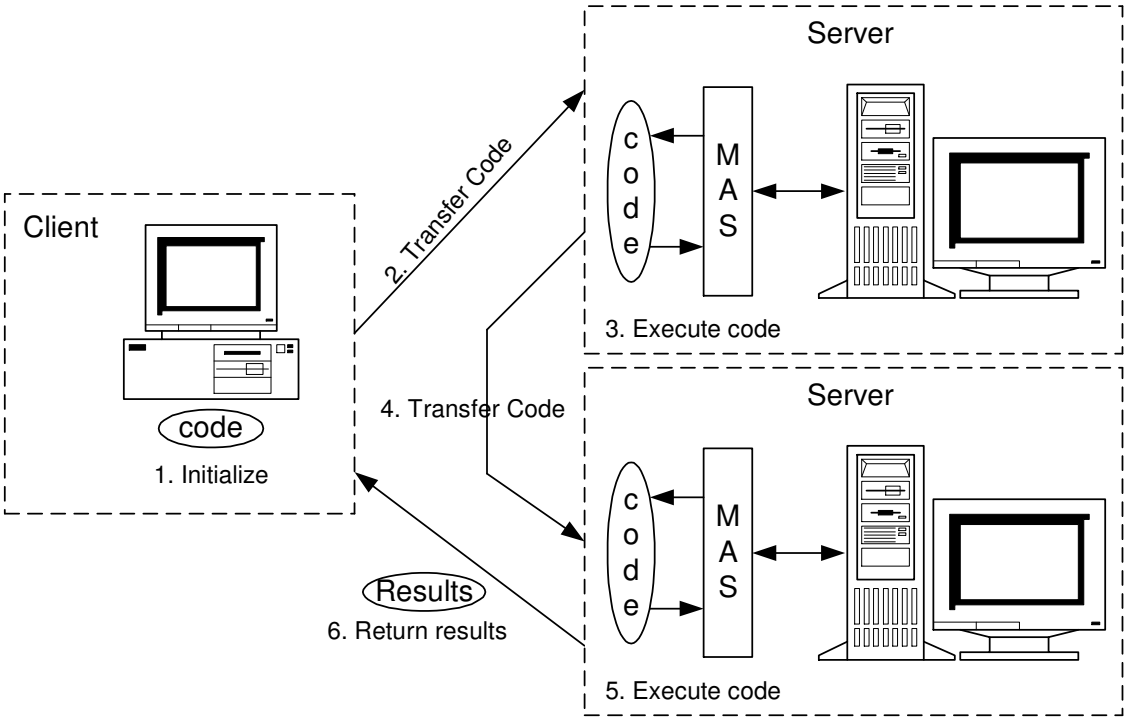

Figure 2.5 Mobile agent computing model

2.3. Java Support for Mobile Agent Computing Paradigm

Java is an object oriented network programming language that is built using C++ as a blue print while still omitting rarely used and confusing features of C++. Some call Java as a language of the internet because of its support for network programming very easily without placing too much burden on the programmer's side.

As Arnold and Gosling put it "It would be an understatement to say that Java programming language has revolutionarized the mobile agent field" [6].In fact, so much so that around 80% of mobile agent systems available today are built using Java. Some of the features of Java, that makes it ideal for mobile agent programming, are:

- Platform independence or Portability:

Java, from its inception, is supposed to operate in a heterogeneous network environment, using architectural neutral byte code as opposed to non portable native code. For the byte code to be executed at a given computer all that is needed is the Java runtime environment or Java Virtual Machine (JVM). JVM masks the underlying complexities of the computers. This platform independence feature of Java allows us to dispatch mobile agents without worrying much about the make up of the computer they are going to be visit.

- Security:

Java has a built in security mechanism through security managers and security policies.

- Multithreaded programming Support:

Multithreaded programming enables different sections, threads, of a given code to be executed independently or autonomously of each other. Threaded programming support of Java helps implement the autonomous behavior of mobile agent very friendly, as different mobile agents with in the same agent runtime environment are executed as threads.

- Support for Migration and Communication:

Java provides support at the programming level for mobile agent migration and communication through Serialization and Deserialization: A mechanism to store the state of an object (Serialize) in a from sufficiently detailed for the object to be reconstructed later on (Deserialization); Dynamic class loading: Capability to load and define classes at runtime; Reflection: Capability to discover information about fields, methods and constructors of loaded classes; and Remote method invocation.

The Conceptual model of a Java-Based mobile agent platform is depicted in figure 2.6. The agent platform provides an agent Application Program Interface (API) with basic functionality for agent management, migration and communication, in the form of Java packages. TCP/IP is used as the main transport mechanism. The agent server is a multithreaded and runs on top of the JVM [1, 6].

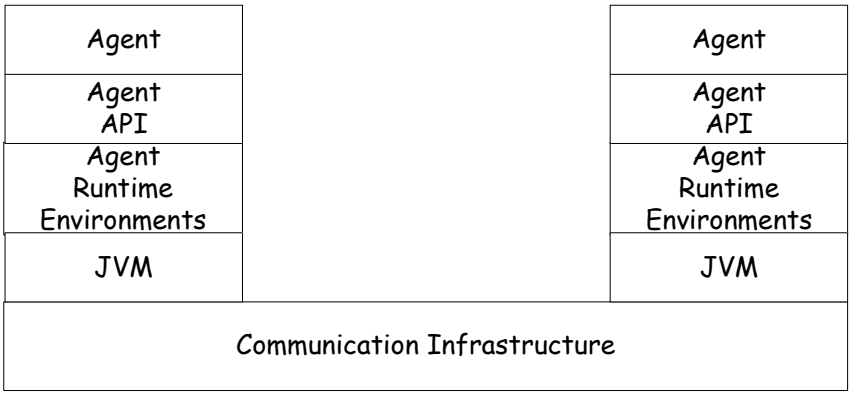

Figure 2.6 Detailed view of mobile agent platform

Two of the most popular Java-based platforms are IBM's Aglets and IKV^{++}'s Grasshopper. Aglets is one of the first Java-based mobile agent platforms.

Chapter 3

THREATS AND COUNTERMEASURES

3.1. Introduction

We have seen in the previous chapter a detailed view of the makeup of mobile agent systems. Such a view helps us to grasp the idea behind the concept of mobile agents, programs that migrate from one host to another, on the user's behalf. However to discuss issues related to security, it is sufficient enough to consider the two fundamental components of MAS: the agent and the agent platform.

If we play around between these two basic components of the MAS, considering one as a threat to another at a time, it would become clear that the security issues of MAS are of multidimensional. An agent could attack a platform, an agent could attack another agent when they meet somewhere in the agent space and lastly but not least, a platform could launch an attack against its visiting agents. The first two attacks have their counter part in the traditional client server environment and solutions available to the client server environment could be applied with very little modification. But the last kind of attack, a platform launches an attack against its visiting agent, is the most difficult one of all attacks to solve, some researchers even claim that it is impossible to solve [3]. This thesis is all about looking into this attack (Malicious Host Problem).

In this chapter, first we will take a closer look at the various kinds of attacks a malicious host could launch to a visiting agent and later on the various countermeasures proposed so far to solve the problem will be presented.

3.2. Hostile Host Threats

These types of threats represent a class of threat, where the host compromises the agent. The hostile actions include: Masquerading, Denial of Service, Eavesdropping and Alteration of carried result. These attacks are most difficult to be detected and prevented, since the host has a full control of the agent's code and data.

3.2.1. Masquerading

An agent platform can masquerade as another agent platform in an attempt to deceive the mobile agent as to its true destination. The problem has more to do with the capability of a visiting agent to correctly identify and authenticate its executing host, while it is actually on it [7, 8].

3.2.2. Denial of Service

When an agent arrives at an agent platform, it expects the platform to execute its request faithfully and provide a fair allocation of resources. A malicious platform, however, may ignore agent's service requests, introduce unacceptable delays for critical task or even terminate the agent without notification. Agents, if any, which are waiting for a result from a non-responsive agent on malicious platform must be careful to avoid becoming deadlocked [7].

3.2.3. Eavesdropping

The classical eavesdropping threat involves the interception and monitoring of secret communications. The threat of eavesdropping, however, is further exacerbated in mobile agent systems because the agent platform can not only monitor communications, but also can monitor every instruction executed by the agent, all the data it brings to the platform, and all the subsequent data generated on the platform. Since the platform has access to the agent's code, state, and data, the visiting agent must be wary of the fact that it may be exposing proprietary algorithms, trade secrets, negotiation strategies, or other sensitive information. Even though the agent may not be directly exposing secret information, the platform may be able to infer meaning

from the types of services requested and from the identity of the agents with which it communicates [7, 9].

3.2.4. Alteration

Alteration threatens the integrity of the agent as a whole. As already discussed earlier when an agent arrives at a given host, it exposes its code, state and data to the platform. A mechanism that ensures the integrity of the agent, defiant of modification threat needs to be in place, as the agent visits several platforms throughout its life time [9].

3.3. Countermeasures for Malicious Host Threats

Over years, a number of countermeasures for malicious behavior of hosts towards a visiting agent have been proposed, some of them are applicable, while others have only a theoretical significance. It is very nature of a way a visiting agent being executed at a host having a complete control over its execution environment that makes this problem hard to solve. Once agents enter the execution environment they will not have a capability to prevent any malicious behavior from occurring.

Generally, the countermeasures provide either detection or prevention mechanism to the visiting agent. Prevention mechanism aims to protect the mobile agent to such an extent that it becomes difficult or at least very expensive to attack the agent, while detection mechanism performs a regular checking to discover possible security breaches. The following subsections provide a brief introduction to some of the main countermeasures available so far.

3.3.1. Trusted Hardware

This countermeasure tries to enforce the notion of trust between an agent and a host by physically adding, secure – tamper detecting and responding hardware to conventional computing systems. The hardware encapsulates the entire environment in which the agent executes, creating a safe

heaven within hosts in the agent space. It protects the visiting agent from any possible attack that could be launched by the entertaining host.

The major drawback of this proposition is that it comes up with another prerequisite for an agent to execute at a given host. After all, that is all needed for an agent to execute at a given host is the availability of Mobile Agent Platform. But now the agent also needs the existence of the "safe heaven" (literally speaking tamper resistant and detecting hardware) at each host it is going to visit. This has the effect of constraining the itinerary the agent could follow, dramatically limiting the systems' openness [10].

3.3.2. Trusted Execution Environment

This method is a variation of the above method. It eliminates the deployment of the specialized hardware. Instead, according to this method a set of trusted nodes needs to be setup in the agent space prior to any agent to host interactions [2]. Yet again this proposition, like that one discussed above, goes against the notion of an agent freely roaming inside the agent space to accomplish its task. At the end of the day, if a mobile agent has an impeded itinerary, the vast amount of resources available on the internet, as well as good competition between hosts will be lost or severely constrained [2].

3.3.3. Computing with Encrypted Functions

Computing with Encrypted Functions prohibits the executing host from learning anything substantial about the agent. It has been suggested by Sander and Tshudlin as referenced in [8], and tries to ensure the computation privacy of the agent in the untrusted host. The problem of computation privacy arises because of the inability of the agent to prevent disclosure of the program it wants to have executed. If we have a method by which we can encrypt a program, and let a platform executes it without the need for decryption, this problem is essentially solved. In particular, we have code privacy and code integrity in the sense that specific tampering is not possible. The problem is stated by Sander and Tschudin as follows: [8]

"Alice has an algorithm to compute a function f. Bob has an input x and is willing to compute f(x) for her, but Alice wants Bob to learn nothing substantial about f. Moreover, Bob should not need to interact with Alice during the computation of f(x)."

According to the proposition, functions will be encrypted such that their transformation can again be implemented as programs. The resulting program will consist of clear text instructions that a processor understands. But the processor will not understand the "program's function".

The idea is elegant but the challenge is to find appropriate encryption schemes that can transform arbitrary functions. A lot of work needs to be done in this field to explore various possibilities [8].

3.3.4. Code Obfuscation

Code obfuscation is suggested by Hohl, before encrypted functions are even described, as referenced in [8]. According to this proposition an algorithm called obfuscating algorithm will be used to mess up the code, while still creating a semantically equivalent version of a program. The idea is to limit the understanding of the code to only the observability of input and output, making the program to behave like a black box.

Hohl proposed obfuscation as a combination of two mechanisms:

- The first as described above dynamically generates a new and less understandable version of the mobile agent's code.
- The second restricts the life time of the MA code and data, where the agents code and data cannot be modified or read for certain interval and so any hostile actions during that interval do not have any effect – (Time limited black box)

The main drawback of this proposition, there isn't yet a known algorithm for messing up the code and then still capable of performing the intended task [10]. Although a number of approaches could be followed to make a code less understandable, they only have effect on the amount of time required to reverse engineer the code.

3.3.5. Path Histories

In this proposition, each host visited by the MA adds a signed entry to the itinerary carried by the MA. The entry consists of the identity of the current host as well as the next host the agent intends to visit. This approach is strongly used as a countermeasure to malicious agent threats, where what the agent has done before is used to check if it has any malicious behavior. When it comes to hostile host threats, the logs of the agent itinerary entered by the visited nodes, is used to check whether the path carried by the agent has been altered by a malicious host somewhere in its itinerary [8].

The drawback of this proposition, the checks are carried out when the agent returns back to its home. If a malicious host somewhere in its itinerary has compromised the agent, all the tasks that it has carried out are nullified. Turning the data collected no more than garbage and the agent no more than a mobile garbage bin.

3.3.6. Mutual Itinerary Recording

According to this method, agent's itinerary will be recorded and tracked by another cooperating agent in a mutually supportive arrangement. When each agent visits a host, it conveys to its peer information about the last platform, current platform and next platform it has visited and going to visit. The peer maintains a record of the itinerary and takes appropriate action when an inconsistency is noted. The peers have to be wary of the hosts visited by each so that neither of them revisits a host.

The idea behind this method is founded on the assumption that not all, rather few platforms are malicious. Accordingly, even if an agent encounters one, this platform is unlikely to collaborate with another malicious platform being visited by the peer. "Hosts will not cooperate to launch an attack to both cooperating agents at the same time" is the principle behind this proposition.

The cost of setting up an authenticated channel and the number of agents collaborating are some of the drawbacks of the method [7].

3.3.7. Itinerary Recording with Replication and Voting

Schneider proposes this countermeasure, which uses multiple copies of MA to perform a computation, than a single copy. It is used to achieve fault tolerance and suitable in environments where agents can be duplicated without problem. Under this proposition even if malicious hosts succeed in launching an attack to a number of agents, enough replicates survive to tell the story. Agents that has successful accomplished their task participate in the voting process to determine the correct response. Only a result that has a backing of the most will be considered as a legitimate result-hence correct [10].

Chapter 4

PROPOSED COUNTERMEASURE

4.1. Introduction

In this chapter we take a detailed look of the proposed countermeasure : How computing is done under this proposition ; its basic components and security protocol developed , so as a whole the mobile agent is capable of resisting some of the attacks that could be launched by hostile hosts.

The proposed countermeasure is based on trust. Generally a trust that a mobile agent has on a particular host can be blind folded, based on policy enforcement or based on control and punishment.

A blind folded, kind of trust, is the one in which the mobile agent "simply needs to trust its entertaining host". In this scenario, there is too much reliance on the host from the mobile agent. The host can do whatever it wants while giving services to the mobile agent, but still it is trusted, it neither has malicious behavior nor collaborate with other hostile hosts that perform some evil action on the agent. This kind of trust , between the host and the mobile agent ,does not have of much applicability especially when the mobile agent is sent to visit nodes under a variety of security domains, where the owner of the mobile agent does not have the authority at all to set the type of trust the host being visited should provide .

The second kind of trust is based on policy enforcement .In this case the mobile agent and the host have a prior contractual relationship in the form of policy. As defined by Wilhelm and Staamann: Policy is a set of rules that constrain the behavior of a host for all conceivable situations and trust in a host as a belief by the mobile agent that the host will adhere to its published security policies ,as referenced in [5]. Such kind of trust should work fine as long as

the signing parties conform to their rights and obligations. The down side, it requires a prior agreement between the host and the agent.

The last kind of trust is based on control and punishment. Here no prior policy needs to be signed between the two parties. Although there is no contract signed it is neither a blind folded trust as in the first case. The trust here puts certain degree, although not complete, of belief on the host. It assumes that hosts are not by nature malicious and give them a chance to behave accordingly. But it still uses control mechanism to punish the host if found guilty of misbehaviors.

The proposed countermeasure uses a combination of the above two kinds of trusts, based on to which nodes, in the computer network, the mobile agent is interacting or talking to: based on policy enforcement and based on control and punishment. As seen on the previous chapter a number of trust based countermeasures has already been tried – like trusted hardware, trusted execution environment and trusted third parties**.** This proposal can be thought of taking these concepts a step further to try to address the problem. Before we can move on to describe the proposed countermeasure, the next section outlines the guidelines used to develop the countermeasure.

4.2. Design Guidelines

Although mobile agents can be used to perform a number of tasks, the proposition is specifically tailored to agents that are destined to gather information from nodes, these agents are called data collecting agents.

The following points are used as guideline when the proposition is being developed:

- Convenience to the owner of the agent: Once the owner of the agent sent the MA, he does not have to worry anymore about whereabouts of the agent. All that he needs is the agent to return back with its result nothing more.
- Abstraction of the modification: The owner of the agent does not have to feel the difference between the proposed and the normal mobile agent operation and computation.

- No pre negotiation with hosts: The mobile agent should be able to freely roam inside the agent space without a prior agreement between the set of hosts to be visited and the user of the mobile agent.
- Ease of access of information gathered: Once the mobile agent returns to its home the user should be able to retrieve the information gathered from the data carried back by the mobile agent, without the need to contact the hosts involved.

4.3. The Proposed Countermeasure

A closer look and evaluation of the various kinds of attacks launched by hostile hosts reveals that, mobile agents are subjected to such kind of attacks because they are a lonely figure once sent to the agent space. Hence the proposition builds on the concept of introducing a supportive element in order to address hostile host threats.

The computational model of the original mobile agent system consists of home of the mobile agent, computers in the agent space (hosts) and the mobile agent itself. In this model, the mobile agent visits nodes (hosts) that are in the agent space dictated by its list of destinations, gets the data, carries it around as it visits other nodes and will eventually return to its sending node with the result. This is an overall view of the computing model of the original setup. Refer to figure 4.1 below.

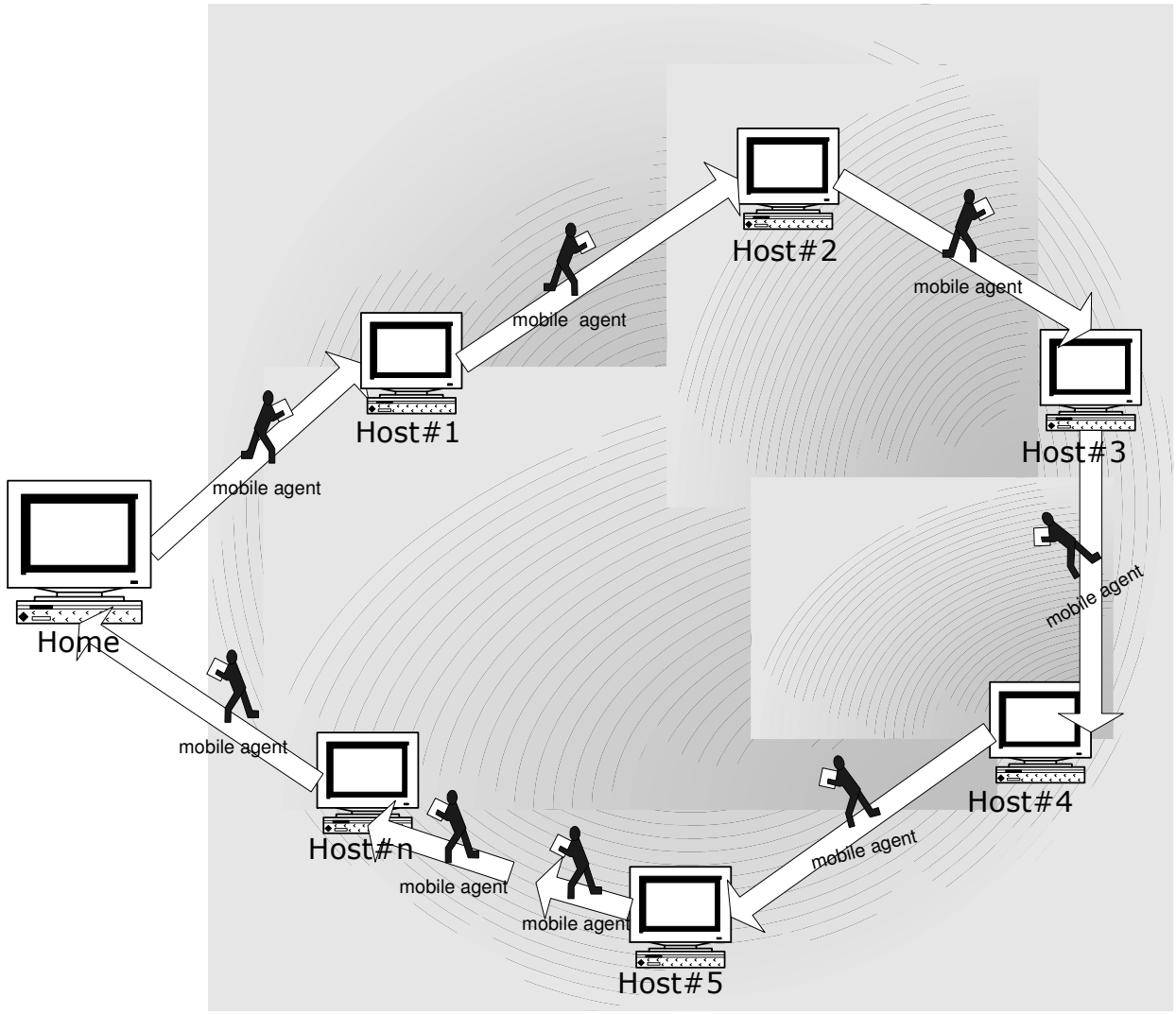

Figure 4.1 Original mobile agent computing model

Figure 4.2 shows an over view of the mobile agent computing model, in which the proposed countermeasure is taken into account, with a number of additional elements. Much like the case of trusted third parties a node is setup in the agent space to provide a different task to the mobile agent. In this setup it is mandatory that the home or owner of the mobile agent has a public-private key pair at its disposal, the public key is published to the world, so that the mobile agent could retrieve this key while it is visiting hosts. As will be discussed later on, these keys are used

by the security protocol to protect the confidentiality and the integrity of parts of the mobile agent. Let us get a glimpse of some of the introduced features.

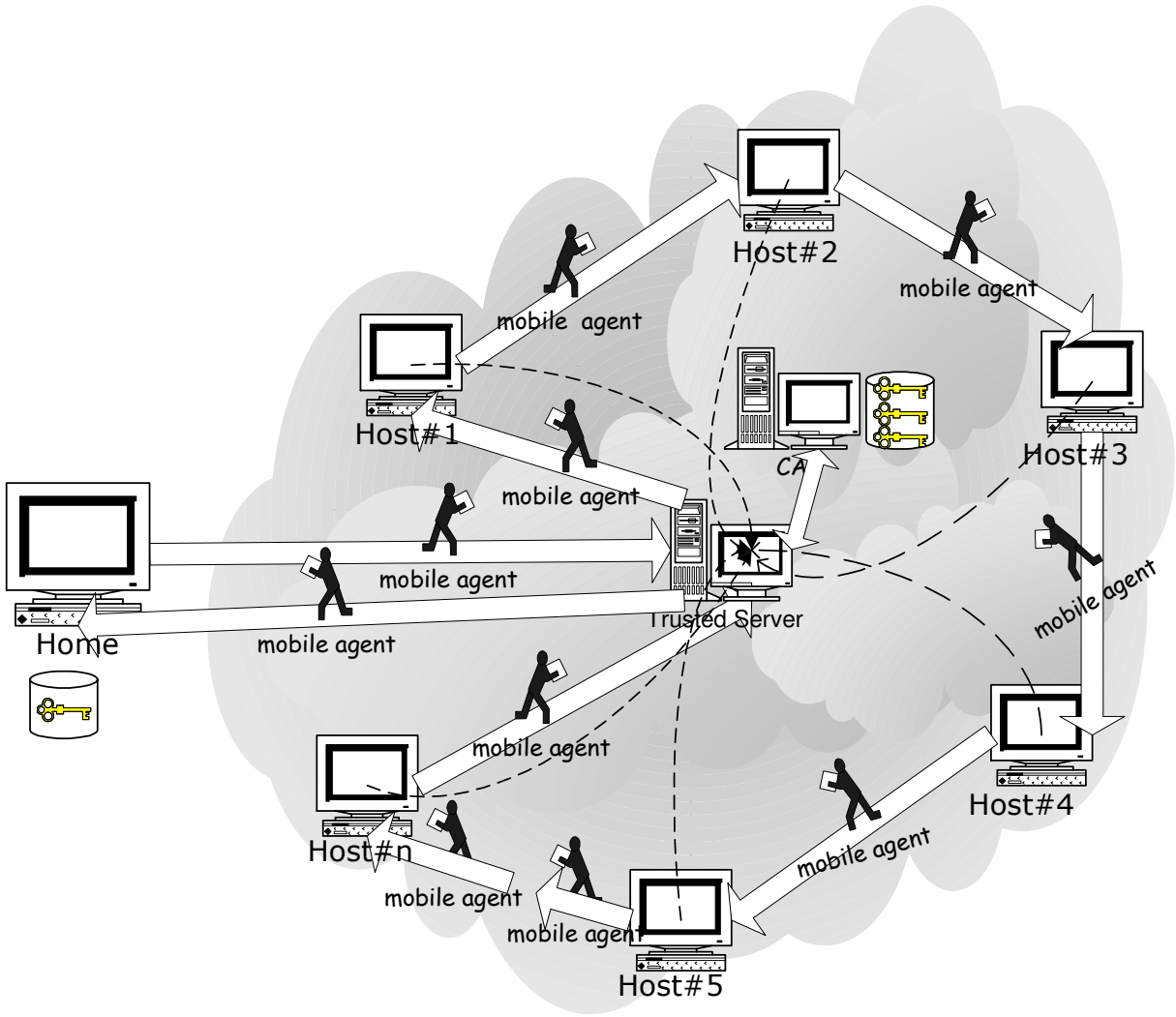

Figure 4.2 Proposed mobile agent computing model.

As it can be seen from figure 4.2, the proposal modifies the way by which the mobile agent computation is done. More specifically the arrows dictate that, the mobile agent first goes to the trusted node, creates a temporary storage element called active storage element (ASE), then moves to the first host to be visited. It goes there, sends the information it has retrieved from the corresponding host to be stored temporarily at ASE. The trusted node accepts the information and

stores it. Each mobile agent that has a trust relationship with this node does the same, creates its own ASE at the trusted node and uses it to store the partial information it retrieves from each hosts. At the end of its mission the mobile agent returns back to the trusted node and asks the corresponding ASE to hand it over the results it has been accumulating so far, carry back the result to its home as if it has been doing the job alone.

Unlike the original model of a unified agent space, it is assumed that the agent space is divided into regions, within each region a node called trusted server is setup. These servers provide various services to the mobile agent while the agent is in the agent space. The mobile agent supported by these third parties trusted nodes as well as the security protocol discussed later on should be able to avert some of the evil acts from hostile hosts. The division of the agent space into regions is analogous to the cells in the mobile communication systems. In case of mobile communication systems, at the center of each cell there is a transmitter and receiver, likewise in the proposed countermeasure at the center of each region there is a trusted server.

The concept of introducing a trusted server setup in a network handled by a third party is not new at all. If we look around a number of servers deployed in an internetwork, they all or at least at some point in their operation provide synonym. Web servers, mail servers and root Domain Name Services (DNS) servers are some to mention. Let us take on the web servers as an example to highlight the similarity as well as the feasibility of the proposition.

These days we can develop our own web page and upload it to a web server for free. Let us take this argument a step further as it might seem less convincing in case of freely web hosting services. Take Ethiopian Telecommunication Corporation (ETC), it provides web hosting service with the amount of fee to be paid depending on the size of file we want to upload as well as other features our page requests from their web server. In either case all that is needed from our side is to pay the price. Indeed the two parties, the ETC and the one who wants to get hosting service, have to be agreed on terms of use. The ETC once agreed on to host our page, it provides all the necessary computational resources, when our page is being viewed by all around the world. The ETC has the obligation not to modify the content of our page without your permission, hence the

ETC should display our page as it is. This has a strong implication on the feasibility of a trusted server set up in the agent space by a third party which could provide a processing service to the mobile agent without altering the data or code of the mobile agent. Much like the terms of use signed between the above two parties, here also terms of use could be signed between the user of the mobile agent (home node) and the trusted server in the agent space in a form of policy enforcement. Hence trusted servers will not modify the mobile agent's content, as web servers do not modify the web page they host. But here the security protocol provides further protection to the mobile agent content at the trusted server.

It is such a similar concept that the proposition wants to exploit. The nodes and the trusted servers could be set up, in a similar style as nodes of root Web servers, by the mobile agent user community. More specifically by the huge set of nodes that are set up to be visited by the mobile agent.

In section to follow, we will take a look at the main components of the proposal and how should the components interact according to the security protocol.

4.4. Components of the Proposed Countermeasure

The figure depicting the overall view of the proposal (figure 4.2) shows that the countermeasure constitutes various components at various degree of multiplicity. Each of these components are listed and defined as follow:

(i) Home of the mobile Agent,

(ii) Mobile Agent,

(iii) Trusted Node (TS),

(iv) Active Storage Element (ASE) and

(v) Host.

Home of the mobile agent (Home):

It is the computer running mobile agent platform and has sent the mobile agent to carry out a task on its behalf. It can also be defined as a computer running a mobile agent based distributed application. The application as a part of its mission packs a task into the mobile agent and sends it to the agent space. The mobile agent after completing its task will eventually return to the home carrying the result.

Mobile Agent:

As defined throughout this thesis, it is a program that migrates from one node to another node in a computer network to accomplish a task given to it by its owner.

Trusted Node:

It is similar in composition to the home of the mobile agent, but differs in the function it provides. It is there to provide support and service to the mobile agent while it is in the agent space.

Active Storage Element:

It is a temporary storage element that is created by each mobile agent, at the trusted server, that is sent to visit nodes in the agent space. It actively participates in the process of temporary information storage and handing over of all the information to the mobile agent.

Host:

It is a computer in the agent space running mobile agent platform and entertains any visiting mobile agent which would like to gather information from it. This component is at the center of the controversy, which could be hostile. The host provides all the necessary resources for the agent to execute there.

4.5. Security Protocol

A security protocol that defines how the basic components of the system (Home, Trusted nodes and Active storage element) should interact with each other as well as what are the needed tasks to be performed at each level, so as the whole system could stand against the possible hostile host threats is developed.

The security protocol alters what the mobile agent constitutes depending on where it is. While the mobile agent is transiting between its home and trusted nodes the usual composition is deemed. But when the agent is in the agent space visiting different nodes it has assumed to be composed of only the two out of the three components that is usually associated with: code and state, to give hostile hosts no chance of disclosure of information collected from previous hosts.

The security protocol also develops a mechanism that lets the user of the mobile agent to digitally sign the list of destinations it wants the mobile agent to visit. After forming a destination object which contains the list of all hosts the mobile agent is going to visit, it digitally signs the destination object using its private key, then the destination object is passed down to the mobile agent. The mobile agent upon its arrival at each and every host in the agent space verifies that it has a valid copy of the destination object before putting that object into use. So the mobile agent avoids the possibility that it would be directed to visit other hosts by altering the list of paths it has carried from its home, as any malicious host could not counterfeit the digitally signed destination object. Let us see, step by step, the security protocol in action and its effects on the components of the mobile agent system. It is assumed that, the home node has a public-private key pair (HPubK-HPrvK).The public key could be retrieved by the hosts from relevant authorities.

At Home, as shown in figure 4.3**:**

- The user of the MA specifies the address of the list of hosts it wants to be visited using the Agent Based Application (ABA),

- The ABA accepts the list and forms a destination object. The destination object includes the list of hosts to be visited, the address of the trusted server (TS) and the home,
- The ABA digitally signs the destination object and passes it to the MA which is programmed to perform the required task and
- The MA accepts the signed object. By using HPubK, the MA verifies that it has the right unsigned destination object from which the address of the next node to be visited is determined and dispatches itself to that node, as pointed out in the previous section it goes first to the trusted node.

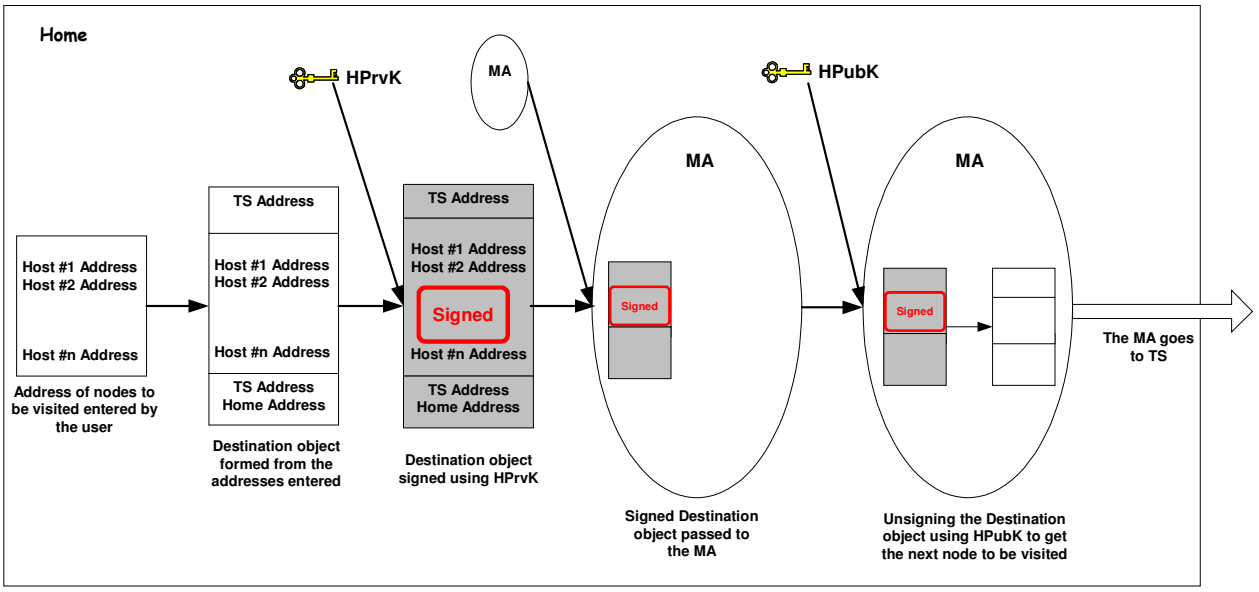

Figure 4.3 Security protocol at Home

At Trusted Server (TS), as shown in figure 4.4:

- The MA arrives at the TS, creates its own Active Storage Element (ASE),
- The MA passes down the necessary information to the ASE so it can effectively communicate with it,
- The MA retrieves the public key of the home, HPubK,
- Using this key, the MA unsigns the digitally signed destination object and determines the next node to be visited. In this case it is the first host in the list and
- The MA Dispatches itself to that node.

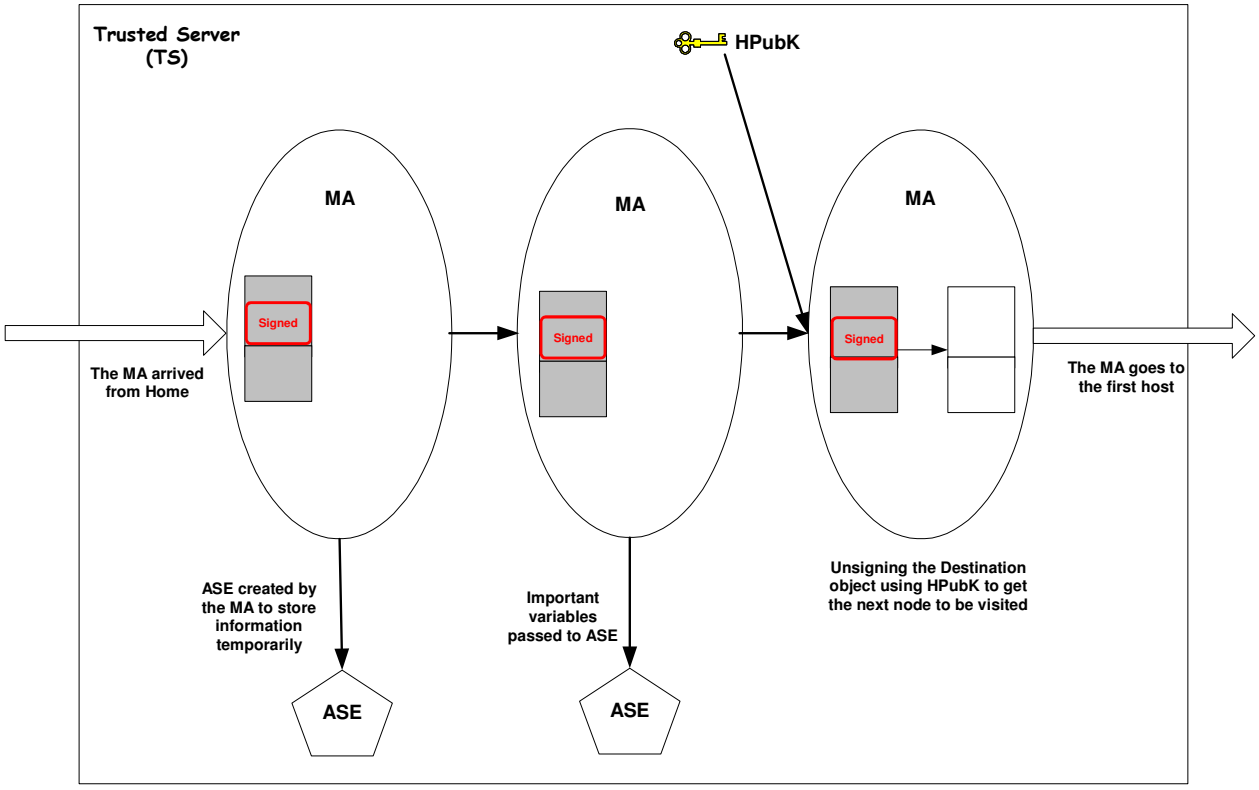

Figure 4.4 Security protocol at TS.

At i^{th} host, as shown in figure 4.5:

- The MA arrives at the i^{th} host, Host_i,
- It generates a random symmetric key, SymK_i,
- The MA retrieves the public key of the home, HPubK,
- Asks the host about the information it wants, Info_i,
- Encrypts the information using the symmetric, SymK_i(Info_i),
- Encrypts the randomly generated symmetric key using the public key, HPubK(SymK_i),
- Sends both of these information, HPubK(SymK_i) and SymK_i(Info_i), to its ASE at the TS to be stored temporarily,
- The MA unsigns its destination object, looks the address of the next node to be visited and dispatches itself to that node.

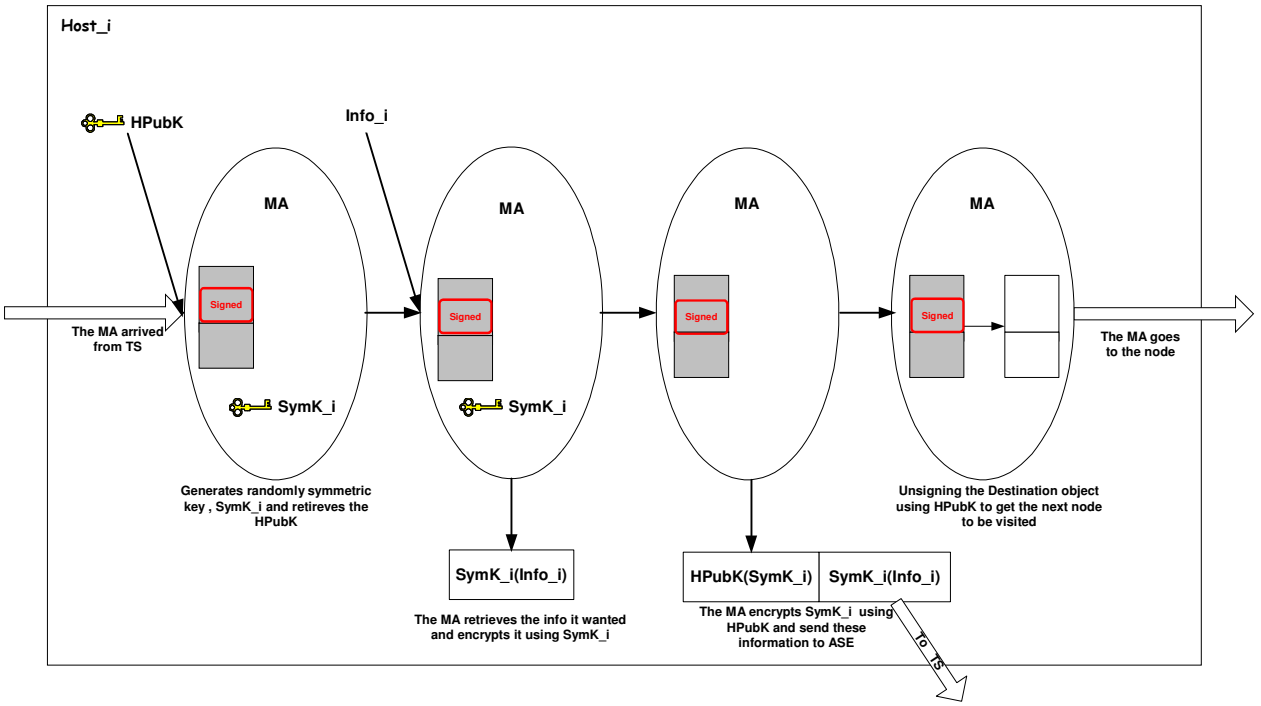

Figure 4.5 Security protocol at the ith host.

If the next node is another host it does the same task as indicated above. Else if it is a trusted server, the following set of actions follows.

At Trusted Server, as show in figure 4.6:

- The MA arrives at the TS, in its last leg of journey,
- Asks the corresponding ASE to hand it the overall information it has been accumulating so far (a pair of HPubK(SymK_i) and SymK_i(Info_i) retrieved from each host), and takes these information,
- After unsigning its destination object, it looks for the address of the next node to be visited. In this case for sure it is the home node and
- The MA dispatches itself to its home.

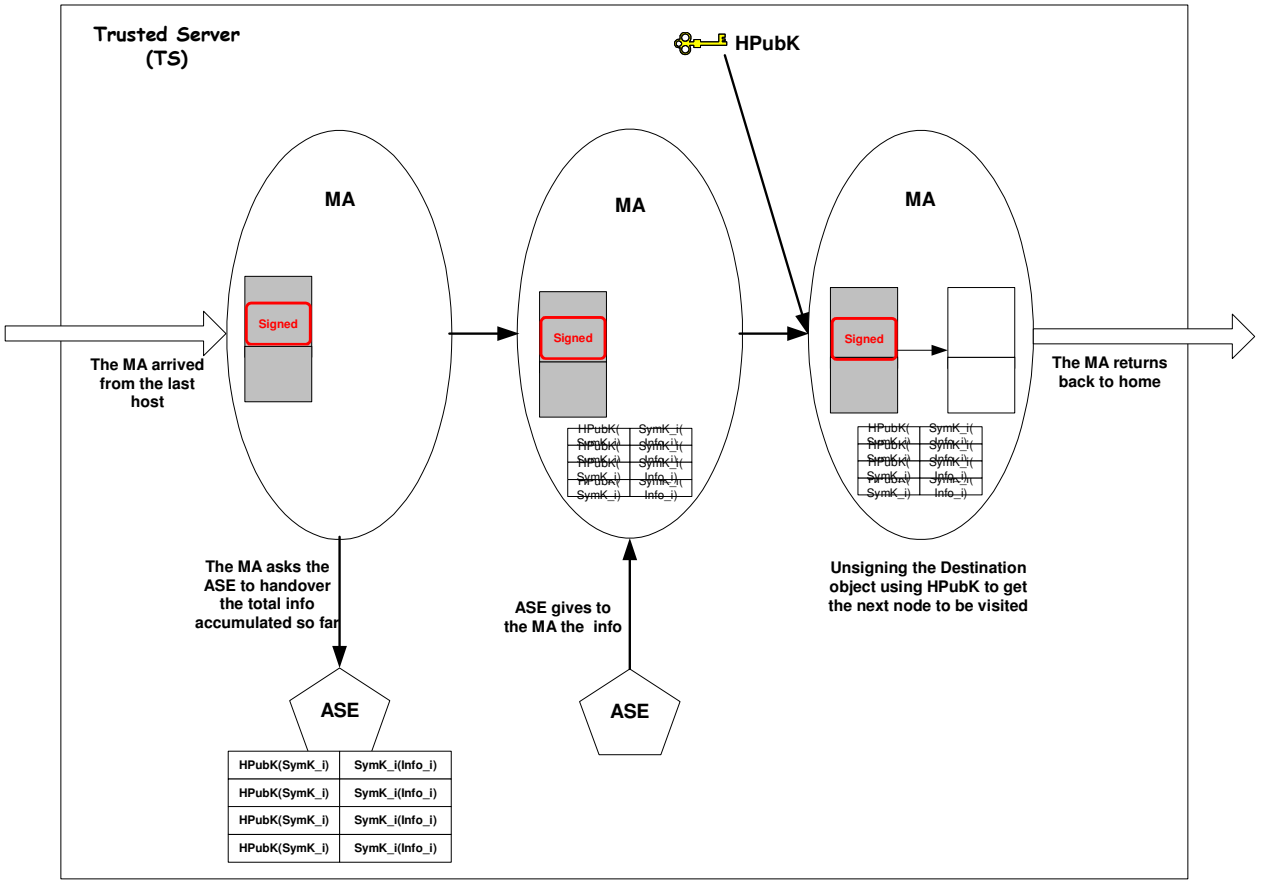

Figure 4.6 Security protocol back at TS.

At Home, as shown in figure 4.7:

- The MA arrives back at home after doing the task assigned to it,
- The MA contains a pair of encrypted information,
- The MA hands the overall information to the ABA. Note that they are all in encrypted form.
- For each pair of encrypted information retrieved from each host, the ABA does the following :
    - First, using its private key(HPrvK) to decrypt the encrypted symmetric key, HPubK (SymK_i) ,

- Second, using the decrypted symmetric key, SymK_i, it decrypts the information which is encrypted using the same key, SymK_i(Info_i),

➢ The ABA does the same process for each pair of information retrieved from every host the agent goes to collect information and

➢ At last the ABA displays the result to the user, Info_i.

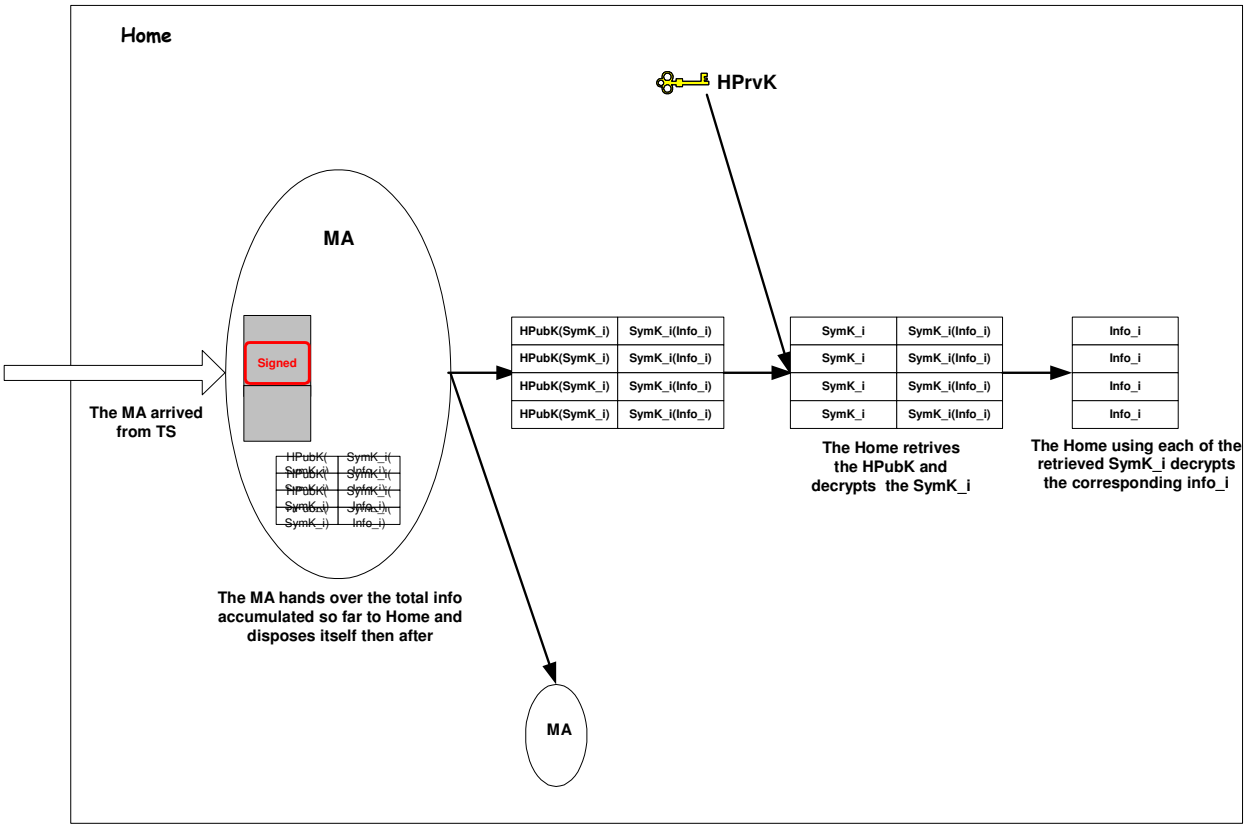

Figure 4.7 Security protocol back at Home.

Security Protocol Summary:

- For N hosts:
  - N hosts addresses digitally signed by the home node,
  - One ASE created at TS,
  - N symmetric random keys (AES) generated at each host,
  - N information retrieved will be encrypted by the corresponding N symmetric keys,

- The N symmetric keys will be encrypted by the public key (RSA) of the home,
- The encrypted N information and encrypted keys stored at the ASE,
- Decryption at home node and
- Displaying the plain text result to the user.

4.6. Proposition Evaluation

In this section we will look at, which of the malicious host threats presented in chapter 3 are addressed using the various functionality of the countermeasure presented above and how.

Alteration:

The countermeasure address alteration of carried results threats from malicious hosts. Unlike the previous mobile computation model, the mobile agent does not carry the result it has gathered from the previous hosts while visiting the next host. It gives no chance to a malicious host interested to hack into the information provided by other competing hosts.

On top of that, the security protocol through usage of randomly generated symmetric keys protects the confidentiality of the information retrieved while the information is being sent to the TS for temporary storage.

The countermeasure through separation of the destination part from the composition of the mobile agent also provides protection of alteration of the destination object. As pointed out above the user of the mobile agent creates a destination object , which consists of the list of all hosts to be visited in the agent space as well as the home and TS addresses, then the user digitally signs this object using its private key. This makes a difficult task to a malicious host which would like to misdirect the mobile agent by supplying a wrong destination object, as it cannot counterfeit the digitally signed destination objects. Any attempt on replacing the destination object while the mobile agent is way in the agent space will be detected by the mobile agent itself as it always checks the validity of the destination object it carries on its arrival at each and every host before using it to select the next host to visit.

Overall from the perspective of alteration threats the proposals addresses it using a divide and conquer approach, i.e. the threat of alteration of information collected and alteration of the list of host to be visited are prevented individually using different mechanisms.

Eavesdropping:

The countermeasure addresses the threat of eavesdropping of the collected information, using the security protocol to randomly generate a symmetric key that will be used to encrypt the collected information. The sending of information in encrypted form prevents external malicious eavesdropper from learning anything substantial while the information is sent for temporary storage. The computing model of the proposal also prevents internal eavesdropping to be an impossible task. A malicious host which would like to look into the information collected so far simply cannot do so, due to the adapted new computing model, as the information is not available there.

Chapter 5

IMPLEMENTATION AND RESULTS

5.1. Overview

In this chapter the development of the prototype and the tools used are discussed at length. In section 5.2 familiarization of the mobile agent system used for the prototype development and the cryptographic service provider integrated into the Java platform are presented. Following that the development of the prototype using the selected MAS is discussed. The last section presents the performance of the prototype, agent based application, developed.

5.2. Introduction to Tools Used

5.2.1. MAS Familiarization

There are various implementations of mobile agent systems in the market today. Some of the most popular implementation includes: Grasshopper (IKV^{++}) [11], Aglets (Originally IBM Japan) [12, 13] and Telescript (General Magic) [14]. A complete list of existing mobile agent systems can be found at Agent Builder web site [15] along with other non mobile agent based systems, like intelligent and Multi agent systems.

As this thesis work is implemented using Aglets MAP, in the following sections to follow we will concentrate on introducing some of the fundamental concepts associated with this platform.

Aglets Basics:

Aglets is one of the first commercial agent system, originally developed by Mitsuro Oshima and Danny B. Lange, at IBM Tokyo Research laboratory. The original name of the project is Aglet Work Bench (AWB) and IBM was responsible for most of the 1.X release. However the project is now hosted at SourceForge.net as open source [13].

Aglets is completely written in Java and includes a complete Java mobile agent platform, with a standalone server called Tahiti along with a library that allows developers to build mobile agents. It is worthwhile to note that in this system the word Aglets refers to the complete system while the word aglet refers to the mobile agent which is developed by the system [6].

Architecture Overview:

The Aglets architecture consists of two layers (runtime layer and the communication layer) and two Application Program Interfaces (APIs) that define interfaces for accessing their functions. Figure 5.1 shows the Aglet architecture with the two layers and the API [16].

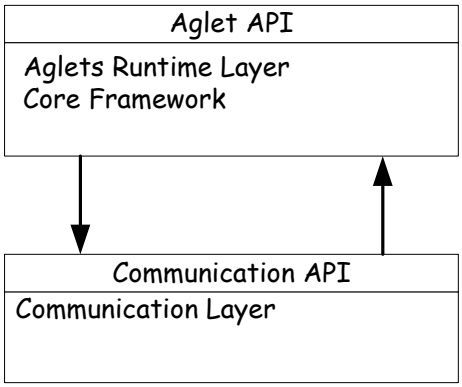

Figure 5.1 The Aglet architecture

The Aglets runtime layer implements Aglets interfaces such as AgletContext and AgletProxy. It also consists of a core framework and subcomponents. The core framework provides the fundamental mechanisms to Aglet execution, i.e. serialization and deserialization of Aglets, class loading and transfer, and reference management and garbage collection [6].

The Aglets runtime itself has no communication mechanism for transferring the serialized data of an Aglet to destinations. Instead, the Aglets runtime uses the communication API that abstracts the communication between agent systems. This API defines methods for creating and transferring agents, tracking agents, and managing agents in an agent-system in independent way. The current Aglets uses the Agent Transfer Protocol (ATP) as the default implementation of the communication layer. The modeling of ATP, which is an application-level protocol, is based on

HTTP. The ATP is independent of platform for transferring mobile agents between networked computers. While mobile agent may be programmed in many different languages and for a variety of vendor specific agent platforms, ATP offers to handle agent mobility in a general and uniform way. To enable remote communication between agents, ATP also supports message passing. Aglets uses ATP for agent transfer and (Remote Method Invocation) RMI for message exchange [17].

The Aglet Model:

Now let us take a closer look at the model underlying the Aglets API. The model defines a set of abstractions to leverage mobile agent technology in Internet like open wide area networks. The key abstractions are aglet, proxy, context and message [6].

Aglet: An aglet is a mobile java object that visits aglet enabled hosts in a computer networks.

Proxy: A proxy is a representative of aglet. It serves as a shield for the aglet to protect the aglet from direct access to its public methods. The proxy also provides location transparency for the aglet.

Context: A context is aglet's workplace. It is a stationary object that provides a means for maintaining and managing running aglet in a uniform execution environment, where the host system is secured against malicious aglet. One node in a computer network may run multiple servers and each server may host multiple contexts. Contexts are named and can be located by the combination of their server address and their name.

Message: A message is an object exchanged between aglet. It allows for synchronous as well as asynchronous message passing between aglets. Message passing can be used by aglets to collaborate and exchange information in a loosely coupled fashion.

Fundamental Operation of Aglets:

Figure 5.2 summarizes the aglets' fundamental operations, namely, creation, cloning, dispatching, retraction, deactivation, activation, disposal, and messaging.

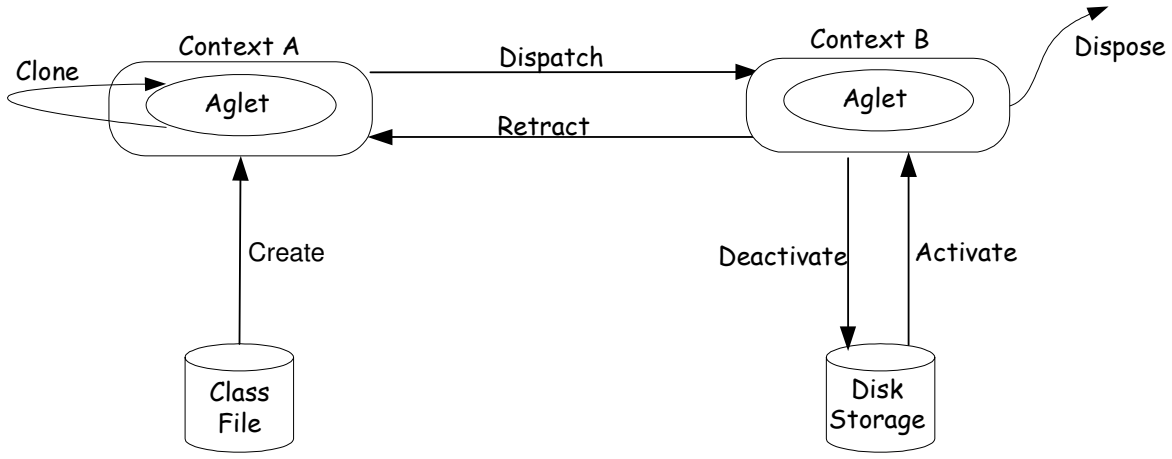

Figure 5.2 Depicting the aglet's life cycle

The Aglet Event Model:

The Aglet programming model is event-based. The model allows the programmer to "plug in" customized *listeners* into an aglet. Listeners will catch particular events in the lifecycle of an aglet and allow the programmer to take action accordingly. Three kinds of listener exist:

Clone Listener: Listens for cloning events. One can customize this listener to take specific actions when an aglet is about to be cloned, when the clone is actually created, and after the cloning has taken place.

Mobility Listener: Listens for mobility events. One can use this listener to take action when an aglet is about to be dispatched to another context, when it is about to be retracted from a context, and when it actually arrives in a new context.

Persistence Listener: Listens for persistent events. This listener allows the programmer to take action when an aglet is about to be deactivated and after it has been activated [6].

A Tour of the Aglet API:

The aglet API is used by programmer's to create and operate aglets. It contains methods for initializing an aglet, message handling, dispatching, retracting, deactivating, activating, cloning and disposing of the aglet. The aglet API is simple and yet flexible. Created in the spirit of Java and representing a lightweight pragmatic approach to mobile agents, the aglet API is a Java package (aglet) consisting of classes and interfaces, most notably: Aglet, AgletProxy, AgletContext and Message.

Aglet Class:

The Aglet class is the key class in the Aglet API. This is the abstract class that the aglet developer uses as a base class, when he or she creates customized aglets. The Aglet class defines methods for controlling its own life cycle. Namely, methods for cloning, dispatching, deactivating and disposing of itself. It also defines methods that are supposed to be overridden in its subclasses by the aglet programmer and provides the necessary "hooks" to customize the behavior of the aglet. These methods are systematically invoked by the system when certain events take place in the life cycle of an aglet.

AgletProxy Interface:

The AgletProxy interface acts as a handle of an aglet and provides a common way of accessing the aglet behind it. Since an aglet class has several public methods that should not be accessed directly from other aglets for security reasons, any aglet that wants to communicate with other aglets has to first obtain the proxy and then interact through this interface. In other words, the aglet proxy acts as a shield object that protects an aglet from malicious aglets. Another important role of the AgletProxy interface is to provide the aglet with location transparency. If the actual aglet resides at a remote host, the proxy forwards the requests to the remote host and returns the result to the local host.

AgletContext Interface:

An aglet spends most of its life in an aglet context. It is created in the context, it goes to sleep there, and it dies there. When it travels in a network, it really moves from context to context. In

other words, the context is a uniform execution environment for aglets in an otherwise heterogeneous world. The AgletContext interface is used by an aglet to get information about its environment and to send messages to the environment, including other aglets currently active in that environment. It provides means for maintaining and managing running aglets in an environment where the host system is secured against malicious aglets.

Message Class:

Aglets communicate by exchanging objects of the Message class. A string field named "kind" distinguishes messages. This field is set when the message is created. The second parameter of the message constructor is an optional message argument [6].

5.2.2. Cryptographic Service Providers

As pointed out in the proposed solution, the prototype development exploits cryptographic concepts to some extent. Hence cryptographic algorithms need to be incorporated in the developed program. Fortunately Java comes up bundled with a number of cryptographic algorithm implementations, providers. On top of that it has a support for a seamless integration of a third party implementation of cryptographic services providers (just providers) through its Java Cryptographic Architecture (JCA) [18, 19].Various providers exist for the Java platform, some are free for academic purposes while others are extremely costly. In the developed prototype we use a cryptography provider called Bouncy Castle. The procedure, concerned with the installation of additional providers, is given in the Appendix section (*Installing and Configuring a Provider).*

The Bouncy Castle also called BC provider offers a pure Java implementation of different cryptographic algorithms. A range of other crypto based products can also be found on the site [20]. It is a free cryptographic package, where its running and maintenance are carried out by volunteers. It has a support for a wide range of various cryptographic algorithms at various levels of strength. To its reputations its jar file is signed by Sun, so that it can be easily integrated into the platform. This provider is integrated into the Java platform and its various crypto packages are extensively used to provide cryptographic services for the various part of the agent system.

5.3. Implementation

To realize the concepts developed throughout the thesis carrying out period, a number of classes are developed. Some of them forms the basic components of the prototype: like statA, mobileA and Agentapp, while other classes are developed to implement the Graphical User Interface (GUI) and other portions of the sample mobile agent based application. Still a number of other classes are developed to compare the performance of the mobile agent that acts or behaves according to the proposition, with those doing it in the usual way. In this section we will discuss in detail the make up of the classes mentioned above.

5.3.1. Utility Classes

Destn:

This class encapsulates the addresses of the nodes to be visited by the mobile agent, including the addresses of its home and its trusted server. It is through this class that the separation of the other parts of the mobile agent from its list of destinations is implemented. The object of this class is constructed based on user's preference and order of list of hosts it wants the mobile agent to visit. The main methods of the class are depicted in figure 5.3.

Destn
Destn(ArrayList, ArrayList) getHosts(): ArrayList getTS(): ArrayList getHome(): URL isAtHome(Aglet): Boolean isAtTS(Aglet): Boolean goToNext(Aglet) goBackHome(Aglet) getHomeURL(): URL getAgletURL(Aglet): URL

Figure 5.3 Methods of Destn class

ProxyH:

This class holds the proxy of the main interacting parties of the countermeasure (the mobile agent, the stationary agent which realizes ASE and the agent based application). As stated in the familiarization section to Aglets mobile agent platform, aglets exchange information first by encapsulating it inside a message object and then use the proxy of the other party to send the message object. Hence this class serves as the container of the proxy of the main interacting components of the system. It provides a number of methods to keep its instance variables valid. In the current implementation of aglet the proxy of a moving agent obtained while it is at one host becomes invalid as it moves from one host to another , hence needs to be constantly updated as the mobile agent migrates. But the proxy of a stationary agent remains valid throughout its life time. The main methods of the class are depicted in figure 5.4.

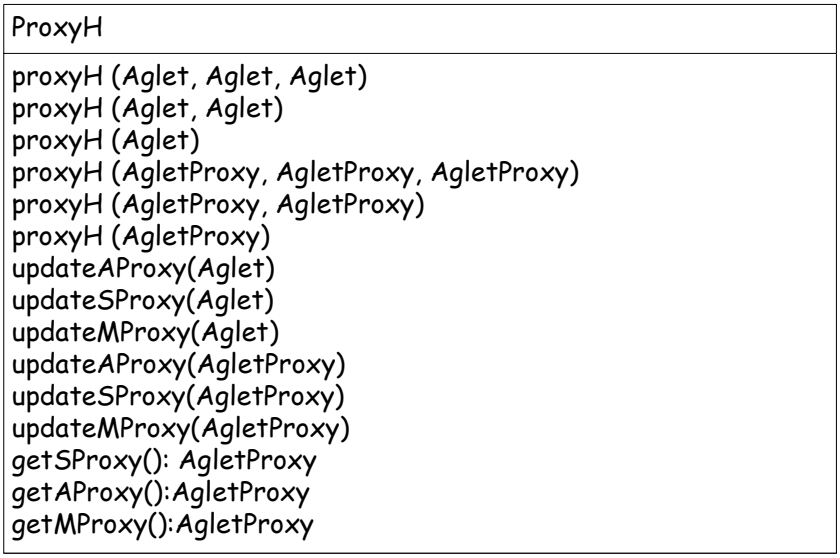

Figure 5.4 Methods of ProxyH class

CipherCls:

This class provides cryptographic services to the whole program especially to the agent based application and to the mobile agent using Legion of Bouncy Castle provider implemented algorithms. It provides various methods to sign object, verify a signed object, encrypt and decrypt

as well as methods to retrieve keys for cryptographic operations. Methods of the class are depicted in figure 5.5.

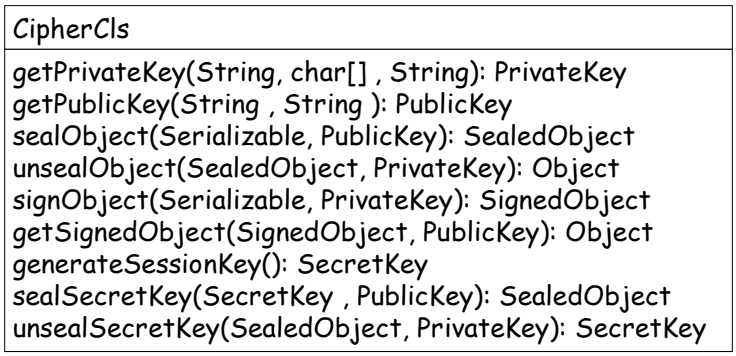

Figure 5.5 Methods of CipherCls class

5.3.2. Components of the Prototype Application

In this section we will see in detail how the implementation of the major components of the proposed countermeasure (AgentApp, mobileA, statA) is carried out.

statA:

This class is used to realize the temporary storage element (ASE) concept. A statA is created by each MA inside a trusted server on its first visit there. It is used to temporarily store partial results that the mobile agent retrieves as it visits different hosts in the agent space. In the prototype the statA stores the information in encrypted forms, so even it doesn't know what it is storing about. Later on the statA handovers all the partial results it has been accumulating to the corresponding mobile agent at the end of the mobile agent's missions on its way back to its home. After that it disposes itself to handover the resources it has been using back to the trusted servers.

A code fragment showing the major parts of statA is depicted below.

```
public class statA extends Aglet
{ ...
```

As shown in the above code, class declaration, this class extends Aglet class so literally it is an agent as per the Aglet mobile agent platform. It is a stationary agent as it does not implement mobilityListener, that would have make it a mobile agent, as we will see for the mobile agents.

```
. . .
     public ArrayList statRes;
     public ArrayList statKeys;
     public proxyH prxy;
. . .
```

The instance variable statRes stores temporarily the information the agent sends while visiting different nodes in the agent space. statKeys instance variable holds a symmetric key that the mobile agent generates randomly as the MA visits various hosts in the agent space. This key is used to protect both the confidentiality and integrity of the partial information as it is sent to the ASE. Note both instance variables hold scrambled data: statRes(information sealed by symmetric key), statKeys (the symmetric key sealed by the public key of the home).

The instance variable prxy is of type ProxyH. It holds the proxy of the three interacting parties in the proposal (the mobile agent, stationary agent and application agent) to facilitate the exchange of information among themselves.

```
. . .
public void onCreation (Object init)
{
     prxy = (proxyH)init;
     statKeys=new ArrayList();
     statRes=new ArrayList();
     prxy.updateSProxy(getProxy());
}
. . .
```

`onCreation(Object init)` is a method that every agent that extends Aglet inherits and this method is called only once in the life cycle of the agent. It is usually used in programs to initialize instance variables of the agent. It also provides a mechanism to pass an array of objects containing important information from the creator of the agent down to agent being created. As shown in the above code fragment, the MA that is responsible for the creation of the statA passes to it a ProxyH object which contains important proxy information. In the code that follows the statA updates its own proxy instance variable, hold inside the proxy object using the `updateSProxy()` method. `getProxy()` method inherited from Aglet class returns the proxy of the current aglet, i.e. the stationary agent.

The next section of block of code inside the statA is shown below

```
public boolean handleMessage(Message msg)
{
if (msg.sameKind("MobieAway"))
{
     SealedObject sos=(SealedObject)msg.getArg("keys");
     statKeys.addsos);
     SealedObject sor=(SealedObject)msg.getArg("infos");
     statRes.add(sor);
     return true;
     }
else if(msg.sameKind("MobileBackV"))
{
     Message newmsg = new Message("MobileH");
     newmsg.setArg("keys", statKeys);
     newmsg.setArg("infos", statRes);
     msg.sendReply(newmsg);
     return true;
}
```

```
return false;
}
```

The above code snippet shows the behavior of the stationary agent as it receives various kinds of messages from the mobile agent. In mobile agents that implements aglet (refer to aglet familiarization) every message that is sent to the aglet is handled inside `handleMessage (Message msg)`. Let us see each of the conditions inside the handle method.

The first *if case* is satisfied, if a message object of kind "MobileAway" is sent to the stationary agent. In the prototype development, such kind of message object is sent by the mobile agent, while it is visiting different hosts in the agent space. This same message object also encapsulates the information the agent retrieves from the corresponding hosts.

The statA uses `msg.getArg()` method to retrieve this information ,which is a sealed object. This same information, as stated in the proposition, is stored in the statA temporarily using `statRes.add(sor)` statement. Similar techniques are used to retrieve the second argument sent by the mobile agent and store it in the corresponding temporary storage.

The second *if case* will be satisfied, when the message object of kind "MobileBackV" is sent to the statA. In the prototype development such kind of message object is sent to the statA by the mobileA, when the MA has finished touring various hosts in the agent space and returns back to collect all the information it has stored temporarily while it is on the move. The statA handovers all the information it has accumulated by constructing a message object of kind "MobileH" encapsulating the information's.

MobileA:

This class is used to realize the actual mobile agent, which moves from one node to another node to perform the computation. Its actions are tuned according to the concept developed in the first part, i.e. is the mobile agent after being dispatched from its home, first goes to its trusted server (TS) creates its stationary agent (ASE) there, that will be used to temporarily store partial results

while it is on the move. The agent at the end of its mission, goes back to the trusted server to collect all the information it has stored so far and returns back to its home.

A discussion on the major parts of the MA code is shown below along with the corresponding code snippet.

```
...
public class mobileA extends Aglet implements MobilityListener
{
...
```

This class by virtue of extending the base class (Aglet) and implementing mobilityListener, it becomes an agent that can listen and react to mobility events(a mobile agent).

```
...
public SignedObject dsDstn;
public String alias="certvirtual";
public int startTrip=1;
public String agletName="statA";
public Message msgInfo= null;
public proxyH prxy;
```

Shown above are some of the most important instance variables used inside the MA class. The MA uses the instance variable startTrip while it is at the trusted server to take on appropriate action depending on whether it is there first or after completing its mission. The instance variable prxy of ProxyH as in the statA is used to hold the proxy of agent that it wants to interact with, in this case the proxy of both the statA and the AgentApp. The instance variable agletName holds the name, specifically the class name, of the stationary agent that this mobile agent will create on the TS.

```
...
public void onCreation(Object init)
{
```

```
        Object [] axcptd =(Object [])init;
        prxy =(proxyH)axcptd[0];
        dsDstn=(SignedObject)axcptd[1];
        Destn dstn=null;
        dstn=(Destn)CipherCls.getSignedObject(dsDstn,
        CipherCls.getPublicKey(alias));
        addMobilityListener(this);
        dstn.goToNext(this);
}
```

As said earlier, the `onCreation(object init)` method is called when the aglet, MobileA, is created. In the prototype, the MA is created by the agent application. It, the application, passes to the MA two objects, one that is a digitally signed destination object and another is a ProxyH. Each of these objects are passed as an array of objects and hence need to be retrieved using appropriate *index* and *casted* to appropriate type. Next the MA unsigns the digitally signed destination object using the utility class's `CipherCls.getSignedObject()`.To unsign the signed object, it needs the public key of the corresponding private key at its disposal. This public key is obtained using the utility class's `CipherCls.getPublickey(alias)` method. The object retrieved, unsigned, is casted to Destn object and it is ready to be used. Later on this object's `goToNext()` method is invoked to dispatch the MA to the next node in its list of destinations. Literally according to the proposition, it goes to the TS and doesn't have any thing to do when it is created. The line of code `addMobilityListener(this)` tells this mobile agent to listen to mobility events and react according to the way 'this' is coded.

```
public void onArrival(MobilityEvent event)
{
        Destn dstn=null;
        dstn=(Destn)CipherCls.getSignedObject(dsDstn,
        CipherCls.getPublicKey(alias));
```

```
if (dstn.isAtTS(this))
{
  if(startTrip==1)
     {
     prxy.updateSProxy(getAgletContext().createAglet(getCode
     Base(), agletName, prxy));
     startTrip=0;
     dstn.goToNext(this);
     }
else{
     Message msg= new Message ("MobileBackV");
     msgInfo=(Message)prxy.getSProxy().sendMessage(msg);
     dstn.goBackHome(this);
     }
}
```

The above code fragment shows a section of code inside `onArrival(MobilityEvent event)` methods. This method is one of the three methods including `onDispatching (MobilityEvent event)` and `onReverting(MobilityEvent event)`, that agents implementing mobilityListener interface need to define. It is executed every time a mobile agent arrives at another node. Usually defined under this method is series of tasks that the mobile agent does as it arrives at a node. In the above case, prototype development, in order to take on appropriate action as the MA arrives to a node, a number of test conditions are used. In the sections to follow we will describe when each of the test condition is satisfied and a range of tasks that will be performed if indeed so.

The first if case if `(dstn.isAtTS (this))` tests whether the mobile agent is currently at its TS home or not. This condition is tested using `isAtTS(this)` method of Destn class. The method compares the URL of the current mobile agent using the argument this with the URL of

the TS it hold insides. The condition will be satisfied if the mobile agent is at the TS. But as per the proposition the mobile agent could be at there at two different instances, at first and at last. In each case it performs various kinds of tasks.The if's inside uses the variable startTrip to determine whether the mobile agent is there at first with a set startTrip value or at the end of its mission with a reset startTrip value.

If the agent is there at first it performs the following range of tasks:

- Gets the context of the platform it is running at using the `getAgletContext()` method of Aglet,
- Creates its temporary data storage (ASE) inside the trusted server, using AgletContext's create aglet method,
- Passes all the necessary information's the method needs through it s arguments (code base, class name of the agent and others) to the agent being created using object init argument,
- Updates instance variables corresponding to the proxy of the stationary agent, using the proxy of the stationary object just returned and
- Uses the dstn go to next this method to select the next node next to it.

If the agent is at TS after completing its task, it performs a range of tasks as pointed out below.

- The mobile agent constructs a message object of "MobileBackV" ,mimicking the fact that it is back at TS,
- Sends the constructed message object, using the proxy of the stationary agent it retrieved from the prxy instance variable's `getSProxy()`,
    - The stationary agent when receives message of such type will send all information it has stored,
    - The information is sent as a message object by the stationary object and stored in a similar message object instance variable,
- The mobile agent invokes the method dstn`.goBackHome()` to return back to its home , 'MISSION ACCOMPLISHED'.

The next if else is shown below.

```
else if (dstn.isAtHome(this))
     {
          prxy.getAProxy().sendOnewayMessage(msgInfo);
          dispose();
     }
```

Similar to the case of being at TS, this if else tests the condition whether the MA is back at home or not. When the condition is satisfied it performs the following tasks:

- Handovers all the result is has collected from the visited hosts, back to the user (agent based application), by retrieving its proxy from the proxy object,
- Disposes of itself, as it is no longer needed.

The last and the remaining if case will be run when either of the above conditions is not met and this means that the agent is neither at home nor at TS. Where else can it be? Right, it is doing its job at some other context, touring the different hosts for the information. The major tasks performed are shown in the code fragment below.

```
else{
     PublicKey pubk=CipherCls.getPublicKey(alias);
     SecretKey secKey = CipherCls.generateSessionKey();
     SealedObject encKey=CipherCls.sealSecretKey(secKey,pubk);
     String result = System.getProperty("os.name");
     SealedObject encRes=CipherCls.sealObject(result,secKey);
     Message msg = new Message("MobieAway");
     msg.setArg("keys", encKey);
     msg.setArg("infos", encRes);
     prxy.getSProxy().sendOnewayMessage(msg);
     dstn.goToNext(this);
    }
```

The agent executes the security protocol developed to protect parts of the mobile agent, while the agent is touring the potentially malicious hosts in the agent space. There, it performs the following range of tasks:

- It retrieves the public key of the Agentapp using the utility method's `CipherCls.getPublicKey()`,
- Generates a random symmetric key using `generateSecret()` method of CipherCls,
- Next the symmetric key generated is sealed(encrypted), using `sealSecetKey()` method of the CipherCls object,
- Does its job,
- Encrypts the data it obtained, using `sealobject()` method of CipherCls class,
- Constructs a message object that will be sent to the stationary agent that encapsulates both the retrieved object and the sealed symmetric key and
- Moves on, continues its journey, either to a next host or back to TS depending on whether it's at the middle of its journey or at the end of its host visit mission.

All of the discussion above pertains to what the mobile agent does, when it is visiting various kinds of nodes. This wraps up the discussion of the implementation of the mobile agent, next we will look at the main application.

AgentApp:

This class realizes the agent based application located at the home computer. It is employed by the user to create and dispatch a mobile agent that acts on its behalf to perform a range of tasks on the hosts list provided. This class constitutes a number of other classes that collects input information from the user, about the mobile agent to be dispatched in a user friendly manner through GUI's.

The supplemental classes are: MobileUIF, pathDlg, infoDlg and aboutDlg. A brief description of each of the classes is presented below.

pathDlg:

This class implements a dialog box that lets user's to enter the address of the list of hosts the mobile agent is going to visit, as shown in figure 5.6. It also provides a choice for the user to enter the address of the TS.

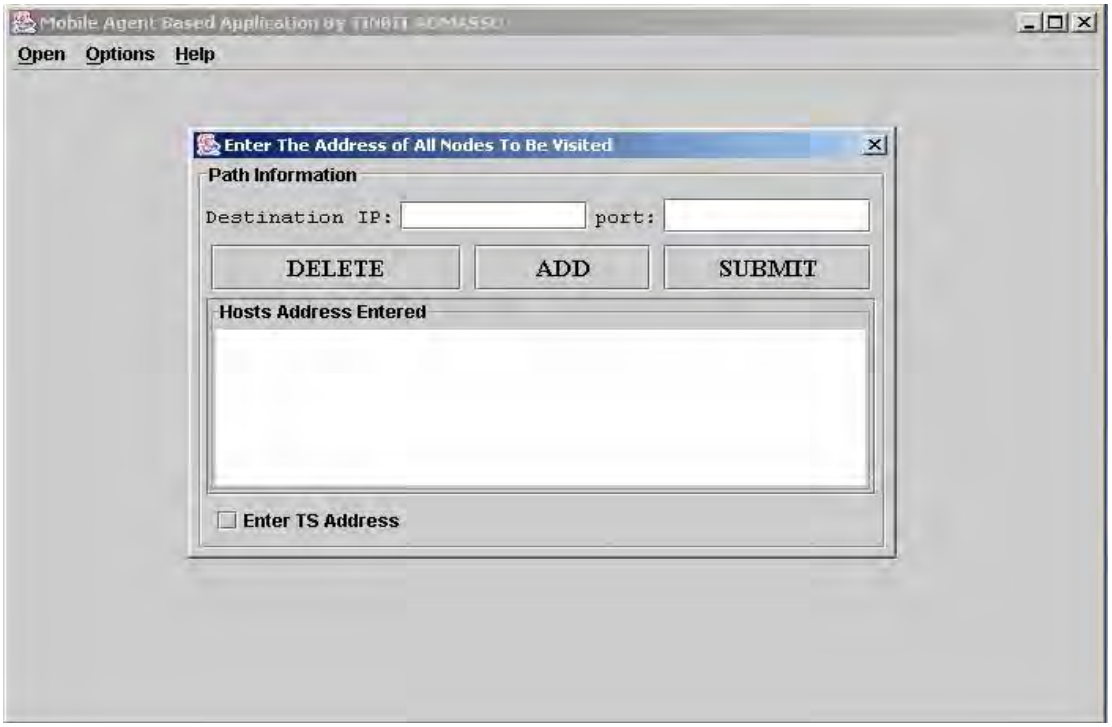

Figure 5.6 The path dialog box

MobileUIF:

This class develops the main window (GUI) of the application. In other words, it is the face of the application. Users see and interact with it, when they run the agent based application.

infoDlg:

This class shows the information the mobile agent has collected, from the list of hosts it visited,as shown in figure 5.7.

AboutDlg:

This class implements a dialog box that provides some information about the prototype developed.

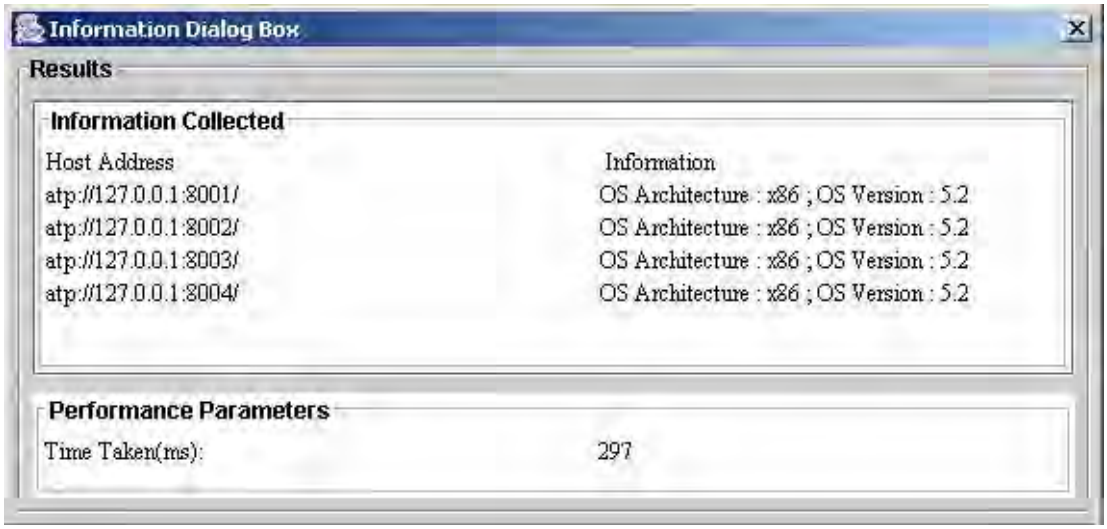

Figure 5.7 Results dialog box

Now let us return back to the main class, AgentApp, and carry on describing its main parts.

```
public class AgentApp extends Aglet implements ActionListener
{ . . .
```

The above code depicts the class declaration of the AgentApp. As it can be seen, it extends Aglet as in the case of mobileA and statA, implying that it is an agent too. This is only done so in the prototype, to exploit the famous information exchange mechanism supported by aglet's platform: message object. The declaration also shows, the class implements the interface ActionListener, so that it can listen and react to events as the user interacts with the GUI components of the user interface. The class is stationary agent as it does not listen to mobility events.

```
public mobileUIF mobui ;
public JMenuItem crtMenuI , DispaMenuI , setinMenuI;
public ArrayList clrKeys=null;
public ArrayList clrInfos=null;
```

```
public ArrayList abahsts=null;
public ArrayList abavhms=null;
public SignedObject dsDstn=null;
public Destn dstn;
public PrivateKey prvk;

public String agletName = "mobileA";
public String alias="kprvirtual";
public String password="virtual";

public long send =0;
public long arrived =0;
public long lived =0;

public proxyH prxy;
```

The code snippet above shows some of the most important instance variables of the AgentApp class. mobui is an instance of mobileUIF class and is used to show up the main GUI as the agent based application runs. The instance variables of JMenuItem, crtMenuI, DispaMenuI, setinMenuI, are used to get a reference to some of the most important menu items inside the mobieUIF class. The AgentApp using these instance variables codes how each should behave as the user interacts with the corresponding menu items. Abahsts and abavhms instance variables are of ArrayList types and holds the list of hosts and TS the mobile agent visits respectively. Instance variables ArrayList clrKeys and ArrayList clrInfos are used to hold the unscrambled keys and results respectively, to show it up to the user. Instance variables SignedObject dsDstn and Destn dstn hold the digitally signed and the unsigned destination object respectively. The instance variable AgletName = "mobileA" holds the class name of the mobile agent that is going to be dispatched by the AgentApp. The next three instance variables are used to access and store cryptographic keys from the keystore as described as follows. Password="virtual" holds the password of the whole of the keystore. alias="kprvirtual" holds a ‘name or alias’ that

differentiates the public key of the application with the other public keys available in the keystore and at last PrivateKey prvk holds the private key to be retrieved from the key store and will be stored in the prvk instance variable. The last instance variables of type long are used to capture performance parameter of the mobile agent.

```
public void onCreation(Object init)
{
     prxy = new proxyH(this);
     abavhms = new ArrayList();
     mobui = new mobileUIF() ;

     crtMenuI = mobui.getCreateMI();
     crtMenuI.addActionListener(this);
     DispaMenuI = mobui.getDispatchMI();
     DispaMenuI.addActionListener(this);
     setinMenuI = mobui.getSettingMI();
     setinMenuI.addActionListener(this);

     clrKeys=new ArrayList();
     clrInfos=new ArrayList();

     mobui.show();
}
```

The above code depicts the major actions undertaken as the agent based application's `onCreation()` method is called. As it can be seen, the instance variables are initialized here. The last line of code, `mobui.show( )`, brings up the main GUI or face of the application.

```
public boolean handleMessage(Message msg)
{
     arrived = new Date().getTime();
```

```
        lived = arrived - send;
        prvk=CipherCls.getPrivateKey(alias, password.toCharArray());
        Iterator kitr=((ArrayList)msg.getArg("keys")).iterator();
        Iterator resitr=((ArrayList)msg.getArg("infos")).iterator();

        while (kitr.hasNext())
        {
        SealedObject sldKey=(SealedObject)kitr.next();
        SecretKey secKey=CipherCls.unsealSecretKey(sldKey, prvk);
        clrKeys.add(secKey);
        SealedObject sldRes=(SealedObject)resitr.next();
        Object res = new Object();
        res=CipherCls.unsealObject(sldRes, secKey);
        clrInfos.add(res);
        }
        infoDlg inf= new infoDlg(mobui ,"Information Dialog Box",
        abahsts ,clrInfos ,lived);
        return true;
}
```

It is the above chunk of code that defines the behavior of the application as it interacts with other agents. The first line of code records the time the agent returns back to home, to compute a performance parameter. Next the private key of the agent application is retrieved to unscramble the information that the mobile agent will be delivering to the AgentApp shortly. The mobile agent passes the information it has collected from the hosts in the agent space using the message object to Iterator for easy traversal. The while loop using the iterator object traverses the scrambled information handed over, to unscramble and store it in a plain form to the instance variables clrKeys and clrInfos. The utility class's CipherCls method `unsealSecretKey()` and `unsealSealedObject()` plays a crucial role in the process. At last the information is passed to infoDlg dialog box to be displayed to the user.

5.4. Results and Performance Comparison

To access the capability of the proposed countermeasure, the following techniques and equipments are used: Two personal computers with 768MB and 512MB of RAM respectively, which run Windows Server 2003 and a number of mobile agents (Proposed MA, DS MA and Normal MA) as described below, are used.

Proposed Mobile Agent: Proposed MA

A mobile agent which is governed by the security protocol and hence performs mobile computation as pointed out in chapter 4.

Digitally Signed Mobile Agent: DS MA

A mobile agent that supports digital signing of the destination object while still performing computation in a way discussed in the proposed solution section.

Normal Mobile Agent: Normal MA

A mobile agent that performs mobile computation in the usual way.

Each of the above mobile agents (Proposed MA, Normal MA and DS MA) are given a similar task to carry out.

Results:

To measure the capability of the proposal towards eavesdropping threat, a test environment is set up using the above mentioned computers as shown in figure 5.8. Computer A takes up the position of trusted server (TS) and computer B runs many host nodes simulated through various port numbers as well as the home node. Ethereal Network Packet Analyzer software is run on computer A. It is open source freely available packet analyzer software that can capture and store packets from a live network for further processing. This packet analyzer software is made to sniff into packets exchanged between the two computers as the various types of MA's do their job. The figure 5.9 shows analysis of the packet captured while the Normal MA and DS MA are in operation.

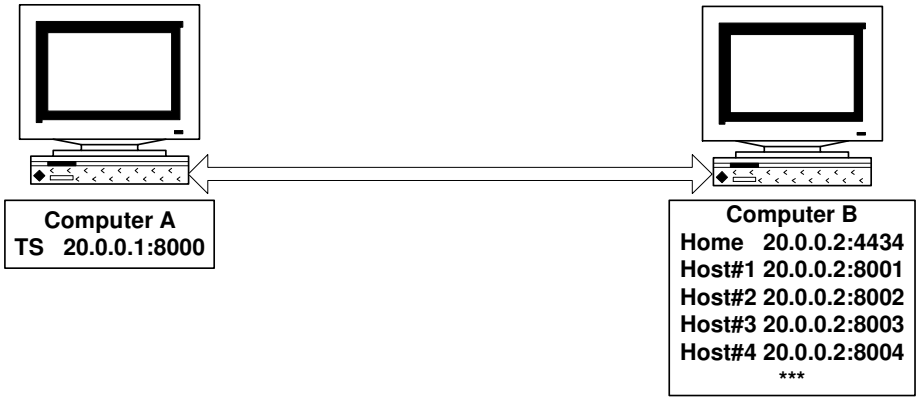

Figure 5.8 Test environment set up

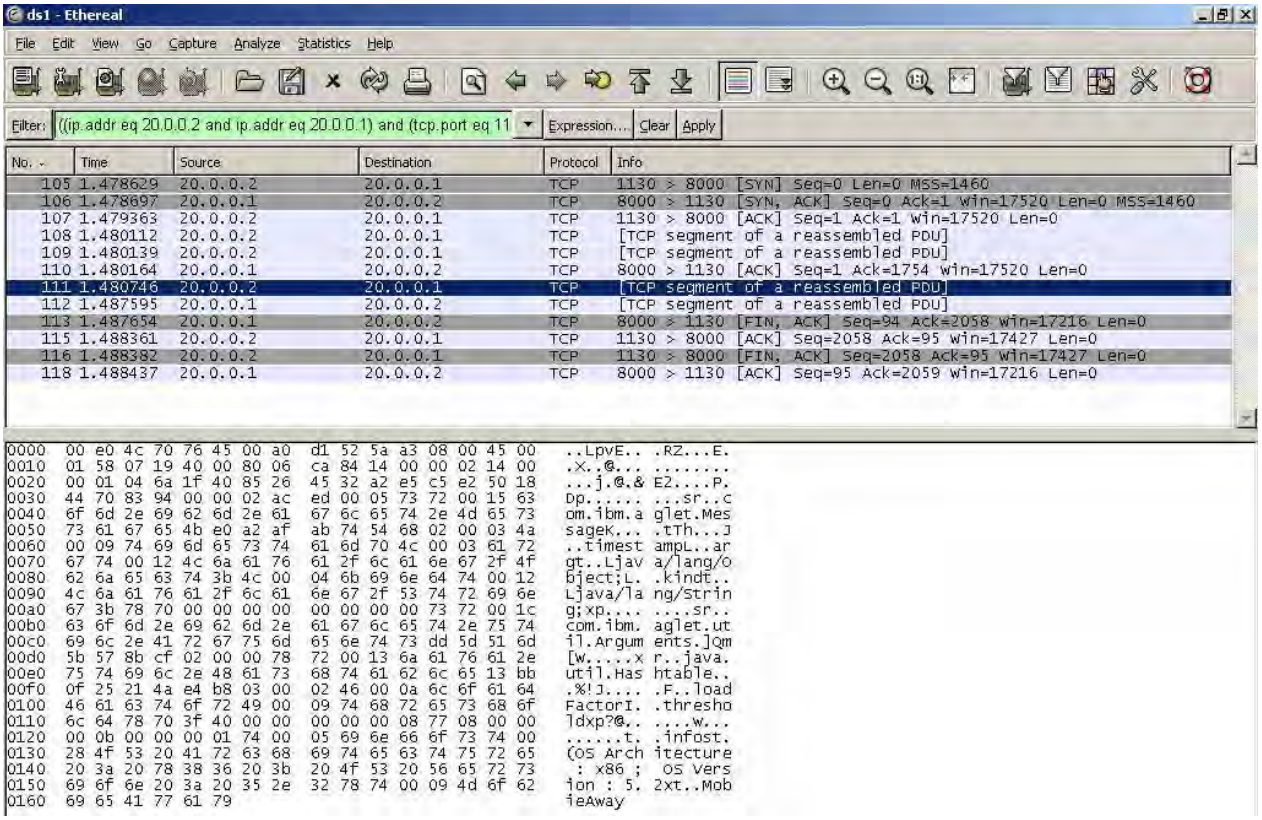

Figure 5.9 Captured packet analysis for DS MA and Normal MA.

As it can be seen from figure 5.9, in either of the cases it is possible to eavesdrop what information is retrieved and exchanged at each host: “OS Architecture: x86; OS Version: 5.2” seen at the bottom right corner of the window.

Figure 5.10 shows analysis of the captured packet while the Proposed MA is in operation. As it can be seen from the figure, unlike the above case since the information is sent to the TS in encrypted form, it is not possible to look in to its content. Hence the security protocol provides the required confidentiality of the information while it is being stored at ASE.

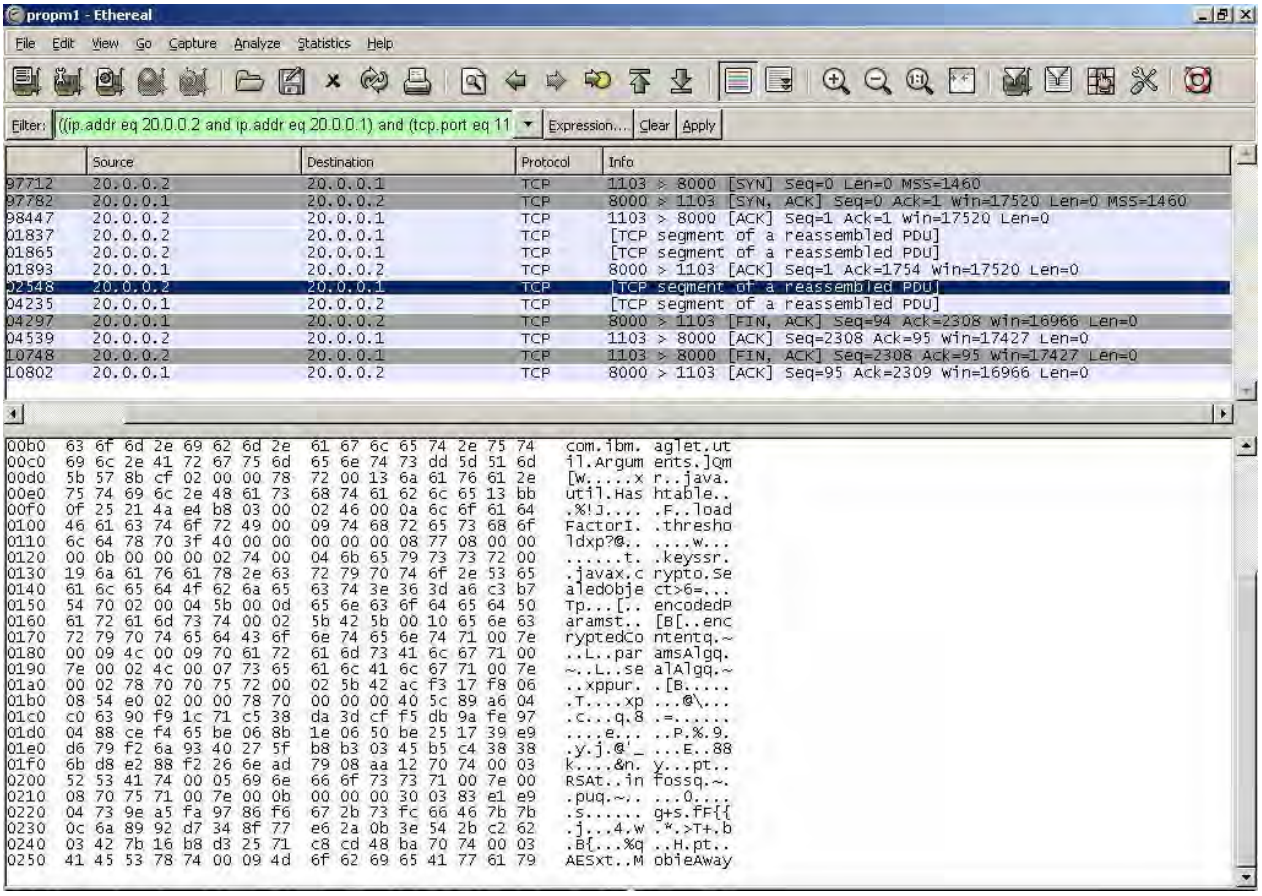

Figure 5.10 Captured packet analysis for Proposed MA.

To test the capability of the proposed countermeasure, towards Alteration threat, a similar test environment as in the above case is used, except that all of the nodes are simulated in computer B as shown in figure 5.11.

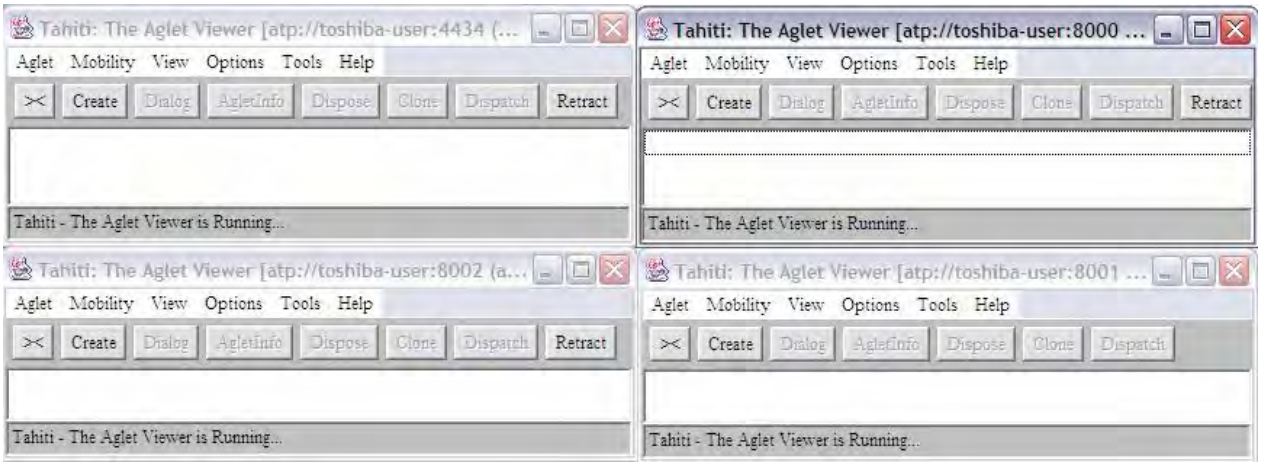

Figure 5.11 Simulating nodes using ports

A hostile node is introduced on a different port number in the same computer, Computer B. This node is planned to behave maliciously towards the Proposed MA. Specifically, it is planned to supply a wrong public key to the MA as the MA arrives there and is in the process of unsigning its digitally signed destination object. But fortunately the MA cannot unsign the signed object using the public key just supplied. This is because the destination object is signed by the private key of the home node, not by a private key which corresponds to the public key supplied by the hostile node. Hence any attempt of alteration of destination object will be detected by the MA as shown in figure 5.12.

Figure 5.12 Detection of alteration threat

Chapter 6

CONCLUSION AND FUTURE WORK

In this thesis we have tried to address the issue of mobile agent security at the same time providing familiarization to the concept of mobile agents programming. The thesis more specifically is an attempt to try to look into a notoriously difficult task as pointed out by many researchers. An attempt has been made to avert some of the malicious host's threats by adopting a number of mechanisms to the way the original computation is made by the mobile agent. The proposed countermeasure introduces the concept of setting up an active storage element in the agent space, as called "home away from home", for partial result storage and separation as well as digital signing of the destination of the mobile agent.

The proposal addresses the issue of alteration of carried result through encryption of partial results which are collected from the visited hosts and are sent to the TS. This prevents the malicious host from altering the carried information. On top of that the integrity and the confidentiality of the information while it is sent to the trusted server and stored there is achieved.

The separation of the destinations form the other parts of the mobile agent helps the home to digitally sign the list of hosts it wants to be visited. On top of that the mobile agent on each destination could verify whether its signed object is tampered with or not before putting that object into use. This mechanism gives a malicious host no chance to alter the list of host addresses the mobile agent has carried and is going to visit.

The proposition also addresses the threats of eavesdropping of collected information both from internal and external malicious host perspective, through partially storing the encrypted information at the TS.

The prototype is implemented using an open source Java based mobile agent platform called Aglets. To incorporate cryptographic functionality into the system so as to provide crypto services to some of the mobile agent's objects a freely available Java cryptography service provider called "BC provider" is used.

The set up of the proposition could also be used in the future to some what address the issue of denial of service (DOS) with respect to mobile agent computation, to send back the information the mobile agent has so far accumulated at its active storage element as last minute retaliation by the trusted server based on some kind of timing mechanism if the TS hasn't heard about the MA for quite large amount of time.

REFERENCES

[1] Dilyana Staneva, Petya Gacheva, *Building Distributed Application with Java Mobile Agents,* International Workshop NGNT.

[2] Rahula Jha, *Mobile agents for e-commerce,* M. Tech. Dissertation, IIT Bombay, India, 2002.

[3] Freeman Yufie Young, *Mobile Computing,* August 21, 2002.

[4] Danny B. Lange, *Mobile Objects and Mobile Agents: The Future of Distributed Computing,* E. Jul(ED): ECOOP'98, LNCS 1445, pp1-12, Springer-Verlag Berlin Heidelberg 1998.

[5] ToDD McDonald, *The World of Mobile Agent Security (MAS),* Department of Computer Science, Florida State University.

[6] Danny B. Lange, Mitsuru Oshima, *Mobile Agents with Java: The Aglet API,* General Magic, Inc ,California USA,1998.

[7] Wayne Jansen, Tom Karygiannis, *Mobile Agent Security,* National Institute of Standards and Technology, Gaithersburg, MD 220899.

[8] E. C. Vijil, *Secuirty Issues in Mobile Agents,* M. Tech. Dissertation,IIT Bombay, India, 2002.

[9] Mousa Altalayleh, Lijiljana Brankovic, *An Overview of Security Issues and Techniques in Mobile Agents,* University of NewCastle, Australia.

[10] Elmarie Bierman, Elsabe Cloete, *Classification of Malicious Hosts Threats in Mobile Agent Computing,* Proceedings of SAICSIT 2002, Pages 141-148, University of South Africa.

[11] Grasshopper, http://ikv.ide , visited on April, 2007.

[12] IBM Japan Aglets, http://www.trl.ibm.com/aglets/index.html, visited on April, 2007.

[13] Aglets Community, http://aglets.sourceforge.net/, visited on May, 2007.

[14] General Magic, Inc, http://www.genmagic.com/, visited on Feb, 2007

[15] Agent Builder Site, http://www.agentbuilder.com/, visited on Feb, 2007

[16] Mitsuru Oshima, Kouichi Oni, Guenter Karjoth, *Aglet's Specification 1.1 Draft,* September 8, 1998.

[17] Danny B. Lange, Yariv Androv, *Agent Transfer Protocol – ATP/0.1,* March 19. 1997.

[18] Java Cryptography Architecture (JCA), API Specification & Reference Guide, 4 August '02.

[19] Java Cryptography Extension (JCE), Reference Guide.

[20] Bouncy Castle, http://www.bouncycastle.org, visited on June, 2007.

[21] Ethereal Network Packet Analyzer Software, http://www.ethereal.com , visited on Nov 2007.

[22] Stan Franklin and Alt Graesser, *Taxonomy for Autonomous Agents,* Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer – Verlag, 1996.

[23] Dan Shiao, *Mobile Agent: New Model of Intelligent Distributed Computing,* IBM China, October 2004.

[24] David Kotz, Robert S. Gray, *Mobile Agents and the Future of Internet,* Dartmouth College, Hanover, New Hampshire, May 15, 1999

[25] Andreas Pashilids, *Security in Mobile Agent Based Network Management,* M. Sc. Thesis Department of Electronic Systems Engineering, University of Essex, UK, 2002.

[26] Rosa Maria, Gil Jranzo, *Modeling Processes Agents negotiating,* Universitat Pomper Fabra, Barcelona ,September 2002.

[27] Kouichi Ono, Hideki Tai, *A Security Scheme for Aglets*, Tokyo Research Laboratory, IBM, 2002, Japan

[28] Levent Ertaul, Jayalalitha Panda, *Mobile Agent Security,* Department of Mathematics and Computer Science, California State University.

[29] David Flanagan, *Java in a Nutshell,* O'Reilly, March 2002.

[30] Cay S. Hortsmam, Gray Cornell, *Core Java™ 2 Volume I & II,* Prentice Hall 2004.

[31] Jan F. Darwin, *Java Cookbook,* O'Reilly, June 2004.

[32] Jonathan B. Knudsen, *Java Cryptography,* O'Reilly & Associates Inc., May 1998.

[33] Luca Ferrari, *The Aglets 2.0.2 User's Manual,* October 6, 2004.

Appendix A:

INSTALLING AGLETS PLATFORM

Installing from a compiled package called aglets-2.0.2.jar.

Decompress the archive:
As Aglets comes in as compressed archive, first decompress it by executing the jar command as shown below:

Jar xvf agelts-2.0.2.jar

The extracted archive has a set of subdirectories as shown below:
bin: contains executable programs for the Aglets 2 platform.
cnf: contains configuration files for the Aglets platform.
public: contains a few examples of agents and should be your root directory as base of your own agents;
lib: contains the Aglets 2 library (as a jar archive) and other libraries required by the Aglets technology.

Install the platform:
Enter the bin directory and then run Ant buildfile build.xml

Set up policy:
The Aglets platform requires entries on the java policy file (usually /.java.policy) to open sockets, execute agents, access local files and so on. Copy entries from the file bin/.agelts.policy (of the aglets installation) in /.java. policy or ask Ant to do it for you, by specifying the following command Ant install-home.

Set up environment variables:
To get the Aglets platform running, set the following environment variables to the installation directory of Aglets: AGLETS_HOME and AGLETS-PATH. Also to run Aglets Platform in a more comfortable way, add the bin directory of the Aglets installation to your Path.

Run the Aglets Server:
Once the Aglets platform and the keystore is installed, the default Aglets Server , Tahiti , can be run through a command agletsd.

If the default keystore is installed use
Username : anonymous
Password : Aglets

Appendix B:

INSTALLING PROVIDERS

Installing a provider is done in two steps: Installing the provider package classes and configuring the provider.

Installing the Provider Classes:
First make sure that the provider classes available so that they can be found when requested. Provider classes are shipped as a JAR (Java ARchive) file.

To install the provider classes, install the JAR file containing the provider classes as an "installed" or "bundled" extension.

The provider JAR-file will be considered as an installed extension if it is placed in the standard place for the JAR files of an installed extension:

<java-home>/lib/ext

Where <java-home> refers to the directory where the runtime software is installed, which is the top level directory of the Java 2 Runtime Environment (JRE) or the jre directory in the Java 2 SDK (Java 2 SDK) software.

Configuring the Provider:
The next step is to add the provider to a list of approved providers. This is done statically by editing the security properties file:
<java-home>\lib\security\java. security

For each provider, this file should have a statement of the following form:
security.provider.*n*=*masterClassName*
This declares a provider and specifies its preference order *n*. The preference order is the order in which providers are searched for requested algorithms when no specific provider is requested. The order is 1-based; 1 is the most preferred, followed by 2, and so on.

masterClassName must specify the fully qualified name of the provider's "master class". The provider vendor should supply you this name.

Appendix C:

Sample Source Codes

AgentApp.java

```
/*
* This program is a mobile agent based application,prototype,
* that sends a mobile agent to the agent space to perform some
task
*/

import com.ibm.aglet.*;
import java.security.*;
import javax.crypto.SecretKey;
import javax.crypto.SealedObject;

import java.net.URL;
import java.util.Iterator;
import java.util.ArrayList;
import java.util.Date;

import java.awt.event.*;
import javax.swing.*;

import java.security.cert.Certificate;
import java.security.*;
import javax.crypto.*;
import java.io.*;
import javax.crypto.spec.SecretKeySpec;

public class AgentApp extends Aglet implements ActionListener
{

     public mobileUIF mobui ;

     public JMenuItem crtMenuI , DispaMenuI , setinMenuI;

     public ArrayList clrKeys=null;
     public ArrayList clrInfos=null;

     public ArrayList abahsts=null;
```

```
     public ArrayList abavhms=null;
     public SignedObject dsDstn=null;
     public Destn dstn;
     public PrivateKey prvk;

     public String agletName = "mobileA";

     public String alias="kprvirtual";
     public String password="virtual";

     public long send =0;
     public long arrived =0;
     public long lived =0;

     public proxyH prxy;

     public URL url;
     public int pt;
     public String prt;


public void onCreation(Object init)
     {

     try{
     prxy = new proxyH(this);
     }catch(Exception ex){
     System.out.println("The   following   error   occured"   +
ex.toString());
     ex.printStackTrace();
     }

     abavhms = new ArrayList();

     mobui = new mobileUIF() ;

     crtMenuI = mobui.getCreateMI();
     crtMenuI.addActionListener(this);

     DispaMenuI = mobui.getDispatchMI();
     DispaMenuI.addActionListener(this);

     setinMenuI = mobui.getSettingMI();
```

```
     setinMenuI.addActionListener(this);

     clrKeys=new ArrayList();
     clrInfos=new ArrayList();

     url=getAgletContext().getHostingURL();
     pt=url.getPort();
     prt=Integer.toString(pt);

     mobui.show();

     }

public boolean handleMessage(Message msg)
{

try{

     arrived = new Date().getTime();
     lived = arrived - send;

     prvk=CipherCls.getPrivateKey(alias,
password.toCharArray(),prt);

     Iterator kitr=((ArrayList)msg.getArg("keys")).iterator();
     Iterator resitr=((ArrayList)msg.getArg("infos")).iterator();

     while (kitr.hasNext())
     {
     SealedObject sldKey=(SealedObject)kitr.next();

     SecretKey secKey=CipherCls.unsealSecretKey(sldKey, prvk);
     clrKeys.add(secKey);

     SealedObject sldRes=(SealedObject)resitr.next();
     Object res = new Object();
     res=CipherCls.unsealObject(sldRes, secKey);
     clrInfos.add(res);
     }

}catch(Exception ex)
     {
```

```
     System.out.println("The   following   error   occured"   +
ex.toString());
     ex.printStackTrace();
     return false;

     }

     infoDlg inf= new infoDlg(mobui ,"Information Dialog Box",
abahsts ,clrInfos ,lived);

     return true;

}

public void actionPerformed(ActionEvent event)
{

Object source = event.getSource();

if (source == crtMenuI)
 {

  pathDlg  pdlg = new pathDlg(mobui , "Enter The Address of All
Nodes To Be Visited");

  pdlg.show();

  abahsts = (ArrayList)pdlg.getHsts();
  abavhms = (ArrayList)pdlg.getVHm();

  return;

 }
else if (source == DispaMenuI)
 {

try{
  dstn = new Destn(abahsts , abavhms);

  prvk=CipherCls.getPrivateKey(alias,
password.toCharArray(),prt);

  dsDstn = CipherCls.signObject(dstn,prvk);
```

```
  Object[] pass = {prxy,dsDstn};

  send = new Date().getTime();

  getAgletContext().createAglet(getCodeBase(), agletName, pass);

  mobui.hide();

 }catch(Exception ex)
     {

     System.out.println("The following here here exception types
captured: " + ex.toString());
     return;
     }
return ;
}

return;

}

}
```

mobileA.java

```
/*
* This program is the proposed mobile agent implementation
* it carries out its task as pointed out in the proposal
*/
import com.ibm.aglet.*;
import com.ibm.aglet.event.*;


import java.security.SignedObject;
import java.security.PublicKey;
import javax.crypto.SealedObject;
import javax.crypto.SecretKey;
```

```
import java.net.URL;
import java.net.InetAddress;
import java.util.Vector;
import java.util.ArrayList;

import java.util.Date;

public class mobileA extends Aglet implements MobilityListener
{
     public SignedObject dsDstn;

     public String alias="certvirtual";
     public int startTrip=1;

     public String agletName="statA" ;
     public AgletProxy slave;
     public Message msgInfo= null;
     public proxyH prxy;

     public URL url;
     public int pt;
     public String prt;

public void onCreation(Object init)
{

try{

     Object[] axcptd=(Object [])init;
```

```
        prxy=(proxyH)axcptd[0];
        dsDstn=(SignedObject)axcptd[1];
        Destn dstn=null;

        url=getAgletContext().getHostingURL();
        pt=url.getPort();
        prt=Integer.toString(pt);

        dstn=(Destn)CipherCls.getSignedObject(dsDstn,
CipherCls.getPublicKey(alias,prt));

        addMobilityListener(this);

        dstn.goToNext(this);

        }catch(Exception ex)
        {
        System.out.print("The    following    error    occured"    +
ex.toString());
        ex.printStackTrace();
        }
}

public void onArrival(MobilityEvent event)
{
        Destn dstn=null;
try{
        url=getAgletContext().getHostingURL();
        pt=url.getPort();
```

```
     prt=Integer.toString(pt);

     dstn=(Destn)CipherCls.getSignedObject(dsDstn,
CipherCls.getPublicKey(alias ,prt));

     if (dstn.isAtTS(this))
     {
          if(startTrip==1)
          {
     prxy.updateSProxy(getAgletContext().createAglet(getCodeBase(
), agletName, prxy));

          startTrip=0;

          dstn.goToNext(this);
          }

          else
          {
          Message msg= new Message("MobileBackV");

          msgInfo=(Message)prxy.getSProxy().sendMessage(msg);

          dstn.goBackHome(this);

          }

     }
```

```
     else if (dstn.isAtHome(this))
     {

          prxy.getAProxy().sendOnewayMessage(msgInfo);

          dispose();
     }

     else
     {

     PublicKey pubk=CipherCls.getPublicKey(alias ,prt);

     SecretKey secKey = CipherCls.generateSessionKey();

     SealedObject encKey=CipherCls.sealSecretKey(secKey,pubk);

     String tmp1 = "OS Architecture : " +
System.getProperty("os.arch");
     String tmp2 = " ; OS Version : " +
System.getProperty("os.version");
     String result=tmp1 + tmp2;

     SealedObject encRes=CipherCls.sealObject(result,secKey);

     Message msg = new Message("MobieAway");
     msg.setArg("keys", encKey);
     msg.setArg("infos", encRes);
```

```
     prxy.getSProxy().sendOnewayMessage(msg);
     dstn.goToNext(this);

     }
}catch(Exception ex)
{
System.out.println("The following error occured" +
ex.toString());
ex.printStackTrace();
}
}
public void onDispatching(MobilityEvent e){}
public void onReverting(MobilityEvent e){}

}
```

statA.java

```
/*
*This class implements a temporary storage place as pointed out
in the proposal
*/

import com.ibm.aglet.*;
import com.ibm.aglet.event.*;
import javax.crypto.SealedObject;
import java.util.ArrayList;

public class statA extends Aglet
{

     public ArrayList statRes;
     public ArrayList statKeys;
     public proxyH prxy;
public void onCreation(Object init)
{
```

```
try{

     prxy = (proxyH)init;
     statKeys=new ArrayList();
     statRes=new ArrayList();
     prxy.updateSProxy(getProxy());

}catch(Exception ex)
{
     System.out.println(ex.toString());
}

}


public boolean handleMessage(Message msg)
{
     if (msg.sameKind("MobieAway"))
     {
try{
     SealedObject sos=(SealedObject)msg.getArg("keys");
     statKeys.add(sos);

     SealedObject sor=(SealedObject)msg.getArg("infos");
     statRes.add(sor);
}catch(Exception ex)
{
     System.out.print("The following error occured"+
ex.toString());
     ex.printStackTrace();
}
     return true;
     }

     else if(msg.sameKind("MobileBackV"))
     {
try{
     Message newmsg = new Message("MobileH");

     newmsg.setArg("keys", statKeys);
     newmsg.setArg("infos", statRes);
```

```
     msg.sendReply(newmsg);

}catch(Exception ex)
{
     System.out.print("The following error occured"+
ex.toString());
     ex.printStackTrace();
}

     return true;
     }

 return false;

}

}
```