uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Lei Jin

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Performance of Low-density Generator Matrix Codes at Short Block Lengths

TITRE DE LA THÈSE / TITLE OF THESIS

Y. Mao

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

T. Yeap

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

A. Karmouch

H. Yanikomeroglu

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Towards the Use of Mobile Agents for Privacy Negotiation

By

Yogesh Kalyani

Thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

In partial fulfillment of the requirements

For the degree Master in Computer Science

Ottawa-Carleton Institute of Computer Science

Faculty of Engineering

University of Ottawa

Your file Votre référence
ISBN: 978-0-494-25792-0
Our file Notre référence
ISBN: 978-0-494-25792-0

Abstract

With the increase in popularity of e-Commerce, concern for privacy has also increased. Users are becoming increasingly concerned about what personal information they could reveal when they go online. Currently, an e-Commerce Website does not provide full support to conduct negotiations concerning personal information between a user and a Web site. A user has to comply with the privacy policy specified on the Web site and has two choices: either they abandon the transaction or accept the privacy practices of the host.

The Platform for Privacy Preferences (P3P) [16] specification, a W3C standard, enables Web sites to specify their privacy practices in a standardized manner. The presence of P3P policies enables users to configure their Web browsers to constrain what they can and cannot do when visiting sites. However, one major limitation of P3P is that no support is given within the specification for negotiation of privacy preferences between a user and a Web site. As e-Commerce is growing, a proper negotiation mechanism should be in place to enhance user control over personal information, so users can decide what personal information they wish to release to the Web site.

This thesis proposes a novel mechanism to address this limitation, focusing on the use of mobile agents with a web ontology language (OWL) for privacy negotiation. Thus far in the literature there appears to be no working mobile agent mechanism that does privacy policy negotiation, particularly in the context of P3P. The architecture proposed here explicitly incorporates the concept of variable end-user requirements with respect to privacy. It is designed keeping in mind a negotiation scenario where the user device has limited resources (i.e., poor connectivity) and cannot negotiate directly with desired Web sites, but, obviously, less constrained user devices may also benefit from this work.

Despite the advantages of a mobile-agent-based architecture, there are several issues which hinder the deployment of mobile agents in real life scenarios. Two major security concerns associated with a mobile agent paradigm are malicious agents and malicious hosts. In this thesis, malicious host concerns have been addressed by proposing a Trace Based protocol. This protocol is designed keeping in mind the desired security properties required, and also considering other factors like resource constrained devices and bandwidth preservation.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I derive immense pleasure in revealing my indebtedness to my thesis supervisor, Dr Carlisle Adams, for his invaluable guidance, unremitting motivation and persistent supervision.

I wish to thank my friends, Mr. Aashish Shah and Mr. Saurabh Bhanot for helping me to get through the difficult times, and for all the emotional support, entertainment, and caring they provided during the preparation of this work. Equally important was the constant support, and encouragement from my parents, brother and sister. Thank you, mom and dad, for your everlasting love and support throughout my life without which this work would not have been possible. To them, I dedicate this thesis.

Chapter 1: Introduction

With the increase in popularity of e-Commerce, concern for privacy has also increased. Users are becoming increasingly concerned about what personal information they may reveal when they go online. In a past privacy survey [61], it was pointed out that 70% of respondents thought that companies took too much personal information and 90% respondents reported that they wanted the ability to control how their information was used after collection. In another privacy survey [62], users were comfortable in giving out general information to Web sites. However, they were often very uncomfortable providing sensitive information like credit card numbers and social security numbers. This gives an indirect impression that Web users currently do not have sufficient control over their personal information. Though Web sites do publish their privacy practices, they are presented in a format that is either too long or too hard for a normal user to understand, which can lead to unintentional consent to release personal information [34]. Currently, it appears that Web sites do not have any mechanism to provide full support for privacy policy negotiation, particularly in the context of P3P. Users have to agree with the privacy practices specified on a Web site to get any service from the site.

The Platform for Privacy Preferences (P3P) [16], a W3C standard, addresses this issue by enabling Web sites to describe their data-collecting practices in a standardized machine readable format, indicating which data are to be collected, for what purposes, with whom it will be shared, and for how long it will be retained. The presence of P3P [16] enables users to configure their web browsers to constrain what they can and cannot do when visiting a Web site. This task includes matching of the Web site privacy policy with user preferences, making appropriate decisions and notifying the user if there is any mismatch between the privacy policy and user preferences. This means a Web site policy specifies how the user's data will be handled and gives users an initial degree of control over their personal information protection concerns by giving a 'take it or leave it' service. That is, users still have to comply with the privacy policy specified on the Web site to get the service; otherwise, they cannot get any service from the Web site. The main advantage of P3P is the

ease of communication between the user and the Web site. However, one major limitation of P3P is that no support is provided within the specification for negotiation of privacy preferences between the user and the Web site. It does provide a very basic negotiation scenario by providing a 'take-it-or-leave-it' scheme. If the Web site and the user do not agree completely on the terms of the exchange, further communication is terminated. To alleviate this problem, earlier drafts of P3P supported a protocol for multi-round negotiation. However, it was believed that this made P3P too complicated, and the protocol was dropped from the specification in later versions [34]. In [73], Thibadeau appreciated the P3P standard as being an outstanding work, but nevertheless, he identified the lack of negotiation contract mechanism as one of the limitations of the P3P standard. As well, Langendorfer and Kraemer [49] pointed out that a negotiation mechanism is a major requirement for privacy-aware applications.

In summary, with the increase in the growth of online activities over the last few years, the privacy concerns have also grown rapidly. As shown in the surveys mentioned [61, 62], users are getting more concerned about their personal information. They need sufficient control to protect their personal information during online activities. As e-Commerce is growing, a negotiation mechanism should be in place to enhance user control over personal information. This will enable users to decide what personal information they could release to the Web during online activities. The negotiation mechanism would allow users to specify a set of alternatives concerning their personal information, which would govern how user information could be released and under what conditions.

Online negotiation can be applied in variety of situations, and even in different forms like one-to-one, one-to-many, many-to-one, and many-to-many. This study focuses on a one-to-one negotiation of user preferences and privacy practices between the user and the Web site. The user defines a set of rules and user privacy preferences beforehand, which in turn governs how negotiation will be conducted with the Web site. It is evident that negotiation increases the chances of getting an appropriate solution to conflicting privacy desires. On the other hand, for client-server architecture, negotiation may increase the network load and latency due to the large number of message transmissions between Web sites and users. It

can also be a time consuming process. Furthermore, if the device is resource constrained and cannot be online all the time (e.g., a wireless device), then it may be difficult to achieve the negotiation.

This study proposes a novel architecture to address the above concerns, focusing on the use of a mobile agent[1] with Web Ontology Language (OWL) [57, 36, 35] for privacy negotiation. OWL specifies the user preferences that are used by the mobile agent to negotiate with a Web site that has a P3P policy. So far, in the literature there appears to be no working mobile agent mechanism that does privacy policy negotiation, particularly in the context of P3P. In this study, we show the benefits of using OWL, but we do not actually use OWL itself in our privacy negotiation. This architecture explicitly incorporates the concept of variable end-user requirements with respect to privacy (e.g., the type and amount of information the user is willing to give to the Web site during online activities such as shopping or registration).

The mobile agent [48] is a goal-directed program that migrates from one host to another, performing actions autonomously on behalf of the user to achieve the goal specified by the user. Some good reasons to use a mobile agent for privacy negotiation include reduced network load and latency, asynchronous and autonomous execution, and reduced client processing [48]. Also, it is a beneficial approach if the device is resource constrained and cannot be online all the time.

The complete negotiation architecture works in four processing phases: a pre-processing phase, a negotiation phase, a post-processing phase, and an agreement confirmation phase. These four phases occur in two rounds of a mobile agent. Pre-processing, negotiation, and post-processing takes place in the first round of the mobile agent itinerary, while agreement confirmation takes place in the second round of the itinerary. These four phases occur in two rounds of a mobile agent. In the first round, the mobile agent visits all the hosts in its itinerary list and does the negotiation with each host. If the negotiation is achieved it collects a signed agent trace offer and travels to another host on its itinerary list. As the mobile agent

[1] We recognize that there are legal concerns with respect to the use of user agents and P3P [3]. This can arise due to inaccurate representations by user agents of a Web site's privacy policy. However, such concerns will not be explored in this thesis.

follows a partially fixed itinerary, it visits all the hosts on its itinerary list, but the order of visits may vary depending upon the mobile agent's decision-making ability. When the mobile agent returns to the origin host, the optimal offer is determined and mobile agent is dispatched again to the selected host to get the final confirmation of the negotiation and to collect the privacy data offer.

Several mobile agent based negotiation mechanisms have been proposed and developed to be used in several different applications like e-Commerce, network management, software distribution, information retrieval, distance education and many more. The mobility and autonomous characteristics of mobile agents make them an attractive solution for all these applications. On the other hand, these characteristics of the mobile agent introduce two major security concerns, which hinder their full-fledged deployment in real life scenarios. These two major security concerns are the malicious agent threat and the malicious host threat. For a host, the concern is that agents visiting it may maliciously try to damage the host environment or data. For an agent, the concern is that the host platforms it visits may contain software to maliciously subvert the goals and integrity of the agent. There have been several effective mechanisms proposed and implemented that address the malicious agent threat including sandboxing, source code verification, proof carrying code, and software isolation. In this study, the malicious host threat will be addressed and a trace based security protocol will be proposed. This protocol broadly achieves three major security properties, namely, data integrity, partial non-repudiation, and data confidentiality.

The protocol works in two rounds. In the first round, the mobile agent collects the agent offer traces from each host (if negotiation is achieved). After returning to the origin host, the owner evaluates the agent offer traces, determines an optimal offer, and again dispatches the mobile agent with the chosen agent trace offer to the selected host to confirm the negotiation and collect the privacy data offer. In this protocol, instead of collecting the actual data offer (privacy data offer) from every host, the agent collects the agent trace offer, which reduces the size of the mobile agent, thereby preserving bandwidth over the network. During the second round, the mobile agent presents to the selected host the agent trace offer over which negotiation was achieved in the first round. The host checks the trace offer and determines its validity. Then the host retrieves and signs the actual privacy data offer and gives it to the

agent. After returning back, the privacy offer is verified at the origin host and actual data is retrieved and compared with the agent offer trace. If the privacy data offer matches with or is the subset of the agent offer trace, the negotiation is said to be successful.

1.1 Objectives of Study

This thesis focuses on the design and partial implementation of the privacy negotiation architecture using a mobile agent. More specifically, the objectives of this study are listed as follows:

- To enhance user control over personal information.
- To design a privacy negotiation architecture specifying an interaction of entities at all stages of the negotiation process, showing how a mobile agent can be used in a privacy negotiation process.
- To implement the proposed trace based protocol for partially fixed itinerary mobile agents.
- To extend Maggi and Sisto's protocol to incorporate enhanced protection from truncation attack for free roaming mobile agents.

1.2 Contribution of the Study

In this study, a mobile agent based privacy negotiation mechanism is proposed, which attempts to show how negotiation of privacy policy can be achieved between users and Web sites. This architecture enhances user control over personal information, so that users can decide what personal information they wish to release to a Web site. A trace based security protocol is been proposed here, which has been implemented, to securely store the results of the negotiation at each host. The proposed mechanism is not limited only to privacy negotiation; it can be applied to a wide variety of situations with slight modifications, if desired.

Two mechanisms are proposed in the study which enhances truncation resilience in the original protocol for a free roaming mobile agent. In the same way, these mechanisms are

not limited only to privacy negotiation, but can also be applied in a wide variety of situations with modifications, if desired.

1.3 Organization of the Study

This thesis is organized into six chapters. Chapter 2 provides a brief overview of the mobile agent paradigm and a review of related work. In chapter 3, a privacy negotiation architecture based on the use of a mobile agent is introduced and described. Chapter 4 presents a description of the proposed trace based security protocol, which is used to securely store the result of the negotiation. Chapter 5 focuses on the implementation details of the proposed trace based protocol, which shows different scenarios in which the protocol can work. A conclusion, thesis summary and suggested areas for future work are presented in chapter 6.

Chapter 2: Literature Survey

This chapter presents a review of literature pertaining to two major aspects of this study: the mobile agent and privacy negotiation. Firstly, a brief description of the mobile agent paradigm, its advantages, and different types of itineraries followed by a mobile agent, as well as the mobile agent framework used in this study, are presented. Then, in the second section, a background study on the various security concerns associated with the mobile agent paradigm is presented. Finally, the third section presents a review of work on privacy, and the associated languages used to describe the privacy practices of the host and to express user preferences. In this study, the term "mobile agent" is interchangeable with the term "agent." Similarly, the terms "remote host," "host," "host agent" and "Web site" are interchangeable.

2.1 Mobile Agent Paradigm

This section presents a brief description of the mobile agent technology, its advantages, and different types of itineraries followed by a mobile agent.

2.1.1 Mobile Agent

A mobile agent is a goal directed program which migrates from host to host performing some actions autonomously on behalf of the user to achieve a goal specified by the user. The mobile agent paradigm [48] introduces several advantages, some of which are described below:

i. Reduced network load: As the mobile agent migrates from one host to another, it brings their execution close to the resource, which results in the saving of a large amount of communication overhead.
ii. Reduced network latency: Network latency is heavily reduced because the mobile agent resides on the platform when it is computing. Furthermore, the mobile agent execution does not depend on latencies in messages sent across slow and unreliable networks.
iii. Asynchronous and autonomous execution: Tasks requiring a continuously open connection between a mobile device and a fixed network are probably not

economically or technically feasible. To solve this problem, tasks can be embedded into the mobile agent, which can then be dispatched into the network. After being dispatched, the agent becomes independent of the process that created it and can operate asynchronously and autonomously. The mobile device can reconnect with the network at a later time to collect the agent. The only interaction is at the beginning of the agent's lifetime, when it is created and sent out to the remote host by the user, and in the end when the agent returns to the user. This makes a mobile agent an attractive paradigm for applications such as mobile computing, where the communication bandwidth is limited or the user cannot be online all the time.

iv. Reduced client load: The mobile agent performs its tasks locally (i.e., at the host platform). This means that the device that sends the mobile agent (e.g., a PDA) can have limited computing power and will not need to utilize excessive power consumption.

2.1.2 Itinerary of the Mobile Agent

Karjoth, et al. [44] introduced three different types of itineraries a mobile agent can follow. The itinerary of a mobile agent is the list of remote host platforms it visits. The itineraries are listed below, from a predetermined itinerary to an itinerary that is not predetermined at all.

i. Fixed itinerary: The agent itinerary and the sequence of visits are set before the agent is dispatched from the origin host [44]. In this type of itinerary, a mobile agent only visits the list of hosts in its itinerary and also follows a strict sequence of hosts which is predetermined at the origin host. For instance, if the mobile agent itinerary list is predetermined at the origin host as $\{P_1, P_2, P_3, P_4\}$, then the mobile agent visits all hosts in the order they are specified (i.e., from origin host to P_1, then from P_1 to P_2 and so forth).

ii. Partially fixed itinerary: In this type of itinerary, the mobile agent visits only the list of the hosts in its itinerary, but the order in which these hosts are visited is not fixed [44]. All remote host platforms that the agent has to visit are set at the start. The agent visits no places that are not on its list, but it is free to choose the sequence in which it visits each host. For example, if the mobile agent's itinerary list at origin

host is determined as $\{P_1, P_2, P_3, P_4\}$, then the mobile agent visits only those hosts which are in its itinerary list, but the order in which those hosts are visited may vary (depending upon the mobile agent). It starts from the origin host and may move to host P_1; from host P_1 it can choose to visit any host among P_2, P_3, and P_4.

iii. Free roaming itinerary: This is a dynamic itinerary and can vary from host to host. The complete itinerary list of the agent is not known at the origin host. Hence, the mobile agent may visit hosts that are not known from the start. As the host can add other hosts in the itinerary list, an agent can freely choose which host it visits next from all hosts that it recognizes during the runtime. For example, if the mobile agent initial itinerary list at the start is $\{P_1, P_2, P_3, P_4\}$ and the final itinerary list when the agent returns to the origin host is $\{P_1, P_5, P_3, P_6, P_2, P_4\}$, it means that the hosts P_5 and P_6 are the newly added hosts on the itinerary list.

2.1.3 Aglets

Aglets [33] are Java objects that can move from one aglet-enabled host to another aglet-enabled host on a network. An aglet that executes on one host can suddenly halt execution, be dispatched to a remote host, and start executing again [33]. When the aglet [47] moves from one host to another, it carries the program code as well as the states of all the objects it is carrying. Upon reaching a target computer, the aglet presents its credentials in the form of recognizable code. Once authorized, it obtains certain privileges to execute and get the data and services which it seeks. In order to fully understand how an aglet works, it is necessary to understand the three main elements of the aglet life cycle: aglet operations, aglet events and aglet messages.

An aglet [33, 47] consists of six main types of operations: creation, cloning, dispatching, retraction, activation/deactivation, and disposal. Each of these operations is described below:

- Creation: The aglet is created and loaded into a context. As soon as it is initialized, it begins execution.

- Cloning: This is another way of creating an aglet in a context. In this case, an already functioning aglet is copied. Once duplicated, the clone has its own identifier and starts running.
- Dispatching: This moves the aglet into a new context. Upon arrival, it starts running in a new thread from its starting point.
- Retraction: This operation recalls an aglet from its present context into another context and inserts it into the context from which the retraction call was executed.
- Activation/Deactivation: An aglet can be stopped and temporarily transferred to a disk. After a certain time the aglet can be activated again in the same context [33].
- Disposal: This operation gets rid of the aglet and allows the Java garbage collector to remove its remains [47].

In this study, we use two aglet operations, creation and dispatching. The aglet event relies on listeners to fire particular actions once an event is caught. The aglet model adds three event listeners: mobility listeners (which catch the changes in aglet location), persistence listeners (which catch changes in the activation/deactivation state of the aglet) and clone listeners (which catch the cloning events). All these listeners can be customized to carry out various types of action depending on the event being caught. In this study, mobility listener events are used and customized in order to meet the requirements of the proposed architecture. A mobility listener tracks the events generated by aglet movements. There are three situations covered: dispatch, arrival and retraction. In all these situations, the aglet is either dispatched, arrives, or is called back to another location. Of these, the present study covers only two situations: arrival and dispatching events.

Aglet messages can be synchronous, asynchronous, and multi or single cast, and may not require a reply [33, 47]. There are three types of aglet messages:

- Message: A message is an object that is exchanged between Aglets. The basic message mechanism is synchronous. The sender waits for a reply. There is also a simple asynchronous mechanism, a one way message, when no reply is needed.
- Future Reply: It is also possible to have asynchronous messages when a reply is needed. The sender carries on while waiting for a reply in the future.

- Reply Set: A reply set can contain multiple future replies.

In order to meet the requirements of the architecture proposed in this study, the Message type is used to communicate between the agents (i.e., synchronous messaging where the sender waits for a reply and can only send one message at a time to only one agent).

2.2 Security Concerns

In this study, we consider a simple model that consists of the mobile agent and the remote hosts. The mobile agent is created at a host, referred to as the origin host, migrates to a list of remote hosts to perform privacy negotiations, and finally returns to the origin host with the results of the negotiation. The mobile agent carries all its data and code when it is dispatched from the origin host. Thus, an origin host can go offline after dispatching the mobile agent and can reconnect at a later time to collect the agent results. The mobile agent [55] consists of two types of data: static data and dynamic data. Static data does not change over the entire life time of the mobile agent, while dynamic data changes due to mobile agent computation, as it migrates from one host to another. As long as the mobile agent remains on the origin host, it is assumed to be safe from tampering or eavesdropping. This assumption no longer holds true when the mobile agent starts migrating from host to host [13]. The mobility and autonomous characteristics of the mobile agent introduces a number of security threats related to host and agent security. Broadly, two major types of threats are imposed by a mobile agent based system: a malicious agent threat and a malicious host threat.

2.2.1 Malicious Agent Threat

A remote host provides the necessary resources and an execution environment to the mobile agent to execute on its platform. If a mobile agent [9, 14] becomes malicious, it can pose several threats to the remote host platform such as masquerading, denial of service, leaking sensitive system information, and altering, fabricating or deleting the files from the remote host. For instance, a mobile agent could masquerade as an authorized agent and gain access to resources to which it is not entitled. Therefore, the remote host must take some measures

to protect the platform from a malicious agent. The threats imposed by a malicious agent can be classified into three categories: integrity, confidentiality and availability.

i. Data Integrity: The remote host resources must be protected from unauthorized modification, erasure, or other means of tampering.
ii. Confidentiality: The sensitive information of the remote host must be protected from leaking through unauthorized channels.
iii. Availability: The remote host must be protected from interference that affects its normal operation and availability of service.

An overview of the existing approaches that address the malicious agent problem is described below:

i. Software fault isolation: This mechanism is also referred to as sandboxing, in which a restrictive execution environment is provided to execute the mobile agent. Each mobile agent can access only the memory space available in its execution environment [40]. This mechanism is ideally suited for situations in which most of the code falls within one trusted domain, since the modules in trusted domains incur no execution overhead [9]. The main drawback of this approach is that it imposes too many restrictions on the mobile agent, which limits its operation [40].
ii. Safe code interpretation: In this mechanism, the code is not directly compiled into machine instructions. Instead, it is compiled into some type of code that can be interpreted (for example: Java byte codes, which can be executed by any operating system having a Java Virtual Machine). The two major advantages of using interpreted code are the ability to verify the safety of the interpreted code before its execution and interoperability with heterogeneous platforms [39].
iii. Signed code: This mechanism is based on source verification instead of code verification. The code producer digitally signs the code with his or her private key and the code consumer verifies the signature on the code before allowing the mobile agent to execute it. The same mechanism can also be used to judge the integrity of the code, as the signed code cannot be modified by any other entity unless it has the private key. The major drawback of this approach is that it is based solely on the reputation of the code producer, not on the actual code safeness [40]. Even if the

source is authenticated, this does not provide any guarantee of safe execution of the code. Hence, this mechanism is ideally suited for trust based domains [9]. Another problem with this mechanism is the authenticity of the digital signature itself: it may be possible that the keys have been revoked or cancelled.

iv. Path histories: The basic idea behind the path histories mechanism is to maintain an authenticable record of the prior platforms visited by the agent, so that a newly visited platform can determine whether or not to allow the agent to execute on it and what resource constraints it needs to apply [76, 40]. This mechanism is based on public key cryptography, in which each host adds a signed entry to the path history by computing a message digest on its identity, the next host in the agent's itinerary and the complete path history up to that point [9, 40]. Upon receiving the path history, the next remote host determines whether to run the agent, by authenticating the previous hosts. The major drawback of this approach is that path verification becomes an expensive process with an increasing path history.

v. State appraisal: The state appraisal mechanism [9] assures that the current state of the mobile agent has not been maliciously altered. In this approach, two types of state appraisal functions are used: maximum function and request function [39]. The maximum function is determined by the origin host, which defines the maximum set of permissions to be given to the agent, as a function of its current state. The request function is determined by the sender (previous host), and defines the set of permissions the sender wants to grant, based on its current state. These privileges are issued by the remote host platform based on the results of the appraisal function and the platform's security policy. The drawback of this approach is the difficulty in making a distinction between the normal result and the malicious result [40].

vi. Itinerary authentication with ad-hoc trust relation: This mechanism utilizes the concept of *one-way* signatures to connect arbitrary hosts in a chain of trust, thus enabling the formation of ad-hoc trust relationships. If all entities involved with the agent can be authenticated, a level of trust can be established, which can then be used for granting or denying execution privileges. The security of this mechanism is based on an assumption that the static and dynamic parts of the mobile agent can be made inseparable. In other words, the agent's functionality is available if and only if both

static and dynamic parts correspond to the same agent or if static and dynamic parts of the agent are mutually authenticating whenever they possess this property.

vii. Proof carrying code: The proof carrying code [52] is a technique by which the host ensures that an untrusted code provided by the code producer satisfies the safety policy of the host platform. This policy is generated and predefined by the host, which guarantees that the code is safe to execute, if the untrusted code satisfies the safety policy. Any attempt to tamper with either the code or the safety proof results in either a verification error or, if the verification succeeds, a safe code transformation [52]. Despite several advantages provided by proof carrying code, it has a few major drawbacks which limit its applicability: the responsibility for proof generation lies with the code producer, proof of safety has to be generated for all the remote hosts after knowing their safety specification, and getting consumer specifications from all the remote hosts is very difficult, which increases the difficulty for the code producer to generate the desired safety proof without consumer specifications [9, 39, 40].

2.2.2 Malicious host threat

The protection of a mobile agent in an open environment from a possible malicious host is possibly the biggest security challenge introduced by the mobile agent paradigm. An agent executing on a remote host is vulnerable to eavesdropping and corruption, since the host exercises complete control over the agent and has unrestricted access to its code and data. A number of approaches have been devised in the literature to protect a mobile agent from a malicious host. Sander and Tschudin [69] classified the malicious host problem into two broad categories: attack prevention and attack detection. An attack prevention approach attempts to prevent attacks before they happen. Prevention mechanisms attempt to make it impossible to modify or access a mobile agent in a meaningful way. A hardware based solution [40] is a good example of a prevention mechanism. In contrast, an attack detection approach aims to detect attacks after they happen. It includes tracing the identity of the malicious host as well as proving the malicious act. The execution trace based mechanism [76], and the KAG protocol [40] are good examples of detection mechanisms. As prevention

of attacks is difficult to achieve, most approaches are aimed at detecting the attack rather then preventing it.

A malicious host can mount several types of attack on a mobile agent, such as altering the current state of a mobile agent, substituting the agent code in order to collect signed data which are then attached to the original agent code, and truncation of better offers provided by other hosts. Out of all attacks, a truncation attack [55] is definitely the most difficult to prevent or detect, in the case of a free roaming mobile agent. Roth [68] pointed out that most of the mechanisms proposed in the literature are vulnerable to truncation attack and interleaving attack. In general terms, a truncation attack [44] is the truncation of a host offer by one or more colluding malicious hosts. A truncation attack can be easily detected for a fixed itinerary mobile agent (i.e., the list of hosts to be visited is fixed and order of visits is also fixed at agent departure from the origin host) and a partially fixed itinerary mobile agent (i.e., the list of hosts to be visited is fixed but the order of visits to those host is not fixed). However, in both cases the malicious hosts who have truncated the hosts' offers are very difficult to detect. Although a truncation attack mounted by a single host can be detected easily by establishing a chaining relation in a free roaming mobile agent, if two or more hosts collude and truncate the chain of offers between them, it is very difficult to detect all the hosts which have been truncated and the malicious hosts who have truncated the hosts' offers from the chain of offers.

A brief overview of existing approaches which address the malicious host problem is provided below:

i. Hardware-based solution: While an agent is executing at a host, it is possible that the host could examine the agent code. Thus, hardware support [9, 40] is recognized as the only tractable solution, since no single software solution proposed so far addresses every possible attack. Tamper-proof hardware on the host can be used to protect the agent code. One example of a hardware based solution is the Tamper-proof Environment [40], which is a regular computer with a specialized OS, manufactured only by authorized well-known parties. For this technique to work all the hosts must trust the manufacturer of the hardware modules, which is a strong

assumption. Also, installing tamper-proof hardware on each host is an expensive prospect.

ii. Trust based solution: The simplest way to prevent malicious hosts from tampering with agents is to allow the agent to visit only trusted hosts. Thus, this mechanism ignores system security by assuming that every entity in the system can be trusted and would not mount any hostile attack on the mobile agent. This mechanism is ideally suited to an environment where all entities trust each other (for example, a single company using a mobile agent). Due to the lack of trust in e-Commerce, this mechanism cannot be an effective method [9]. Also, the advantages of using a mobile agent will not be fully exploited by limiting the mobile agent to only visiting selected trusted hosts.

iii. Obfuscation: This mechanism relies on the idea of preventing the attack on the mobile agent by scrambling the code in such a way that no one gains a complete understanding of its functions (i.e., specification and data) or to modify the resulting code and data without detection [9]. The strength of the scrambling completely depends on the obfuscation algorithms. Time-limited black box [37] security is an example of the obfuscation method. It uses an obfuscation algorithm to convert the agent code into a new code which is hard to analyze. This mechanism provides security for a certain interval of time and assumes that for a fixed interval of time, the code and data of the agent specification cannot be read and modified without detection. Its security is based on the assumption that analysis of the code will take an amount of time greater than the execution time of the mobile agent at a particular host [40]. The major drawback of this mechanism is that it is not possible to quantify the amount of time that the protection will be in place.

iv. Computing with encrypting functions: The basic idea behind this mechanism is to determine a method which allows the agent to compute a function in encrypted format without revealing the original function to the host. It eliminates the need of the mobile agent to rely on clear text data, code and state to do its task [69]. The origin host encrypts a function, which is then executed by the remote host without realizing what the function is. So far, this solution can only be used with a polynomial function. Presently, there is no generally applicable cryptographic theory

for computing arbitrary functions in this manner [40, 9]. Another problem is that the agent is not protected against modification attacks.

v. Environmental key generation: This technique is based on public key cryptography. It involves the encryption of the mobile agent code, which is only decrypted when an environmental condition is matched. Upon encountering an environmental condition, a key is generated by the agent, which is used to unlock some executable code [40]. The environmental condition may be hidden using a one way hash function or public key encryption; hence, the remote host cannot uncover the environment condition message or response action by directly reading the agent's code [65]. The major drawback of this mechanism is that once the code is decrypted, a host can do anything with the code; it may modify the code or it may only read the code but not execute it [39]. The other problem with this mechanism is the dynamic generation of the code, which could be considered as unsafe to execute and therefore may not get sufficient privileges to execute and complete its operation.

vi. Execution tracing: This mechanism aims at detecting unauthorized modification of code during execution of the mobile agent at each host. It is a cryptographic trace based approach which requires each host to create a non-repudiable trace of the agent's execution and submit the cryptographic hash of the trace as a trace summary. After the mobile agent returns to the origin host, the origin host examines all results and if it finds any results suspicious, the trace and trace summary for that result are obtained and verified, and a malicious host is identified [76]. The major drawbacks of this technique are the size and number of logs to be retained, and its detection process, which is triggered occasionally, based on suspicious results [40].

vii. Master-slave framework: In this mechanism, the slave mobile agent visits all possible untrusted hosts on the itinerary list and the master agent moves along on all trusted hosts and keeps an eye on the slave agent. The master agent observes and assigns the task and data to the slave agent. The major drawback of this approach is that the master agent needs trusted hosts to operate on. In extreme cases, this means that the master agent must remain at the agent's home machine effectively reducing the mobile agent architecture to client-server architecture.

viii. Mutually cooperating agents: This mechanism involves the agent's itinerary being recorded and tracked by another cooperating agent, and vice versa, in a mutually supportive arrangement. In this mechanism, both agents cooperate and exchange the information about the current host, last host and the next host through an authenticated channel. An agent never visits a platform which has been visited by its peer agent. By dividing up the operations of the application between two agents, certain malicious behaviour of a host platform can be detected. This mechanism is based on the assumption that there are only a few host agents that are malicious and that both cooperating agents are not likely to visit two colluding hosts at the same time [67]. The major drawback of this mechanism is that if any agent is killed, it is difficult to determine which of the two platforms is responsible.

ix. Itinerary recording with replication and voting: In this mechanism, multiple copies of an agent are sent out to perform a task, instead of a single copy of an agent [70]. At each stage, the host receives multiple copies of an agent. It checks to ensure that the arriving agents are intact, makes replicas of only valid agents and forwards them to the next host. The major drawback with this technique is its assumption that there are only a few malicious hosts in the system and that they can corrupt only a few copies. The other obvious drawback is the additional resources consumed by replicated agents.

x. Software based solution: This mechanism introduces a software based solution called Remote Distributed Scheme (RDS) to protect the computation of mobile agents. The RDS [21] scheme is based on two major ideas: first, launching a set of replicated agents instead of one agent to perform the computation remotely, and second, a secret sharing scheme where each mobile agent holds only one share of each transaction and reveals as little information as possible to the host platform. In this scheme, each agent relocates itself on a different host and based on the message it receives from the host, its algorithm and its internal settings, the agent chooses one share from its set of shares and sends it to the host platform [21]. When an agent has enough shares of the same transaction, it uniquely constructs the transaction, performs its computation, and broadcasts its response to all the agents. The major drawbacks of this scheme are increased communication overhead due to replicating

the agents, relocation of agents on the trusted hosts, the performance dependency on the relocation of all agents, and the collection of shares of a transaction.

xi. Watermarking and finger printing: This technique, proposed by Esparza, et al. [27], uses a software watermarking technique to detect a manipulation attack performed during the mobile agent code execution. In this approach, fixed watermarks are embedded in the mobile agent and this mark is transferred to the agent result during execution [27]. These marks are verified at the origin host and if the marks differ from the expected marks then the host is considered to be malicious. This technique not only detects manipulation attacks, but also proves the malicious behaviour of a host.

    Later on, the Esparza, et al. [28] extended their work and introduced the concept of embedding watermarks that change dynamically during execution, instead of fixed watermarks, as in the previous technique. In this mechanism, a data container is created at each host where the watermark is transferred. When the agent returns to the origin host, the origin host verifies the execution integrity by applying a set of integrity rules to the containers [28]. The mechanism also shows how mobile agent watermarking can be used to punish malicious hosts by using a trusted third-party and host revocation authority.

xii. Append only container: This technique, proposed by Karnik and Tripathi [42, 43] is aimed at data integrity, by which new data can be added to the data container but any subsequent modification of the data contained therein can be detected only by the origin host. It is achieved by appending a cryptographic checksum that binds the current data to the previous data. The major drawback of this scheme is the lack of binding between the dynamic data and the static data of the agent, which makes it vulnerable to an interleaving attack [55]. Furthermore, this technique is also vulnerable to a truncation attack, where an untrusted host i_n cooperating with a second untrusted host i_j can easily truncate the chain of data at k for each $j \leq k < n$ without being detected [55].

xiii. Partial result authentication code: In [78], Yee proposed the partial result authentication code (PRAC). He introduced the idea of "forward integrity," which implies that all the results collected from hosts visited prior to a malicious host can

be trusted. In this technique, the result of the agent's computation at each host is encapsulated using a Message Authentication Code (MAC) and sent back to the originator. The result at each host and the resultant MAC are called the Partial Result Authentication Code. In PRAC, the agent and the host maintain a list of the secret keys or by using a one way function, incrementally generate secret keys for all the hosts to be visited [78]. Each key is used only once and is assumed to be erased by the server after encapsulating the offers collected from the host. However, only the originator can verify the results, since there are no other copies of the secret key. This technique is based on the assumption that the host will destroy the secret key after finishing the computation, but if any malicious platform retains a copy of the original keys of an agent and if an agent revisits the host, the previous partial result entry could be modified without the possibility of detection [55].

xiv. KAG: In [44], Karjoth, et al. reformulated and improved Yee's proposal. In particular, they introduced a family of protocols using certain cryptographic notions which aim at preserving the integrity and confidentiality of data acquired by free-roaming mobile agents. The protocols are based on a chaining relation: the offer from each host is cryptographically related with the offer from the previous host and the identity of the next host [40]. This is done to preserve the integrity of data collected by a mobile agent at every host. Roth [66, 68] pointed out that the proposed protocol is vulnerable to an interleaving attack. This attack can be mounted because of a lack of verifiable binding between the data collected and the static piece of the mobile agent's code.

xv. Abstract protocol: Extending the work of Yee [78] and Karjoth, et al. [44], Maggi and Sisto [55] proposed a generic protocol that could be configured according to security requirements and application. Their work addresses the problem of protecting the data carried by mobile agents from possible attacks of a malicious host. It specifically takes into account the two major issues related to the malicious host threat (i.e., data truncation and a binding of the static code to the collected data) and defines a generic protocol that does not suffer from all the previous limitations of the protocols proposed in [78, 44]. In the protocol, it is shown that limited truncation resilience can be achieved by forcing the agent to securely store the

addresses of the next hosts to be visited. They also showed how the static piece of an agent can be bound with the collected data (dynamic data) using cryptographic operations. In their protocol, they showed how data confidentiality, data authenticity, non-repudiation, forward data integrity, freedom from interleaving attack, and limited truncation resilience could be achieved by using cryptographic operations. In this protocol, data confidentiality is achieved by encrypting data offer d_n using the originator's public key; the forward integrity property is achieved by establishing a chaining relation between the previously collected signed offer, the current offer d_n, and the identity of the next host (i_{n+1}); data authenticity and non-repudiation are achieved by digitally signing the data d_n together with the static piece of data Π_0 (Π_0 is the digitally signed static code provided by the origin host). This protocol avoids interleaving attacks by asking the host to sign data d_n with the agent static piece of data Π_0. This ensures that the mobile agent having static signed code Π_0 has really been executed on i_n, producing offer d_n. This protocol achieves a weak form of truncation resilience by forcing the agent to securely store the addresses of the hosts with the offer. In practice, the agent gathers two pieces of data on each visited host i_n: a set P_n of hosts to be visited (possibly empty, if the visited host does not want to add new hosts to the agent's itinerary) and the actual data d_n (possibly a dummy value, if host i_n does not want to or is not able to provide data).

2.3 Privacy Review

This section presents a review of privacy preference languages, such as P3P, APPEL, XPref, and OWL, and brief overview of related work in privacy negotiation.

2.3.1 Platform for Privacy Preference (P3P)

The Platform for Privacy Preferences (P3P) [16], a W3C standard, enables Web sites to describe their privacy practices in a standardized machine-readable format, indicating which data are to be collected, for what purposes, with whom it will be shared, and for how long it will be retained. The presence of P3P policies enables users to gain more control over the use of their personal information while browsing the Web. It allows automatic retrieval of privacy policies by the P3P agent, matching of the privacy practices of Web site with user

preferences, and notification to the user if there is any mismatch between the privacy practices and user preferences [16]. P3P uses XML format for encoding privacy practices. The P3P 1.1 Specification [16] provides detailed definitions for all of the XML elements that comprise the P3P vocabulary. Table 2.1 provides an overview of some of these elements.

ENTITY	Identifies the legal entity making representation of privacy practices contained in the privacy policy, for example, contact information for the business, organization, or person who owns the Web site
ACCESS	Describes whether the Web site allows individuals to access their personal information and see what type of personal data a Web site keeps about them in its databases
DISPUTES	Describes the dispute resolution procedure to be followed if any dispute arises about the entity's privacy practices. Also includes a REMEDIES sub-element, which specifies the possible remedies if a policy breach occurs
DATA	Describes the kind of data collected by the Web sites
PURPOSE	Describes how collected data is to be used (12 types of purposes are specified), and whether individuals can opt-in or opt-out of any of these uses
RECIPIENT	Describes the legal entity to which data may be distributed. It specifies whether and under what conditions data may be shared and whether there is an opt-in or opt-out
RETENTION	Describes how long the personal data about any entity may be kept in their database
CONSEQUENCE	Describes a human-readable explanation of the Web site's data practices. It explain why the suggested practice may be valuable in a particular instance even if the user would not normally allow the practice

Table 2.1: P3P Data Elements

Among all the elements mentioned, we specifically consider the elements purpose, data, retention, and recipient, which are necessary for negotiation. All other elements like access or disputes can be negotiated but this may lead to high overhead for any company which wishes to switch its dispute procedure or change its access conditions [34]. Therefore, it is very hard to achieve negotiation on these elements. The other two elements (entity and consequence) provide information about the policy and the Web site. They are not considered in the present negotiation model. Therefore, to get any service from a Web site it is assumed that user has agreed to all other elements except the remaining four.

P3P has predefined choices for the purpose, recipient and retention elements. The purpose element has twelve choices, namely, current, admin, develop, tailoring, pseudo-analysis, pseudo-decision, individual-analysis, individual-decision, contact, historical, telemarketing and other-purpose [16]. All elements specify different purposes for which personal information will be collected. For example, current purpose means the data will be collected for completion of the current transaction, and admin means the personal information will be collected for administrative purpose like Web site and system administration. A purpose element can have more than one choice; for example, a Web site can specify "personal information will be collected for current and administration purpose." The recipient element has six choices, namely, ours, delivery, same, other-recipient, unrelated, and public. A recipient element can have more than one recipient for the personal information it collects; for example, a Web site can specify that personal information collected may be distributed to companies having privacy policies which are the same as the current Web site and to delivery companies which are helping to satisfy the current request.

The retention element has five choices, namely, no-retention, stated-purpose, legal-requirement, business practices, and indefinitely. A more detailed explanation of P3P is available in [16]. A P3P policy can have one or more purpose or recipient choices, but it cannot have more than one retention choice. For example, a simple P3P policy [16] taken from the P3P standard is shown in Table 2.2.

```
1  <POLICIES xmlns="http://www.w3.org/2002/01/P3Pv1">
2  <POLICY discuri="http:// xyz.com/privacy.html" name="medicalpolicy">
3  <ENTITY>
4   <DATA-GROUP>
5    <DATA
6     ref="#business.contact-info.online.email">abc@xyz.com
7    </DATA>
8    <DATA
9     ref="#business.contact-info.online.uri">http://xyz.com /
10   </DATA>
11   <DATA ref="#business.name">Web Privacy with P3P</DATA>
12  </DATA-GROUP>
13  </ENTITY>
14  <ACCESS><nonident/></ACCESS>
15  <STATEMENT>
16   <CONSEQUENCE>We retain data for 3 months</CONSEQUENCE>
17   <PURPOSE><admin/><current/><develop/></PURPOSE>
18   <RECIPIENT><ours/></RECIPIENT>
19   <RETENTION><indefinitely/></RETENTION>
20   <DATA-GROUP>
21    <DATA ref="#user.name"/>
22    <DATA ref="#user.telecom.telephone"/>
22    <DATA ref="#user.telecom.cell"/>
23    <DATA ref="#user.creditcard"/>
24    <DATA ref="#user.bloodgroup"/>
25   </DATA-GROUP>
26   <DATA-GROUP>
27    <DATA ref="#dynamic.clickstream"/>
28    <DATA ref="#dynamic.cookie"/>
29   </DATA-GROUP>
30  </STATEMENT>
31 </POLICY>
32 </POLICIES>
```

Table 2.2: Simple P3P Policy

P3P defines a data schema also known as the P3P base data schema [16]. It includes four data element sets, namely, user, thirdparty, business, and dynamic. The user, thirdparty and business sets specify the data provided by a user, and the dynamic set includes data that are dynamically created, such as cookies.

In summary, there are several advantages of making a Web site P3P enabled [16, 35], and they are as follows:

- A P3P policy is retrieved automatically and interpreted easily by user agents; therefore, there is no need to read the whole policy.
- A P3P policy is composed of simple multiple choice questions; therefore, it is easy to understand.
- P3P policies often include information about aspects of privacy that sites have chosen not to cover in their human-readable policies. Therefore, P3P encourages companies to think carefully about their privacy policies and do a systematic assessment of their practices.

However, two major limitations of P3P are that no support is given within the specification for negotiation of privacy preferences between a user and a Web site, and there is no policy enforcement. In other words, P3P does not provide a technical mechanism to enforce Web sites to act according to their policies. In this study, we will address the first limitation of P3P and propose privacy negotiation using a mobile agent.

2.3.2 A P3P Preference Exchange Language (APPEL)

The designers of P3P also designed a preference language called APPEL [19] to allow users to express their privacy preferences. These user preferences are interpreted automatically by the P3P agent and a comparison is made with the Web site's P3P privacy practices while browsing the Web site. With APPEL [19], a user can specify his preferences in a set of preference-rules, also called a ruleset. When a user browses a web page, the P3P agent retrieves the related P3P policy from the Web site and invokes an APPEL rule evaluator to evaluate the APPEL ruleset with regard to the related P3P policy. If the Web site privacy practices are compatible with the user preference rule, the appropriate rule fires and the

corresponding behaviour defined in the rule is initiated. In APPEL [19], all rules in a ruleset are evaluated strictly in order and this can greatly influence the behaviour of a P3P agent. Unfortunately, as Agrawal, et al. [1], have shown, APPEL has serious design problems which cannot be corrected unless it is re-designed. They showed that in APPEL, users can only directly specify what is unacceptable in a policy, but not what is acceptable. This is because a policy is allowed to contain multiple statements and APPEL follows a strict rule ordering: the rule evaluation is stopped once the first rule matches with the P3P policy. Furthermore, in APPEL, simple preferences are hard to express, and writing user preferences is error prone. Thus, Agrawal, et al., proposed an alternative to APPEL called XPref.

2.3.3 XPath Based Preference Language (XPref)

XPref [1] is an XPath based language. It uses a strict subset of XPath 1.0 for almost all of its functionality. Additionally, it makes use of quantified expressions from the XPath 2.0 Working Draft [1]. The main design goal of XPref was to remove the deficiencies of APPEL while retaining as much of the APPEL syntax and semantics as possible. XPref subsumes the full functionality of APPEL while avoiding its pitfalls. Agrawal, et al., showed that XPref serves quite well as a preference language and it is as expressive as APPEL.

2.3.4 Web Ontology Language (OWL)

As APPEL is an XML-based user preferences language, it is difficult to use APPEL to describe various complex logic connective relations between different types of P3P elements [41]. Therefore the knowledge domain of user online privacy preferences cannot be represented in APPEL effectively. XPref [41] is also an XML based language and does not have any standardized semantics. Therefore, it still cannot describe relations between data and logic rules existing in the knowledge domain of user online privacy. In [41, 34], the authors showed that OWL can address the above issues and serve quite well to express user privacy preferences. It allows the expression of logical relations between the P3P data elements using properties and the expression of cardinality on properties [34, 36]. Therefore, in this thesis, the user's preferences are represented in OWL [57].

As an example of the above, "user's first name is a subclass of user's name" and "user's name is releasable under condition A" together allow a system conforming to OWL to conclude that "user's first name is releasable under condition A." The same information, if stated in APPEL [41], does not allow the system to assert the third statement. The software developer has to write an extra predefined program to implement similar semantics [36]. In addition, ontological support for multi-language platforms is needed for international Web use [34, 41]. For example, in English and French, different terms may be used for the same P3P term. This would be very difficult to express in APPEL and XPref, but in OWL it can be expressed by using equivalentClass, which would state that a term in French user preferences is the same as a term in user preferences in English.

2.3.5 Online Privacy Agent (OPA)

In 1999, Mayer [55] proposed the Online Privacy Agent. It is based on the idea of specifying constraints on given sets of information. These constraints define the conditions under which a particular set of information can be accessed. For example, during a transaction, a Web site issues a request to gain access to a particular set of information. The owner of the information checks this request and tries to verify whether the requestor satisfies all the constraints defined for the requested set of information. If the constraints are satisfied, the requestor is allowed to access the information. If the constraints are not satisfied, the owner can reject the request or start negotiations.

i. Terminology

OPA has four major elements around which the whole negotiation process revolves [56]. These are described as follows:

- Information: This represents the finite set of personal information $P = (d_1, d_2, d_3, ..., d_n)$ where $d_i (1 \leq i \leq n)$ is the data element that represents information, such as the person's age or name.
- Rule: A rule r specifies the circumstances under which a set of information can be accessed. It is represented as a pair:

$$r = (D_r, C_r), \quad D_r \subseteq P$$

where D_r is a set of information and C_r is a set of constraints defined on D_r. $C_r = \{c_1, c_2, c_3, ...c_n\}$ represents a particular condition and each constraint in C_r must be met to get access to D_r.

- Constraint: The constraints of a rule specify the conditions that must be satisfied to get access to the information. For example, a family may set up rules to control access to its cars. The rule would be as follows:

$$r_1 = (\{convertible\}, \{(driver = parent), (season = summer)\})$$

Here, D_r = $\{convertible\}$ and access to $\{convertible\}$ is only granted when the constraints C_r = $\{(driver = parent), (season = summer)\}$ are satisfied. In this case, it means that the convertible can only be driven by the parents when it is summer.

- Facts: The facts f are associated with a request for information. They are defined by a pair (D_f, V_f), where $D_f \neq \varnothing$ represents the data elements requested and the set $V_f = \{p_1, p_2, ..., p_l\}$ represents the conditions under which the elements of D_f are requested. The facts formally describe a request for a set of data-elements and the conditions under which this request is made. The owner of the requested data-elements can evaluate the facts and match them against his or her rules. If the facts satisfy a rule, the owner may grant the requestor access to the requested information. An example of facts could be $f_1 = (\{convertible\}, \{(driver = mother), (season = winter)\})$, which can be checked against the rule r_1 as shown above. The set $F = \{f_1, f_2, ..., f_k\}$ denotes all possible combinations of facts of a particular domain.
- Ruleset representation: Rules define the conditions under which a set of information can be accessed. In order to express such constraints for various sets of information, more than one rule needs to be defined. A ruleset is a set of rules, represented as $R = \{r_1, r_2, ..., r_k\}$. In OPA, rulesets are represented by a tree format. For example, assume person X has three rules to control access to three different sets of

information: $r_1 = (D_{r_1}, \{c_1, c_2\})$, $r_2 = (D_{r_2}, \{c_3, c_4\})$, and $r_3 = (D_{r_3}, \{c_3, c_5\})$. These rules can be grouped into a ruleset as $R_x = \{r_1, r_2, r_3\}$. The tree representing these rules is shown in Figure 2.1. Each of the three rules is represented by a path from the root R_x to one of the leaves of the tree. The leaves of the tree represent the data set D_{ri}, with *i = 1, 2, 3* and the constraints lie between the root and the leaf.

ii. Rule Evaluation

In OPA [56], rule evaluation is done by matching the facts against the rules. When a match is found, access to the requested information can be granted. In order to get access to the information, a fact needs to satisfy two requirements. First, it must satisfy all constraints in the rule and second, the requested information in the fact must be a subset of the information in the rule [56]. This means that the requestor/facts can only ask for data elements that are part of the rule's set of information. In short, facts match a rule when all of the rule's constraints are satisfied and the requested data elements (as specified in the facts) are covered by the rule. This has certain implications. First of all, facts can contain more information than necessary to satisfy the constraints. Second, the rule can cover more data elements than those requested in the facts [56].

iii. Counter-Offer Generation in OPA

Mayer [56] introduced a distance-based approach to generate a reasonable counter offer. The term "reasonable" in this context means that a counter-offer produced is semantically related to the rejected request. A distance function is used to measure the distance between a rule and the facts. There are three distance functions that are defined in OPA [56]: first, a function which measures to what degree the fact satisfies a rule constraint; second, a function that measures how well the two sets of information overlap; and finally, a function which maps the distance between a rule and a fact. The rule with the minimum distance is considered to be the closest rule and is used to produce a counter-offer. After finding the smallest distance between the rules and facts, the closest rule is determined. OPA [35] uses a depth first search to find the closest rule.

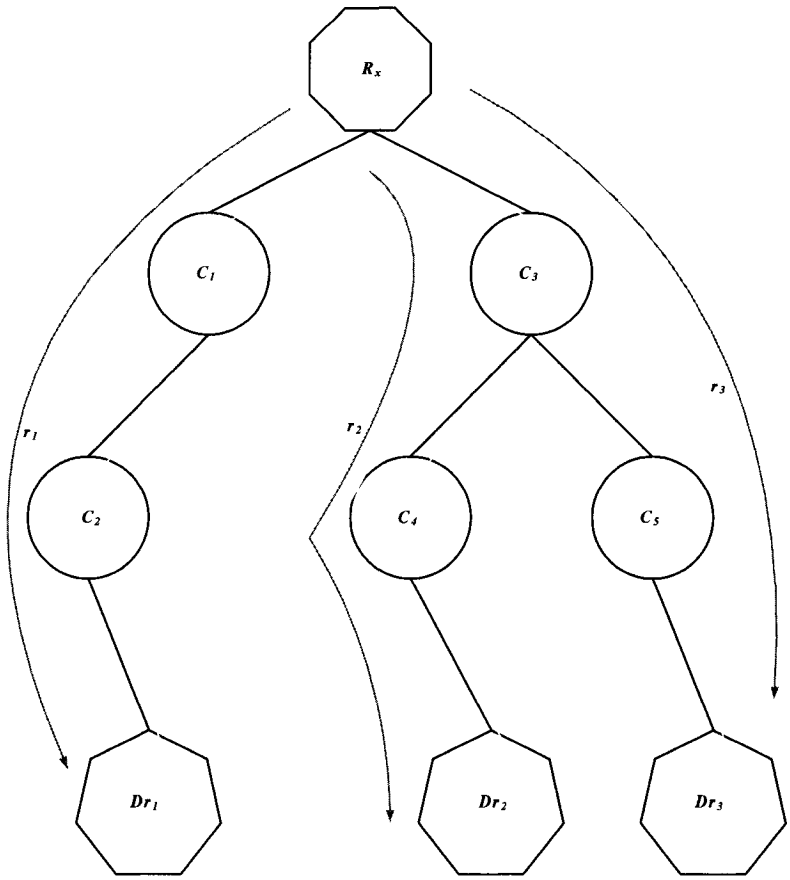

Figure 2.1: Tree representation of the user preferences

2.3.6 Arbitrary Agent

In [34], He proposed online support for negotiation of privacy contracts through a software agent. This approach is based on the idea of OPA and introduces a usage based concept to conduct privacy negotiation using a software agent. This architecture addresses the two major limitations of OPA [34]. The first is that the facts can contain more conditions than necessary to satisfy the constraints; He showed that this is not suitable for privacy information matching. He uses an example, in which there is a rule for user address (r_1= ({address}, {(purpose=current), (recipient=ours), retention=6 months)})) and a fact (f_1= ({address}, {(purpose=current), (recipient=ours), retention=6 months) (other-purpose = admin)})). According to the OPA evaluation, f_1 matches r_1, and the user's address is revealed, but this actually violates the user's preference, as the user preference only specifies

that the address is to be collected when the purpose is current. The second limitation of the OPA addressed by the arbitrary agent architecture is that OPA organizes a rule by its release conditions and if a request does not match a rule which contains the requested data, and then a different set of data which has the same condition of the request is released.

He proposed the arrangement of the user's preferences into a tree based structure, to relate the user's preference rules to a data hierarchy. He took the four disjoint data sets of P3P and constructed a user preference tree for each data set. A tree may be empty if no part of the data set is allowed to be released under any condition. The maximum number of trees possible is equal to the number of disjoint data sets, four. This architecture also introduces a usage-based concept consisting of a triple of P3P defined elements upon which both users and host can usefully negotiate.

2.4 Related Work

In Ref. [72], Smith and Shao proposed a new approach to improve the information filtering of user preferences in an E-commerce application. This approach specifically addresses two major problems of exploitation of the user preferences and privacy issues. Firstly, an e-business system may take advantage of a user if that system learns about the user's constraints on a purchase (type of item, spending range, etc.). Secondly, when a user's preferences are sent in full to an e-business system, the user actually releases a substantial amount of personal information to the e-business (i.e., information about their likes and dislikes). And if an e-business system is dishonest, it can exploit the knowledge contained in this full preference statement. This could result in the user being returned a set of results that are not optimal for them, instead being optimal to the e-business.

In order to address these concerns, the authors proposed a new privacy enhancing technique in which information is released gradually when necessary to the e-business system. Thus, the onus is switched to the user side of interaction rather than the e-business side. This maximizes the user's privacy by minimizing the preference knowledge transmitted, while preventing exploitation of the knowledge contained in the full preference statement. This approach is shown in Figure below:

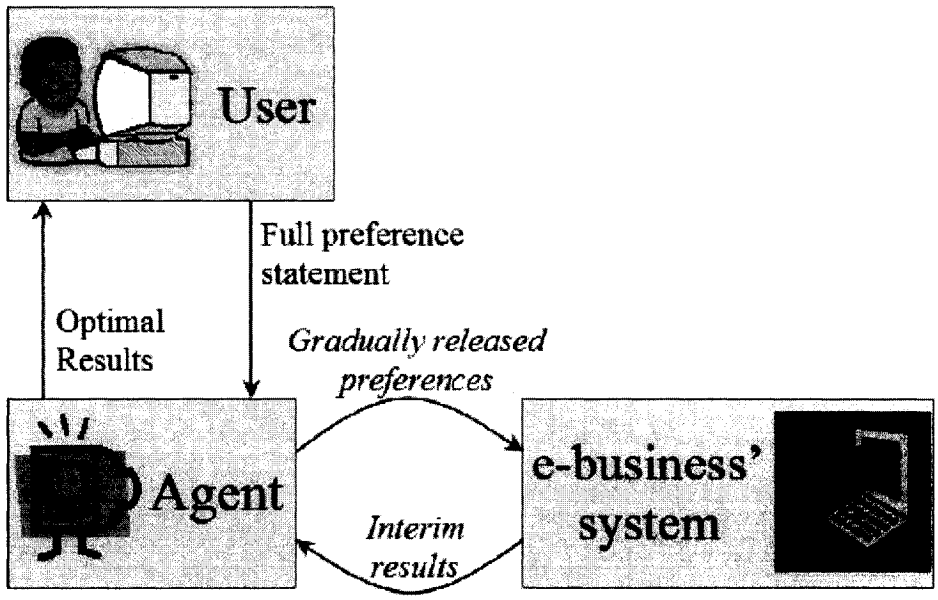

Figure 2.2: PET Approach

In this technique, a user agent acts on behalf of the user and is used to select the most preferred items from the untrusted remote system. The user agent collects the user's preferences (by some means), analyses these preferences and separates them into elements. It then gradually releases these elements to the remote e-business DBMS, analyzing the results obtained, until the results were satisfactory or it ran out of preferences. The user agent then presents the results to the user. Transferring the job of selecting the most preferred items from the untrusted e-business' system to a trusted agent helps in maximizing the privacy and preventing the exploitation of preference knowledge.

Further, the user agent will have complete knowledge of the user preference. The agent uses this information to release them gradually to the remote e-business system, thereby preserving privacy to the maximum extent. First, the agent generates an initial query and sends it to the remote host. This query effectively searches for the user's ideal item. The results of this query are returned to the agent. If enough and optimal results were returned, then the agent applies some logic to the result set to give them another level of preferred ordering. The agent would also discard some of the results if the user specified a limit to the amount of results they wanted - it would remove the least preferred of the results. However,

if no (or not enough) results were returned, then the agent would relax the query a step and resend this to the e-business' system. This query will release little more information (as compared to previous query) about the user likes and dislikes. This process goes on repeatedly until enough and optimal results were returned, or the agent ran out of preferences.

Overall, the author addressed the problem of exploitation of the user preferences very well and proposed a novel PET which allows an agent (on behalf of the user) to release the user preference gradually whenever necessary, to counter the problem of exploitation of user preference in E-commerce. This ensures that the results received are the "best" results for the user whilst retaining as much privacy as possible.

This architecture pretty much addresses the same problem which our architecture does but in a different way. This technique follows a client server architecture, where a user agent acts on the user behalf, does negotiation with a remote e-business system and returns the final result of the negotiation to the user. The main advantage is the gradual release of the user information to the e-business system thereby retaining as much privacy as possible and also getting the best results for a user.

In contrast, our architecture addresses the first problem, exploitation of user preferences, by sending a mobile agent to the different remote host, where it negotiates with each remote host and collects the result of negotiation. As mobile agent visits n number of hosts, it forces or makes the remote host to think before giving the negotiation offer. Even if at least one host is honest and did not exploit the user preferences and generates an offer which is optimal for both a user and the remote host will probably eliminate all other offers, while evaluation of the offers at the origin host, supplied by the dishonest hosts (which are being optimal to the e-business rather than to a user).

Further, in our architecture, the mobile agent does carry all the user preferences with it. However, those are inserted into a tree using a label value which may allow a remote host to make a guess about the importance of the user preferences by guessing their value but if the

labels are assigned randomly to the values it will not give the complete knowledge about the preferences to any of the remote hosts.

The architecture described in [72] follows a client server architecture which may involve an exchange of 'N' message transmissions between the agent and the e-business system. Generating an ideal query which would allow an agent to retrieve the best results by releasing minimum information can be very tough in case of privacy negotiation because the user' perception about its information varies with the changes in the transaction. Also, this may vary from one user to another. This can be an ideal choice for a transaction involving price negotiation or any other simple information filtering application. However, transferring a complete P3P privacy policy may be a tedious task, and most importantly, it will shift the evaluation of the privacy policy to the user side which will put extra overhead to the user rather than host side as in our architecture.

Thus, in terms of user overhead, our architecture reduces the over load from the client slide and shifts the negotiation process completely towards the server side while in the architecture proposed by Smith and Shao [72], the over head of evaluating the preferences is shifted to the user agent (which acts on behalf of the user).

The architecture proposed by Smith and Shao also allows a resource constrained device to do negotiation by sending a simple request. The user agent will do all the processing for the user and returns an appropriate result. In our architecture, user device has to do some processing over its system like evaluation of offers and signing and verification. Thus, for a scenario like a scenario where transfer of messages between the client and server is low over the network, the approach proposed by Smith and Shao can be deployed.

2.5 Summary

In this chapter, we have presented a review of the mobile agent paradigm, its advantages, and the existing approaches addressing the two major security concerns associated with the mobile agent paradigm. This chapter has also presented a review of related work on privacy negotiation. All of the above work on privacy negotiation architecture follows the client

server architecture. This architecture takes the idea of the previous work on the privacy negotiation by Mayer [56] and He [34] and proposes a mobile agent based privacy negotiation architecture.

Chapter 3: Privacy Negotiation Architecture

This chapter presents the mobile agent based privacy negotiation architecture. First, the basic terminology required to understand the architecture is described, then, a description of how user-preferences are collected and represented is presented. After that, the entities involved in the architecture at the origin host and the remote host are described, along with their interaction throughout the negotiation process, including the interaction of the entities before, during and after the negotiation. Finally, the last section provides a detailed description of the negotiation process. This architecture takes the idea of the previous work on the privacy negotiation by Mayer [56] and He [34] and proposes a mobile agent based privacy negotiation architecture.

The negotiation process covers four processing phases: the pre-processing phase, the negotiation phase, the post-processing phase, and the agreement confirmation phase. These four processing phases take place in two rounds of the mobile agent. In the first round, the mobile agent visits all the hosts in its itinerary list and does the negotiation with each host. If the negotiation is achieved, it collects the signed agent trace offer and travels to another host in its itinerary list. As the mobile agent follows a partially fixed itinerary, it visits all the hosts in its itinerary list, but the order of visits may vary depending upon the mobile agent's decision ability. When mobile agent returns to the origin host, the optimal offer is determined and mobile agent is dispatched again to the selected host to get the final confirmation of negotiation and to collect the privacy data offer. The terms "negotiation achieved" and "negotiation successful" have different meanings in this study. Negotiation is said to be achieved, if during the first round, the mobile agent and the host agree on an agent trace offer. Negotiation is said to be successful if during the second round a host confirms the negotiation over the agent trace offer and provides the signed privacy data offer.

3.1 Negotiation Terminology

The basic terminology and definitions used in this chapter are described below.

i. Purpose: The purpose element indicates the ways in which the personal information collected by the mobile agent can be used by the remote host. It is represented as:

$P = \{p_1, p_2, p_3, p_4, \ldots\ldots\ldots, p_n\}$ where $p_i = (1 \leq i \leq n)$ is the predefined values specified in the P3P standard for the <PURPOSE> element. There are twelve predefined values in P3P for the purpose element and the purpose element can have more than one value for the data collected, such as <current, administration>.

ii. Recipient: The recipient element indicates the legal entities or organizations with which the collected personal information can be shared or to whom it can be distributed. It is represented as: $R = \{r_1, r_2, r_3, r_4, \ldots\ldots\ldots, r_n\}$ where $r_i = (1 \leq i \leq n)$ is the pre-defined values specified in the P3P standard for the <RECIPIENT> element. The P3P standard has six predefined values for the recipient element, and the recipient element can have more than one value for the data to be collected, such as <ours, delivery, same>.

iii. Retention: The retention element indicates the length of time that the collected personal information can be retained by the host. It is represented as a pair: Retention = (value, length) where "value" is the pre-defined value of the P3P element <RETENTION>. The P3P standard has five pre-defined values for the retention element, and the retention element can have only one value, such as no-retention, stated-purpose, indefinitely. The "length" specifies the actual retention length, such as 2-weeks, 3-months, or 1-year. The length value lies in the $(0 \leq i \leq \infty)$ range. If the value element = "no-retention," then length value = 0, and if the value element = "indefinitely," then length value $= \infty$.

iv. Personal information: Personal information represents a finite set of user information $p = (d_1, d_2, d_3, d_4, \ldots\ldots\ldots, d_n)$ that can be released to the host if the negotiation between a remote host and a user is *successful*. Here, d_i $(1 \leq i \leq n)$ is the data element that represents the user's personal information, such as user.name, user.age, user.telecom and dynamic.cookie.

v. Negotiation achieved: The negotiation between a host and a mobile agent is said to be achieved if certain conditions are met for the requested personal information. These conditions are specified by the user during preference setting and may vary for a data element according to the transaction type and the user privacy sensitivity level. The privacy sensitivity level defines how strict the user is about his or her personal information and under what conditions this personal information can be released to a

host. For example, user "A" is very sensitive about his cell phone number and does not want to reveal the number to every host. On the other hand user "B" is less sensitive and may release the number to any host. In the same way, the conditions for a data element may vary with the change in the transaction type. For example, a user may release his blood group information for a medical transaction, but the same user might not release the blood group information for a banking transaction (such as for insurance purposes). During negotiation, if any data element requested by the remote host is not included in a set of user preferences, the request for that element is rejected by the mobile agent, assuming the implicit rejection of the user concerning that requested data element.

vi. User preference: A user preference relates personal information to a set of conditions. The personal information can be released to the host if at least one of the conditions associated with it is satisfied and the negotiation is successful. It is represented as a pair: $R = (d, C_r)$, where $d \in p$ is a data element and $C_r = (C_1, C_2, C_3 \ldots C_n)$ is a set of conditions under which data element d can be released to the host. For example, the user preference for user.name is given by: $R = (\{user.name\}, ((\{admin\}, \{ours\}, \{stated\text{-}purpose, 3 - months\}), (\{admin, current, development\}, \{ours\}, \{stated\text{-}purpose, 3 - months\})))$. This preference rule contains one data element, "user.name," and two conditions. The user.name can be released to the host if any one of the above conditions is satisfied by the remote host for that data element.

vii. Condition: The condition specifies the constraints on personal information which must be satisfied in order to get access to the personal information related to it. Satisfying the condition does not in itself guarantee the release of personal information. The negotiation between a user and a remote host must be successful to get access to the personal information. The condition is a triple, including a set of purposes, a set of recipients and one retention element. It is represented as C = (Purpose, Recipient, Retention). For example, a condition may look as follow: C = ($\{admin, current\}$, $\{ours\}$, $\{stated\text{-}purpose, 3 - months\}$), where {admin, current} is the purpose, {ours} is the recipient and {stated-purpose, 3 – months} is the retention element.

viii. Alternative preference: An alternative preference is obtained from a user preference. Each combination of a condition with a data element gives an alternative preference. Therefore, the number of alternative preferences is equal to the number of conditions associated with a data element. It is represented as a pair, $R = (d, C)$. Here, d is a data element and C is one of the conditions under which a data element can be released. The alternative preferences are used during the evaluation of the offer/counter-offer made by the remote host and to produce a counter-offer. For example, given a user preference $R = (\{user.name\}, ((\{admin\}, \{ours\}, \{stated\text{-}purpose, 3 - months\}), (\{admin, current\}, \{ours\}, \{stated\text{-}purpose, 3 - months\})))$, it constructs two alternative preferences: $a_1 = (\{user.name\}, (\{admin\}, \{ours\}, \{stated\text{-}purpose, 3 - months\}))$ and $a_2 = (\{user.name\}, (\{admin, current\}, \{ours\}, \{stated\text{-}purpose, 3 - months\}))$. During negotiation, if an offer/counter-offer provided for the data element user.name matches with or is a subset of any of the alternative preferences, then the request for the data element is accepted. If it does not satisfy any of the alternative preferences, then one of the alternative preferences is selected and given as a counter-offer.

ix. Offer: This refers to a set of data elements requested by a remote host under a corresponding condition attached with each data element. It is represented as $O = (\{d_1, C'_1\}, \{d_2, C'_2\}, \{d_3, C'_3\}.....\{d_n, C'_n\})$, where $(d_1, d_2, d_3.....d_n)$ are the data elements requested under the corresponding conditions $(C'_1, C'_2, C'_3.....C'_n)$. For example, an offer for two data elements from a host would be O = $(\{user.name\}, (\{admin\}, \{ours\}, \{stated\text{-}purpose, 3 - months\}), (\{user.telecom.telephone\} (\{current\}, \{ours\}, \{stated\text{-}purpose, 3 - months\})))$. In this example, user.name is requested under the condition $(\{admin\}, \{ours\}, \{stated\text{-}purpose, 3 - months\}$ and user.telecom.telephone is requested under the condition $(\{current\}, \{ours\}, \{stated\text{-}purpose, 3 - months\})$.

x. Counter-Offer: A counter-offer is generated by both the mobile agent and the remote host. The mobile agent generates a counter offer based on the rejected request for the data element in an offer/counter-offer and user preferences corresponding to the rejected data elements. The mobile agent first evaluates the offer/counter-offer made

by the host and finds the data elements which are not acceptable. It then looks into the alternative preferences for that data element and chooses a reasonable alternative preference as a counter-offer. In the same way, a counter-offer is also generated by the host. The host evaluates the counter-offer given by the mobile agent and determines if the counter-offer is acceptable; otherwise, it responds with another counter-offer. In this study, the mobile agent only provides counter-offers while the host can provide both offers and counter-offers.

xi. Directory service: The directory service is a centralized service which provides service to both the Web user and the remote host. It is considered a trusted service in the sense that the information provided by it is assumed to be always correct. The directory service keeps all the information about the services provided by remote hosts. It stores four sets of information about a host: host ID (a unique ID associated with the host), public key of the host, host address, and the services the remote host provides. It uses user specifications (i.e.,, transaction identifier) as a search query to find the relevant host addresses, compares them with the stored information for different hosts and returns the desired information of the relevant hosts.

xii. Token: A token is a constant unique identifier provided by the host if negotiation is successful. During the agreement confirmation phase, the host signs the privacy data offer together with a generated unique token. When the mobile agent returns to the origin host, the monitoring agent retrieves the token and enters it into the user history file with the corresponding signed privacy data offer and host ID. In the future, this token can then be used to avoid any unnecessary negotiation between the host and the mobile agent over the same set of user preferences and host privacy practices.

xiii. Privacy data offer: The privacy data offer is the actual privacy policy of the host under which negotiation was achieved. In the first round, if negotiation is achieved then this privacy data offer is stored in a temporary history file (described in the next section) at the remote host and retrieved later for creating a signed negotiation agreement during the agreement confirmation phase.

xiv. Signed negotiation agreement: This is a privacy data offer along with some pertinent information signed by the remote host when the negotiation is successful. It includes four pieces of information: the privacy data offer on which negotiation was successful,

the agent trace offer, the token value, and a public certificate used for validating the signed privacy data offer at the origin host.

3.2 Overview

This section presents an overview of the privacy negotiation architecture. It describes how user preferences are collected and represented in a tree structure, how they are used at a remote host during negotiation, and finally, how they are retrieved and used for evaluation at the origin host.

3.2.1 User Preference Collection

The user specifies the preferences for his or her personal information using a graphical user interface (GUI). These preferences are represented in OWL. During setting the preferences

- The user specifies the transaction type for which user preferences have to be collected.
- Each alternative preference of a data element is ranked by assigning a score ranging from 1 to 20. This ranking divides the user preferences into four slots: critical preference slot, primary preference slot, secondary preference slot, and irrelevant preference slot.
- Eventually, an OWL file is constructed that carries user preferences for a single transaction. In the same way, an OWL file is constructed for all the transactions.

3.2.2 User Preference Representation

The collected user preferences are converted into a tree structure using a parser and are then provided to the mobile agent for negotiation purposes. During the conversion of the preferences from OWL format to tree structure

- The parser uses the transaction identifier to generate the tree. The transaction identifier is used as the root of the tree.
- The parser parses the OWL file and converts the preferences into a generalized representation where each data element is related to one or more conditions of release and the scores associated with it. These generalized preferences are now used to generate the tree.

- The P3P data hierarchy has four disjoint data sets: business, user, dynamic, and thirdparty. In this study, these data sets are used to build the tree, dividing it into four sub-parts.
- The tree is constructed by inserting all the generalized preferences into the appropriate data set. For example, user.name is inserted in the user data set and dynamic.cookie is inserted into the dynamic data set. When constructing the tree, the parser maps each generalized preference score with a unique label value, which is then assigned to the alternative preferences in the tree (relating data element and condition of release).

3.2.3 Negotiation

During negotiation with a remote host, the mobile agent uses the preference tree to evaluate an offer/counter-offer, to produce a counter-offer, and to produce an agent trace offer.

- During evaluation, the mobile agent takes each data element of the offer/counter-offer. The mobile agent compares each of these with the data element in the user preference tree and checks to see whether the conditions of release associated with that data element are satisfied. If the conditions are satisfied, it accepts that data element request; otherwise, it chooses a reasonable alternative preference (condition and data element). This way, it evaluates each data element of an offer/counter-offer and responds with an appropriate message, depending upon the result of the evaluation.
- A counter-offer is generated if the conditions concerning the requested data elements are not satisfied. In this case, the mobile agent chooses a reasonable alternative preference for all the data elements which are not acceptable (condition and data element) and supplies it as a counter-offer.
- When negotiation is achieved, the mobile agent uses label values to produce an agent trace offer of the user preferences on which negotiation is achieved.

3.2.4 Offer Evaluation

When the mobile agent returns to the origin host, the monitoring agent starts the offer evaluation process. During offer evaluation:

- The monitoring agent first maps the label values of each data element of the offers (agent trace offer) to its actual score.

- The monitoring agent then categorizes each offer into three slots on the basis of its score.
- Once the categorization is complete, the monitoring agent starts the evaluation of offers, which is done in three stages. After the first evaluation, the three best offers are selected (more than three offers if two or more offers have same score and are among the top three offers). In the second stage evaluation, only the offers selected after the first stage evaluation are evaluated for their second slot preferences. The best two host offers are selected (more than two offers if two or more offers have same score and are in top two offers). Then the evaluation moves to third stage, and the selected offers of the second stage are evaluated for their third slot preferences. Finally, one top offer is selected as an optimal offer.
- After selecting the optimal offer, the agent is dispatched to the selected host to confirm the negotiation over the agent trace offer and to collect the actual privacy data offer of the host on which negotiation was achieved.

3.2.5 Privacy Data Offer Evaluation

When the mobile agent returns to the origin host, the monitoring agent starts the privacy data offer evaluation process. During offer evaluation:

- The monitoring agent retrieves the actual user preference for the agent trace offer and matches those preferences with the privacy data offer supplied by the host.
- If the host privacy data offer matches with or is a subset of the user preferences then negotiation is said to be successful.

3.3 User Preference Collection and Representation

The user preferences are collected in OWL. It is assumed that the user is provided with a GUI where he or she can set the user preferences. The user preferences are grouped together on the basis of the transaction type (e.g., financial transaction preferences are set differently from subscription transaction preferences). Two things are taken into consideration during the collection of the user preferences. The first is the transaction identifier field, which identifies the transaction type. When specifying the user preferences, the user should specify the transaction type for which the preferences should be used. Thus each transaction type

has its own set of user preferences. The second consideration is the score field, which is used by the user to rate the importance of a user preference. The scoring of the preference by the user is used primarily for two purposes: first, when arranging the user preferences into the tree structure, and second, when evaluating the agent trace offer of hosts to determine an optimal offer. Before dispatching the mobile agent, user preferences from OWL files are retrieved and converted into the tree structure. This conversion can be done using a parser. The mechanism for conversion of user preferences from an OWL file to the tree structure is beyond the scope of this study. In the following section, only information required and used during conversion is presented.

3.3.1 Preference Ranking

The preference ranking is used to explicitly cover the user's strictness over his or her personal information. The privacy sensitivity for a data element may vary from user to user. Therefore, when setting the preferences, the user rates each preference on a scale of 1-20. The importance of the preference is judged by its score: a score of 1 has the lowest importance while a score of 20 has the highest importance. This score range sub-divides user preferences into four slots: critical preferences, primary preferences, secondary preference, and irrelevant preferences. The critical preferences ($18 < S \leq 20$, where S is the score value) are those preferences about which the user is very strict and very much reluctant to make any compromises. The primary preferences ($12 < S \leq 18$) are those preferences for which the user is comparatively less strict. The secondary preferences ($6 < S \leq 12$) are those preferences for which the user is more flexible than the previous two preference slots, and the irrelevant preferences ($1 < S \leq 6$) are those preferences about which the user is totally flexible. The number of alternative preferences for a data element may increase with a decrease in the importance of the data element. This increases the probability of achieving negotiation since the more the choices available, the greater the probability of getting a match for the conflicting preferences. The subdivision of preferences on the basis of score also provides increased flexibility to the user to choose the behaviour of negotiation according to his or her requirements. It helps the user to clearly understand the importance of the personal information and its effect on the release of that information to the host. Once a data element is assigned a score and a slot, all other alternative preferences for the same

data element are assigned in the same slot. For example, if user.name has been assigned a score of 17, all other alternative preferences for the user.name fall into primary slot and should have scores from $12 < S \leq 18$. The preference categorization is described below:

i. Critical preferences: A preference whose score ranges from 18 to 20 (i.e., $18 < S \leq 20$) falls into the critical preference slot. These preferences are considered to be the most important from user sensitivity level. In this slot, a user specifies the preferences about which he or she is very strict and very reluctant to make any compromises. Usually the preferences assigned to this slot will not have any alternative preferences. The remote host has to agree to these preferences to achieve the negotiation. If all the preferences specified have scores between 18 and 20 and do not have any alternative preference, then it becomes a "take it or leave it" scenario. This scenario matches with the scenario the current P3P standard provides. The user agent compares host privacy practices with user preferences and generates an alert if there is mismatch. The user is provided with only two choices: either he or she can abandon the transaction or accept the privacy practices of the host (i.e., there is no support for negotiation).

ii. Primary preferences: A preference whose score ranges from 12 to 18 (i.e., $12 < S \leq 18$) falls into the primary preferences slot. In this slot, the user is not very strict and wishes to conduct negotiation over his or her personal information. For example, primary (1, having score of 17) means the user does not want to release his or her credit card number. The primary (2, having score of 16) means the user may release his or her credit card information if the purpose of use is <current> and recipient is <ours>.

iii. Secondary preferences: A preference whose score ranges from 6 to 12 (i.e., $6 < S \leq 12$) falls into the secondary preferences slot. In this slot, the user is more flexible than with the preference slots described above. For example, secondary (1, having score of 11) means user does not want to release his or her email ID and phone number together. The secondary (2, having score of 10) means the may release his or her e-mail ID and phone number if the purpose of use is <current> and the recipient is <ours>. The secondary (3, having score of 9) means the user could

release his or her email ID and phone number, if the purpose of use is <current, administration> and recipient is <ours>.

iv. Irrelevant preferences: A preference whose score ranges from 1 to 6 (i.e., $1 < S \leq 6$) falls into the irrelevant preference slot. In this slot, the user is more flexible than with the preference slots described above. This slot includes all information the user does not care much about and contains maximum alternative preferences for a data element.

3.3.2 Transaction Type

Users' sensitivity about their personal information may vary according to the transaction. Therefore, a single data element may have different conditions of release for different transactions. Thus, to capture accurately the privacy sensitivity requirement of the user concerning his or her personal information, user preferences are determined separately for different transactions. The user sets preferences for all transactions beforehand and before being dispatched, the mobile agent retrieves the user preferences for a desired transaction. By arranging preferences by transaction type, a mobile agent only has to carry the required preferences for a transaction instead of carrying the user preferences for all transactions. The problem in setting preferences for each transaction separately is the redundancy of the preferences, which in turn increases the size of the preference module at the origin host. Another problem is that the user has to pre-specify the transaction before the start of the mobile agent itinerary.

3.4 Building the Preference Tree

In this study, the preferences in OWL [34, 35] are converted into a tree based structure. This conversion can be done by parsing the OWL file for each transaction and arranging all the preferences into a tree structure. The advantage of using a tree representation is that it makes evaluation of the offer/counter-offer faster, and also gives faster access to the user preferences to generate a counter-offer [35]. Two things are taken into consideration during the collection of the user preferences. Firstly, a transaction identifier field identifies the transaction type. When specifying the user preferences, the user specifies the transaction type for which the preferences should be used. Therefore, each transaction type has a single

user preference tree and the number of trees equals the number of transaction types. Secondly, a score field is used by the user to rate the importance of a user preference. The scoring of the preference by the user is used primarily for two purposes: first, when arranging the user preferences into the tree structure and second, when evaluating the agent trace offer of hosts to determine the optimal offer. The P3P data hierarchy has four disjoint data sets: business, user, dynamic, and thirdparty. In this study, these data sets are used to build the tree and they divide the tree into four sub-parts. Thus, the root node always has four child elements, namely, business, user, dynamic and third party. This conversion of OWL into a tree structure is done in such a way that the preference tree remains compatible with the P3P data hierarchy. If any data set type does not have any data elements (no user preferences are specified for that data set) then it is left empty with no child. The conversion is done by relating the condition with each data element. In the tree, the last node or leaf node of the tree represents the condition concerning the data element. A data element can have more than one condition. Thus, alternative preferences for a data element are arranged by considering its score. The alternative preferences that have higher importance or higher scores are arranged on the left side of the node, in contrast with other preferences which have lower importance.

By parsing the OWL files, the user gets a set of generalized user preferences for a transaction, and these preferences are used to construct the tree. Each preference is assigned a specific data set. For example, user.name is assigned to user data set, and dynamic.cookie is assigned to the dynamic data set. Thus, the preferences from an OWL file for a financial transaction can be parsed and represented, as shown in Figure 3.1

For example:

R_1 = (user.name, {{C_1, 14}, {C_2, 13}})

R_2 = (user.name.first, {{C_8, 14}, {C_9, 13}})

R_3 = (user.telecom.telephone, {{C_3, 15}, {C_4, 13}})

R_4 = (user.telecom.cell, {C_5, 16})

R_5 = (dynamic.cookie, {{C_6, 13}, {C_7, 12.0}})

The generalized preferences as shown above are used to construct the tree. The data user.name, user.name.first, user.telecom.telephone, user.telecom.cellphone, and dynamic.cookie have preference rules. The preferences user.name, user.name.first, user.telecom.telephone, and user.telecom.cellphone belong to the user data set, and the preference dynamic.cookie belongs to dynamic data set. In this example, the business and thirdparty data types do not have any user preferences. The preference tree is built using all the preferences for a financial transaction. Each data element, together with its parent node, is inserted in a specific data set. For example, user.name is assigned to user data set, user.name.first is assigned to user.name in the user data set, and dynamic.cookie is assigned to the dynamic data set. The converted tree using the above preferences is shown in the Figure 3.1.

The root of the tree identifies the transaction. In this example, it indicates "financial." The root node has four child elements and all preferences are arranged specifically in their respective data sets. The score is used to place the alternative preference at the right node in the tree. The alternative preferences which have higher importance are placed on the left side, compared to the preferences which have lower importance. The business and thirdparty data sets are empty (they do not have any preferences but are inserted into the tree). The data with their alternative preferences are added to their parent in the preference tree. In the above example, user.name is a child node (element) of its parent user node and user.name.first is the child node of user.name. In this way, all data elements are inserted into the preference tree.

3.5 Agent Trace Offer

If negotiation between the mobile agent and the host is achieved, then the agent trace offer is stored instead of the privacy data offer. The storage of the agent trace offer has two advantages: first, the agent trace offer makes evaluation of the offer at the origin host fast, and second, it saves bandwidth, as storing agent trace offers rather than storing privacy data offers at each host reduces the size of the mobile agent, thereby saving bandwidth over the network.

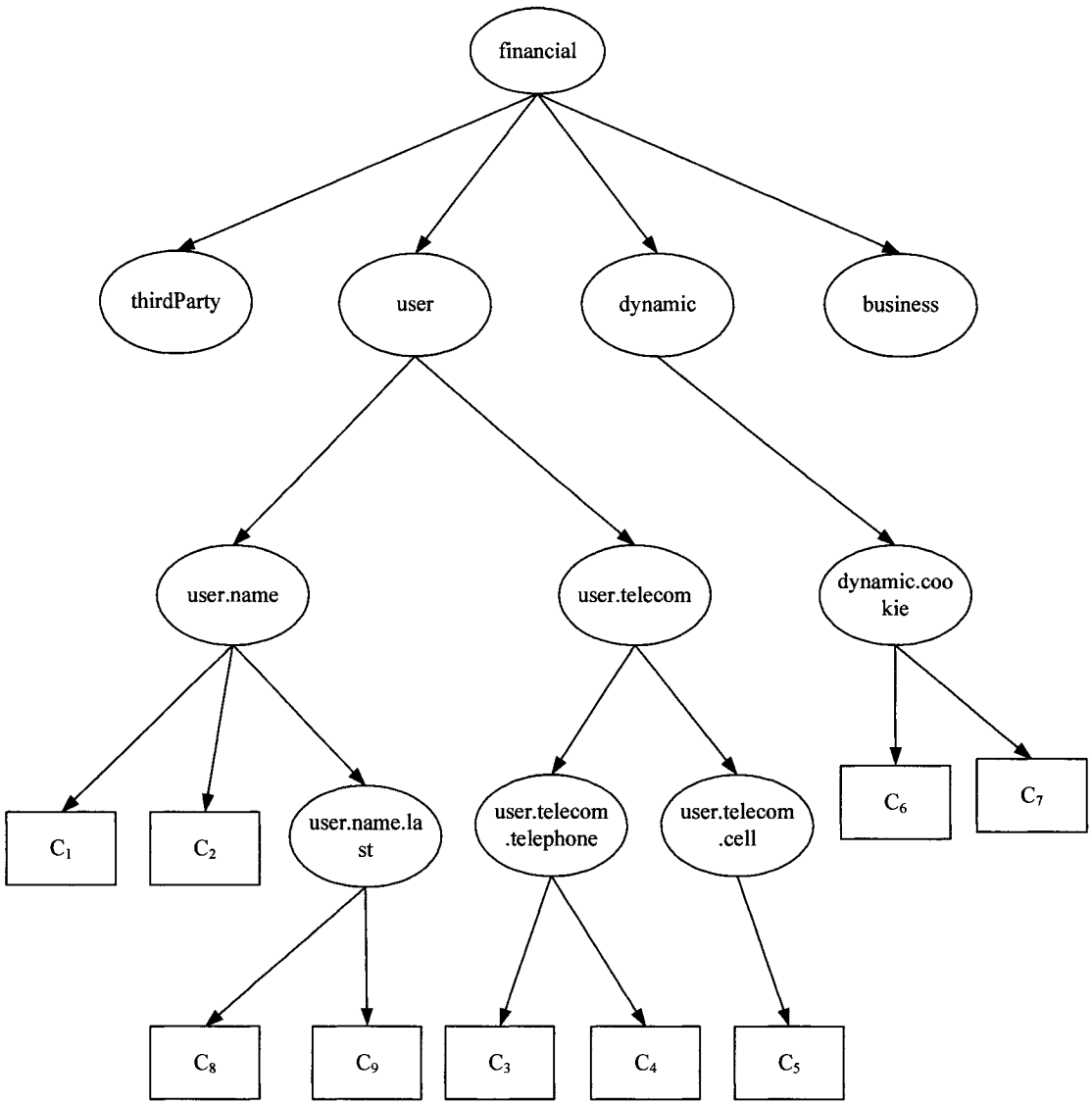

Figure 3.1: User Preference Tree Structure

This agent trace offer is produced in such a way that when the agent returns to the origin host, the monitoring agent retrieves the corresponding user preferences by looking at the agent trace offer on which negotiation was achieved. To achieve this, a simple labeling scheme is used. Each alternative preference is assigned a unique label value. Each label value actually indicates the corresponding score assigned to a user preference by the user during the setting of preferences. This label value is now used in producing the agent trace offer. Instead of storing the complete user preferences (i.e., data elements with their conditions) on which negotiation was achieved, the mobile agent only stores data elements

with their label values. For example, if the user preference is a_1 = (*{user.name}, ({admin}, {ours}, {stated-purpose, 3 – months})),* instead of storing the complete user preferences, the agent only stores a_1 = (*{user.name}, b),* where *b* is a unique label value which corresponds to the condition *({admin}, {ours}, {stated-purpose, 3 – months}).* When the mobile agent returns to the origin host, the same labelling scheme is used by the monitoring agent to retrieve the preference data from the agent trace offer for each data element. For example, at the origin host, the trace data element a_1 = (*{user.name}, b)* is retrieved as a_1 = (*{user.name}, ({admin}, {ours}, {stated-purpose, 3 – months})).*

Thus, when converting the OWL preferences into the tree structure, each alternative preference for a data element (i.e., every leaf node) is assigned a unique label value which identifies it. For example, the above preferences after assigning the unique label value can be represented as shown below:

R_1 = (user.name, {{C_1, a}, {C_2, b}})

R_2 = (user.name.first, {{C_1, c}, {C_2, d}})

R_3 = (user.telecom.telephone, {{C_3, e}, {C_4, f}})

R_4 = (user.telecom.cell, {C_5, g})

R_5 = (dynamic.cookie, {{C_6, h}, {C_7, i}})

The preference tree constructed after using the above preferences is shown in Figure 3.2

During negotiation, these label values are used to generate the agent trace offer. If negotiation is achieved between the host and the mobile agent, the label values corresponding to the user preferences are stored rather than the privacy data offer of the host or the user preference data of the mobile agent. For example, if negotiation is achieved on these user preferences, {Financial (user.name, C_1), (user.telecom.telephone, C_4), (dynamic.cookie, C_6)} where C_n is the condition over each data element, the agent trace is stored, instead of storing the above preferences or host privacy data. In this example, the agent trace is {Financial (user.name, a), (user.telecom.telephone, f), (dynamic.cookie, h)}. In this way, the mobile agent stores the agent trace offer at each host, and when the agent

returns to the origin host, these agent trace offers are evaluated and the optimal offer is determined by the monitoring agent.

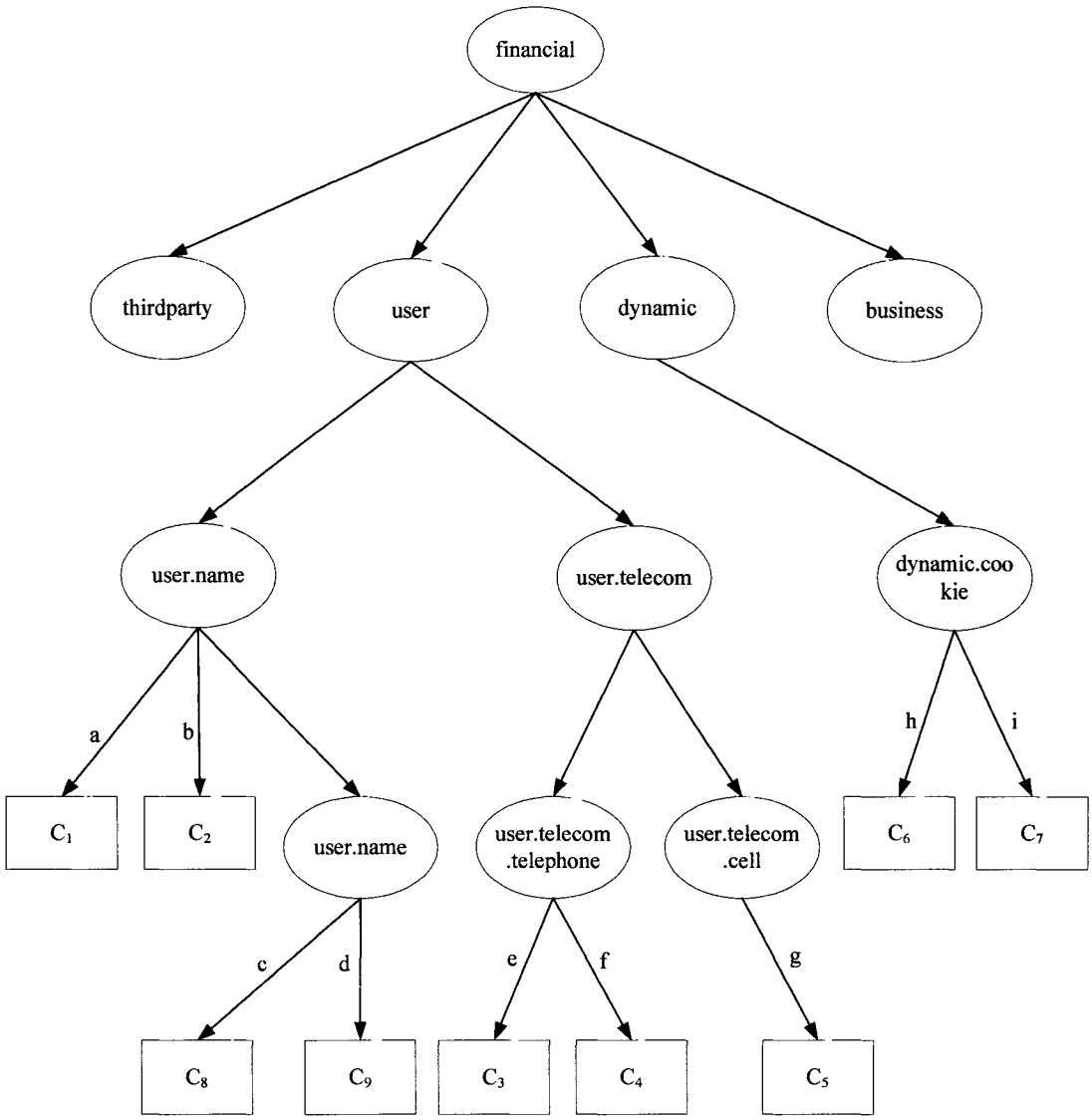

Figure 3.2: User Preference Tree after Labelling

3.6 Evaluation of the Agent Trace Offer

At the origin host, the agent trace offers are evaluated by the monitoring agent and the optimal offer is determined according to the host which has requested the least personal information and the least sensitive information (in terms of user strictness). During offer

evaluation, the monitoring agent maps the unique label value of each preference to the actual score assigned by the user. After assigning the actual score, the monitoring agent divides each agent trace offer into three slots: the first slot consists of the critical preferences and the primary preferences (all user preferences having score values between 12 to 20, i.e., $12 < S \leq 20$), the second slot consists of the secondary preferences (all preferences having score values between 6 to 12, i.e., $6 < S \leq 12$), and the third slot consists of the irrelevant preferences (all preferences having scores between 1 to 6, i.e., $1 < S \leq 6$).

The agent trace offer collected at each host might have different data elements requested in its offer. Some hosts may request a particular data element while others may not. For example, one host may request the complete user name while another host may request only the first name of the user. Thus, each agent trace offer might have different preferences in its offer. As discussed in the previous section, during negotiation, if the mobile agent does not have any user preferences for a data element requested, it rejects the request. Therefore, any change in the agent trace offer can only be affected by the user preferences carried by the mobile agent.

The difference in the number of elements requested in each offer may cause difficulty in evaluating the offers. As each offer will involve different data elements requests, comparing offers and determining and choosing an optimal offer becomes a tedious task. To avoid this problem, every agent trace offer is made the same in terms of the number of data elements requested, by adding the missing data elements in each offer. This way, every agent trace offer has the same number of data elements and the same data elements requested.

During the evaluation, if the monitoring agent does not find one data element in other offers, it appends that data element into those offers to make all offers the same. The score assigned to that data element is the highest score assigned by the user during the setting of preferences. For example, if host "A" offers a request for user.telecom.cellphone and if host "B" offers request for user.telecom.telephone, the monitoring agent assigns the highest score of user.telecom.cellphone to B's offer. Similarly, it assigns the highest score of user.telecom.telephone to A's offer. This way both offers have the same data elements in

their offers. The highest score is the score of the first condition of release concerning that data element. For example, if the user has specified three conditions for releasing user.telecom.telephone, which have scores of 17, 16 and 15, respectively, then the highest score for this preference is 17. If one offer requested user.telecom.telephone at any condition and the other host did not request user.telecom.telephone, then, to make same number of elements in both offers, a dummy request for user.telecom.telephone is added to the latter offer with the highest score it has (i.e., 17). In this way, scores for all missing data elements are assigned, and eventually all offers have the same number of data elements and same data element requests (including dummy requested data). This conversion is done during the three stages of evaluation. The steps in evaluating the agent trace offer are as follows:

1) The monitoring agent examines all offers made by the hosts and removes those offers on which negotiation was not achieved (i.e., dummy offers).
2) After eliminating dummy offers, the monitoring agent maps data elements of each offer (agent trace offer) with its actual score.
3) The monitoring agent categorizes each agent trace offer into three slots on the basis of its scores.
    a. The first slot consists of the preferences of an agent trace offer having scores between 12 and 20 (i.e., critical and primary user preferences, $12 < S \leq 20$).
    b. The second slot consists of the preferences of an agent trace offer having scores between 6 and 12 (i.e., secondary user preferences, $6 < S \leq 12$).
    c. The third slot consists of the preferences of an agent trace offer having score between 1 and 6 (i.e., irrelevant user preferences, $1 < S \leq 6$).
4) After categorization, the monitoring agent starts evaluating the offers. The evaluation is done in three stages:
    a. First, evaluation is done on the data elements of an offer for the first slot preferences.
    b. Second, evaluation is done on the data elements of an offer for the second slot preferences.
    c. Third, evaluation is done on the data elements of an offer for the third slot preferences.

5) During the first evaluation, if the monitoring agent does not find any data element in the preferences of the first slot of other offers, it appends that data element into the offers so as to make all offers the same for the first slot preferences. The score assigned to that data element is the highest score the user has assigned to that data element. After making all offers the same for the first slot preference, the monitoring agent starts counting the scores for all data elements of all the offers. By comparing all the offers, it selects the top three offers (more than three offers if two or more offers have same score and are in top three offers).
6) The monitoring agent does the same processing for the second slot preferences. Only the offers selected after the first evaluation are considered for the second stage evaluation (i.e., all other remaining offers are discarded). At first, the monitoring agent makes all selected offers the same for the second slot preferences by appending the missing data elements with the highest score assigned by the user during the setting of preferences. Then, it counts the scores for all data elements of all these offers and also adds the count of the previous evaluation to the offer count. After counting, it compares the counts, and if more than one offer needs to be selected for agreement confirmation, it chooses the top two offers (more than two if two or more offers have the same score and are in the top two offers) and moves to the third round of evaluation. However, if only one offer needs to be selected for agreement confirmation, it picks the offer which has the highest score and stops the evaluation. In this study, only one host is selected for an agreement confirmation.
7) In the third round of evaluation, only the offers selected after the second stage evaluation are considered. At first, the monitoring agent makes all selected offers the same for the third slot preferences, by appending the missing data elements to the highest score assigned by the user when setting the preferences. Then, it counts the scores for the data elements of all these offers and adds the count of second stage evaluation to the offer count. Comparing all offer counts, it selects the top offer (more than one if two or more offers have the same highest score).

3.7 Modules Present at the Origin Host

In this section, the various entities involved in the architecture and their interaction throughout the negotiation process are described. The section focuses primarily on the entities involved during negotiation at the origin host. Therefore, only a brief description is provided for the entities at the remote host.

3.7.1 Preference Module

The preference module is a simple storage module encompassing user preferences for all transaction types. Using OWL, the user sets preferences in the score range of 1-20. The importance of preferences is judged by their scores; a score of 1 has the lowest importance and a score of 20 has the maximum importance. The preferences from the OWL file are converted into a tree structure. A parser can be used to convert the preferences from an OWL representation to a tree representation. The details of conversion of OWL preferences to tree representation are beyond the scope of this study. It is assumed that the preferences are converted from OWL and arranged into the tree structure. The mobile agent retrieves the preference tree for a transaction from the preference module, before being dispatched from the origin host.

3.7.2 Monitoring Agent

The monitoring agent, also known as the user agent, takes feedback from the mobile agent. After the mobile agent returns to the origin host, the monitoring agent collects all results of the negotiation and chooses an optimal offer, by examining the agent trace offer generated at each host during the agent itinerary. At first, it checks whether each host on the itinerary has provided an offer. It then determines an optimal offer from all the agent trace offers. It uses the score associated with each preference to determine the optimal offer. After selecting the optimal offer, it provides the chosen optimal trace offer to the mobile agent. The mobile agent is dispatched again to the host whose agent offer has been selected to confirm the negotiation and to collect the privacy data offer from it.

In the first round, when the agent returns to the origin host it collects all the agent trace offers (result) generated at each host by the mobile agent. After collecting the agent trace offers it validates the offers. As the mobile agent follows a partially fixed itinerary the monitoring agent checks whether every host in the itinerary list has provided an offer or not. If it finds that any host offer is missing, it suspects a truncation attack and starts execution tracing (described in detail in chapter 4). The monitoring agent checks the integrity and authenticity of each offer by verifying the digital signature on it. This ensures that the result of the negotiation has not been tampered with and that it has been produced by the host who has signed the trace offer. After verification, the monitoring agent removes the offers of the hosts with whom negotiation has not been achieved (i.e., hosts who have signed the dummy trace) and starts evaluating the remaining agent trace offers. It uses the score associated with each preference to determine the optimal offer. After selecting the optimal offer, it provides this offer to the mobile agent, which is dispatched to collect the privacy data offer from the selected host.

In the second round, when the mobile agent returns to the origin host, the monitoring agent collects the privacy data offer on the optimal offer chosen in the first round. The monitoring agent checks its integrity and authenticity by verifying the digital signature on it and then comparing it with the agent trace offer. If the privacy data offer matches or is a subset of the agent trace offer generated in first round, the negotiation is said to be successful.

3.7.3 User History Module

The user history module maintains a user history file which stores the records of past negotiations of the hosts which were suspected to be malicious and of those with whom negotiation was successful. The monitoring agent enters a maximum of five types of information into the history file after each negotiation process. They are the host ID, the negotiation result (indicating whether the negotiation was successful or whether the remote host was identified as malicious), the transaction identifier, the token supplied by the remote host when the negotiation is successful (the host agent has signed the negotiation agreement), and lastly, the signed negotiation agreement given by the remote host during agreement confirmation in the second round. The host ID is a unique constant identifier of

each host. It is assumed that the host ID will remain the same for a specific duration. The information in the history file is entered at two stages of the negotiation process, after the first round, when mobile agent returns to the origin host, and after the second round of the mobile agent.

After the first round, the monitoring agent enters information for hosts which have been identified as malicious (if any). It enters the host ID and result of the negotiation (i.e., malicious) into the history file. In the second round, it enters the information for hosts with whom the negotiation has been successful. It enters the host ID, the result of the negotiation, the transaction identifier, the token value, and the signed privacy data offer, into the history folder. The user history module is used for two purposes: first, to filter the list of hosts to be visited by the mobile agent by removing hosts with whom past negotiation experience has been negative, and secondly to give the token to the mobile agent for the hosts with whom negotiation has been successful in the past. Before dispatching the mobile agent, the itinerary list is sent to the history module, which examines every host's record in its file. When examining the list of hosts, it checks the past outcome of negotiations (i.e., whether the negotiation was successful or the remote host was identified as malicious). It approves hosts which either have a good past record (negotiation has been successful) or no record (may be negotiating for the first time). If the outcome has been successful, it extracts the corresponding token and gives it to the mobile agent. This token is used by the mobile agent to avoid unnecessary negotiation with the same remote host over the same user preferences and policy practices. If the outcome has been malicious, that host address is eliminated from the list of hosts to be visited. After examining the complete list, a final list of hosts is supplied to the agent. The history module holds limited storage on the user device to store the records of past negotiations. Once a remote host is identified as a malicious host for any transaction type, the history module maintains an entry, identifying it as a malicious, and no future negotiation takes place with it for a specified period (communication is halted for a specific duration). The history module also maintains entries for prior transactions for which negotiation has been successful, for partial non-repudiation purposes. After a specific duration, the history module automatically removes entries from the database for malicious remote hosts, on the assumption that there may have been a change in their situation or

behaviour. It follows the first in first out method to remove these entries. Successful negotiation entries are removed on a periodic basis and only those entries whose contract duration is over are removed.

Host ID	Result	Transaction Identifier	Token	Remote Host Signed Offer
111	Pass	Financial	10001	12js586n696nsss9
222	Malicious	-	-	-
444	Pass	Financial	10002	12js56hhj696nss9
555	Pass	Medical	20001	14hshhs8484j9kf0

Table 3.1: User History Files

3.8 Modules Present at Remote host

The remote host consists of a host agent which handles all incoming agents on the platform. In this study, it is assumed that that mobile agent arriving on the remote host will not mount an attack on the host platform (i.e., the mobile agent will not behave maliciously). When an agent arrives on the host platform, it may first check the authenticity of the agent, by verifying the agent signed code and also by checking the host address in the itinerary list of the agent. After verifying the agent, it gives a context to the agent to start negotiation. When the host agent receives the request for negotiation, it starts negotiating by sending the offer to the agent. It uses the policy module to retrieve the offer for a specific transaction, the evaluation module to evaluate the counter-offer of the mobile agent and to generate an appropriate response to the counter offer, and the cryptographic module to perform certain cryptographic operations.

3.8.1 Policy Module

The policy module is a storage module encompassing host privacy practices for all the transactions. It is specified using P3P. At the start of the negotiation, the privacy policy data for the desired transaction is retrieved and presented as the offer. During negotiation, it is used to retrieve the corresponding privacy policy, which is then used for generating offers and counter offers. We have identified one specific change for specifying P3P policies.

There should be an option for specifying the alternative P3P privacy policy data. While a parser generates the offer/counter-offer for the mobile agent, the policy data is converted into a generalized representation format (as discussed in the previous section) and then presented to the mobile agent.

3.8.2 Cryptographic Module

The cryptographic module is used for performing certain cryptographic operations which are listed below.

1) When the mobile agent arrives on the host platform, it may be used to verify the digital signature of the signed code supplied by the agent
2) It is used to verify the itinerary list to see if the mobile agent is on the right host (one on the itinerary list). If the remote host does not find its own address on the itinerary list, it does not allow the agent to execute on its platform and dispatches the mobile agent to one of the remote hosts which are on the list.
3) After negotiation, the cryptographic module is used to digitally sign the agent trace offer, if negotiation is achieved; otherwise, it is used to sign the dummy trace with the private key of the host.
4) During the agreement confirmation phase, it is used to sign the privacy data offer.

3.8.3 History Module

The history module maintains two files, a temporary history file and a permanent history file. The temporary file stores information for a short period of time. It consists of four types of information: the agent ID, the agent trace offer, the privacy data offer of the host, and the retention duration. This information is entered into the temporary file if the negotiation is achieved between the mobile agent and the host. The agent trace offer is the trace offer produced by the mobile agent, which is given to the host if negotiation is achieved. The privacy data offer is the data offer or policy on which negotiation was achieved. The privacy data offer should match with the agent trace offer or should be a subset of the agent trace offer. The history module stores the agent trace offer and agent ID in the temporary history file to identify the mobile agent in the second round of negotiation (the agreement confirmation phase). The agent ID is used to identify the mobile agent, and ensures that

during negotiation, the agent has generated the corresponding agent trace offer in the temporary file. When the mobile agent returns during the agreement phase, it presents the agent ID and the agent trace offer to the host. The host compares this data with the data of temporary history file. If it finds a match, it retrieves the actual privacy data offer and digitally signs the data offer. If it does not find a match, it rejects the request for confirmation and negotiation is terminated. The latter is possible in the following situations:

1) The host does not find the agent ID in its temporary file (i.e., there is no entry in the file).
2) There is a mismatch between the agent trace offer presented (by the mobile agent in the agreement confirmation phase) and the agent trace offer the history module has in its temporary history file.

The above information is used for final confirmation of the negotiation agreement. As soon as agreement is achieved or the time allotted expires, the stored information is deleted from the temporary history file. This could happen under three scenarios: the mobile agent arrives late (the time limit is expired and the information is deleted from the file), the mobile agent presents the wrong agent trace or agent ID, or the mobile agent is behaving maliciously.

The permanent history file is a simple file encompassing information about the previous successful negotiations of the host with mobile agents (Web users). It stores eight types of information: the agent ID, the token value, the policy version, the transaction identifier, the date on which agreement was signed, the retention period, the agent trace offer, and the privacy data offer. All this information is entered into the file if negotiation between the mobile agent and the host is successful (i.e., the host agent has signed the negotiation agreement). The information in the permanent history file is mainly used for two purposes: first, to avoid unnecessary negotiation in the future with the same mobile agent, and second, to check and delete personal data from the permanent history file if the retention duration of any negotiation agreement is over. In the first scenario, the remote host uses the token value as the identification with the agent ID. The token indicates that a mobile agent which has an agent ID has negotiated successfully in the past. The host examines the agent ID and token value in the permanent history file. If it finds a match for both, it checks the current policy version and the retention time of the agreement. The remote host checks the policy version to ensure that there have not been any changes in the host policy practices. If the verification

is successful, the remote host indicates readiness to sign the negotiation agreement over the same privacy data offer by signing the corresponding agent trace offer; otherwise, if the verification fails, it asks the mobile agent to start negotiation again.

Agent ID	Agent Trace	Privacy Data Offer	Retention
111	{user.nameB*User.CellA}	user.name{(<admin,current>, <ours>), user.cell(<admin>, <ours>)}	3 months
222	{user.nameB*User.CellB}	user.name{(<admin,current>, <ours>), user.cell(<admin, current>, <ours>)}	6 months

Table 3.2: Temporary History File

3.9 Description of the Negotiation Process

The negotiation architecture works in two rounds and four processing phases: a pre-processing phase, a negotiation phase, a post-processing phase, and an agreement-confirmation phase. The pre-processing phase, negotiation phase and post-processing phase occur in the first round of the mobile agent, and the agreement confirmation phase occurs in the second round. In the first round, the mobile agent visits all remote hosts on its itinerary list and does negotiation over user preferences with each host. In the pre-processing phase, the mobile agent does all the preparation for negotiation (collecting the desired preferences and the itinerary list, and performing certain cryptographic operations). The mobile agent is then dispatched to perform negotiations at each remote host. On the remote host, it enters the negotiation phase and negotiates with the remote host over the specified user preferences.

After the negotiation phase, it enters the post-processing phase, which includes all the processing after the negotiation, such as collecting the signed offer by the host and being dispatched to the next remote host. At the next host, it enters back into the negotiation phase and negotiates with the current host. In the same way, it switches between the negotiation and post-processing phases until it returns to the origin host, where it provides all its results

to the monitoring agent. The monitoring agent examines the offers given by the hosts and the optimal agent trace offer is determined. The mobile agent is dispatched again to the selected host (the host whose negotiation offer has been chosen as the optimal offer). In the first round, after negotiations, the mobile agent collects the signed agent trace offer, if negotiation between the mobile agent and the host is achieved; otherwise, it collects the signed dummy-trace offer. In the second round, the mobile agent collects the signed privacy data offer and then returns to the origin host.

Agent ID	Token	Policy Version	Transaction Identifier	Date Signed	Retention	Agent Trace Offer	Actual Data
111	1234	1.1	Fin	08/03/2006	3 M	{username*B, usercell*A}	username{(<admin,curren>, <ours>), usercell(<admin>, <ours>)}
222	2227	1.2	Med	08/05/2006	6 M	{username*A, usercell*B}	username{(<admin>, <ours>), usercell(<admin>, <ours>)}

Table 3.3: Permanent History File

3.9.1 Pre-processing phase

The pre-processing phase is the first phase of negotiation. In this phase, all preparations for the negotiation are carried out. The mobile agent extracts the user preference tree from the preference module, and sends a request to get the list of hosts from the directory service. When it receives the list, it sends it to the history module to get the final list of the hosts to be visited, and does other cryptographic operations (such as signing the agent code to authenticate itself at the remote host). After getting the necessary input information, the mobile agent starts its itinerary and visits all the hosts in the itinerary, in no particular order.

Figure 3.3 shows the various operations performed by the agent in the pre-processing phase (the numbers indicate the sequence of operations).

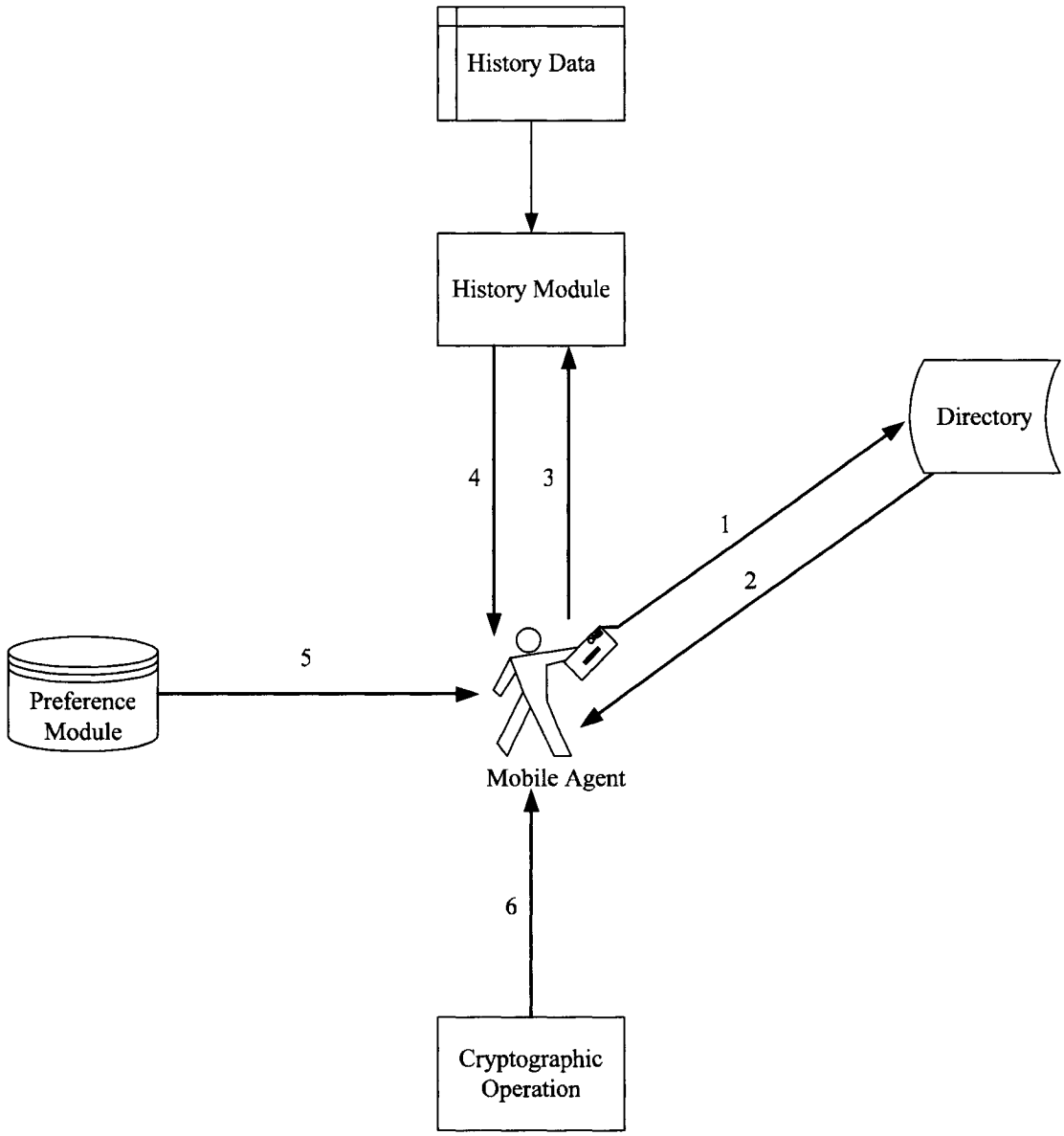

Figure 3.3: Interaction of Modules under Pre-processing Phase

On creation, the mobile agent is initialized with a unique constant ID. This ID identifies the user and remains the same for all negotiations pursued by the mobile agent. A user can assign a different ID every time a mobile agent is created, but in order to avoid unnecessary re-negotiation with the same remote host, it is mandatory that the ID should be the same as it

was when mobile agent negotiated with it in the past. Thus, it is assumed that all users have a constant globally unique ID. After initialization, the mobile agent sends a request to the directory service to get the itinerary list for the desired transaction. The directory service utilizes the user transaction identifier as the search criterion, compares it with the information stored for different hosts and returns three kinds of information about the host: the host ID (a constant unique ID associated with the host); the public key of the host (Pk_n); and the address of the host. After the mobile agent receives the list, it sends the initial itinerary to the history module. The history module examines the itinerary list and a final list of hosts is supplied to the agent. It approves hosts which have either a good past record (negotiation has been successful) or no record at all (may be negotiating for the first time). Then, the mobile agent retrieves the corresponding user preference tree to be used from the preference module, the signed agent code (the code to be executed) from the code module, and the private key from the keystore, and finally, signs the itinerary list. This signed itinerary ensures that the itinerary list has not been tampered with any way and also ensures that the mobile agent visits only the hosts which were on the list when the agent started from the origin host.

3.9.2 Negotiation Phase

The negotiation phase is the second phase of the negotiation model. It covers the complete negotiation scenario, which involves the exchange of offers and counter offers between the mobile agent and the host. After the mobile agent arrives at the remote host platform, the mobile agent code which has been signed at the origin host may be authenticated by the remote host. The remote host might search for its address in the signed itinerary by verifying and retrieving the list of hosts to be visited by the mobile agent. We assume here that if the remote host does not find its address on the itinerary list, it will not allow the agent to execute on its platform and dispatches the mobile agent to one of the remote hosts which are on the list. If the remote host finds its address on the itinerary list and the agent code is authenticated, the mobile agent is provided with a context in which it starts execution and sends a request to the host to start negotiation.

At this stage, a mobile agent can be in any one of a number of states. If it is carrying a token for that host, it first presents it and the agent ID to the host to check if negotiation can be achieved on the same user preferences on which negotiation was achieved in the past, avoiding unnecessary negotiation. The remote host verifies the token by looking into the permanent history file and determines if it can still sign the offer on the same policy. It checks both the agent ID and the token supplied and sees if it has both past negotiation records and pertinent information supporting token confirmation. It then retrieves the agent trace offer from the permanent history file, confirms the token, and responds with a message indicating readiness to sign the negotiation on the same user preferences, by signing the past agent trace offer. If the verification fails, the host may ask the agent to start a new negotiation.

During negotiation, the remote host looks into its privacy policy module, retrieves the corresponding offer (with respect to the transaction), and gives it to the mobile agent. The mobile agent evaluates the offer and accepts it if the offer matches with its user preferences. Otherwise it responds with a generated counter-offer. This counter-offer consists of both the accepted data elements and the alternate preferences for unacceptable data elements. The host evaluates the counter-offer provided and indicates whether or not it is acceptable. If it is not acceptable, the host may respond with a generated counter-offer. This process goes on until negotiation is achieved or until either the host or the agent withdraws from the negotiation. Each party may have different criteria for deciding when to withdraw from a negotiation. For example, the mobile agent may withdraw if an agreement is not achieved after a maximum of "N" rounds.

Figures 3.4 and 3.5 respectively, show mobile agent and remote host state transition diagrams for the first round negotiation process (featuring both negotiation and token confirmation over the user preferences/privacy policy). They cover both the negotiation state and the token confirmation state. The state transition diagrams do not show the authorization steps of the mobile agent at the remote host. After arriving at the remote host, the mobile agent sends the request to start the negotiation. The remote host generates the offer and supplies it to the mobile agent. The mobile agent evaluates the offer and generates the

appropriate response. In this way, exchanging certain offers and counteroffers continues between the mobile agent and the remote host. Once the negotiation for a transaction is completed, negotiation may start for another transaction. In this study, it is assumed that the mobile agent can negotiate only one transaction at a time (i.e., the agent cannot do parallel negotiations).

3.9.3 Post Processing Phase

The post-processing phase is the third phase of the negotiation model. This phase refers to the processing after the negotiation phase, which includes requesting the host to sign the agent trace offer agreed upon, or dummy trace offer, if the negotiation is not achieved. If the negotiation is achieved, the host is requested to digitally sign the agent trace offer, the ID of the next host i_{n+1} to be visited by the agent, the previously signed offer given by host i_{n-1}, and the static signed agent code. This ensures data integrity, data authenticity, non-repudiation, and also freedom from interleaving attack. In the same way, if negotiation is not achieved (the host or the mobile agent withdraws from the negotiation), the host still has to sign all the above data with one exception: a dummy trace offer value is used instead of an agent trace offer. When the mobile agent returns to the origin host, the monitoring agent examines all the offers with the host IDs in the itinerary list. If it finds that a host offer is missing, it suspects a truncation attack and starts the execution tracing discussed in the next chapter; otherwise it chooses the optimal agent trace offer among all the trace offers produced at each host, selects the corresponding host, and dispatches the mobile agent to the selected host for final confirmation and to collect the privacy data offer from the host.

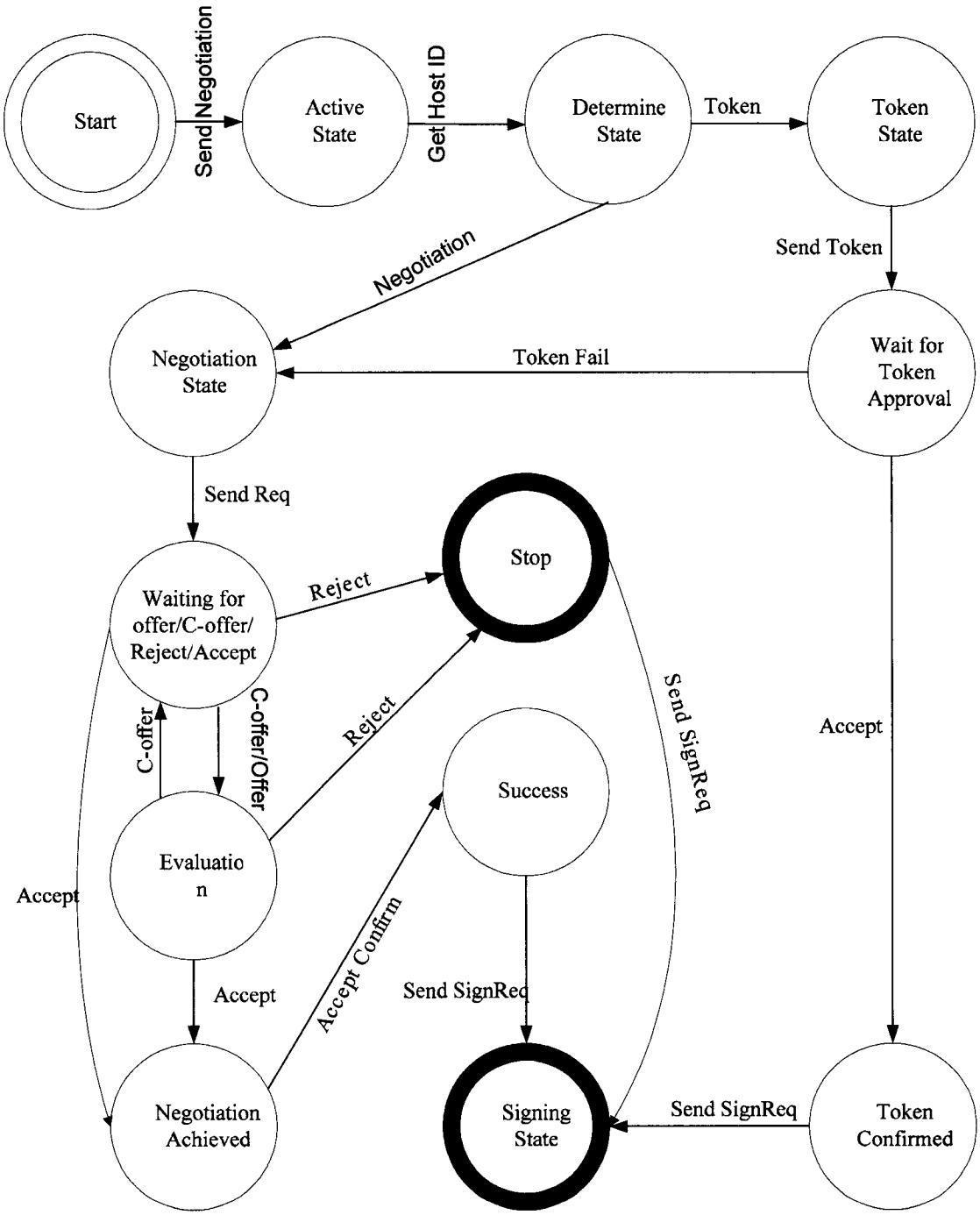

Figure 3.4: Mobile Agent State Transition

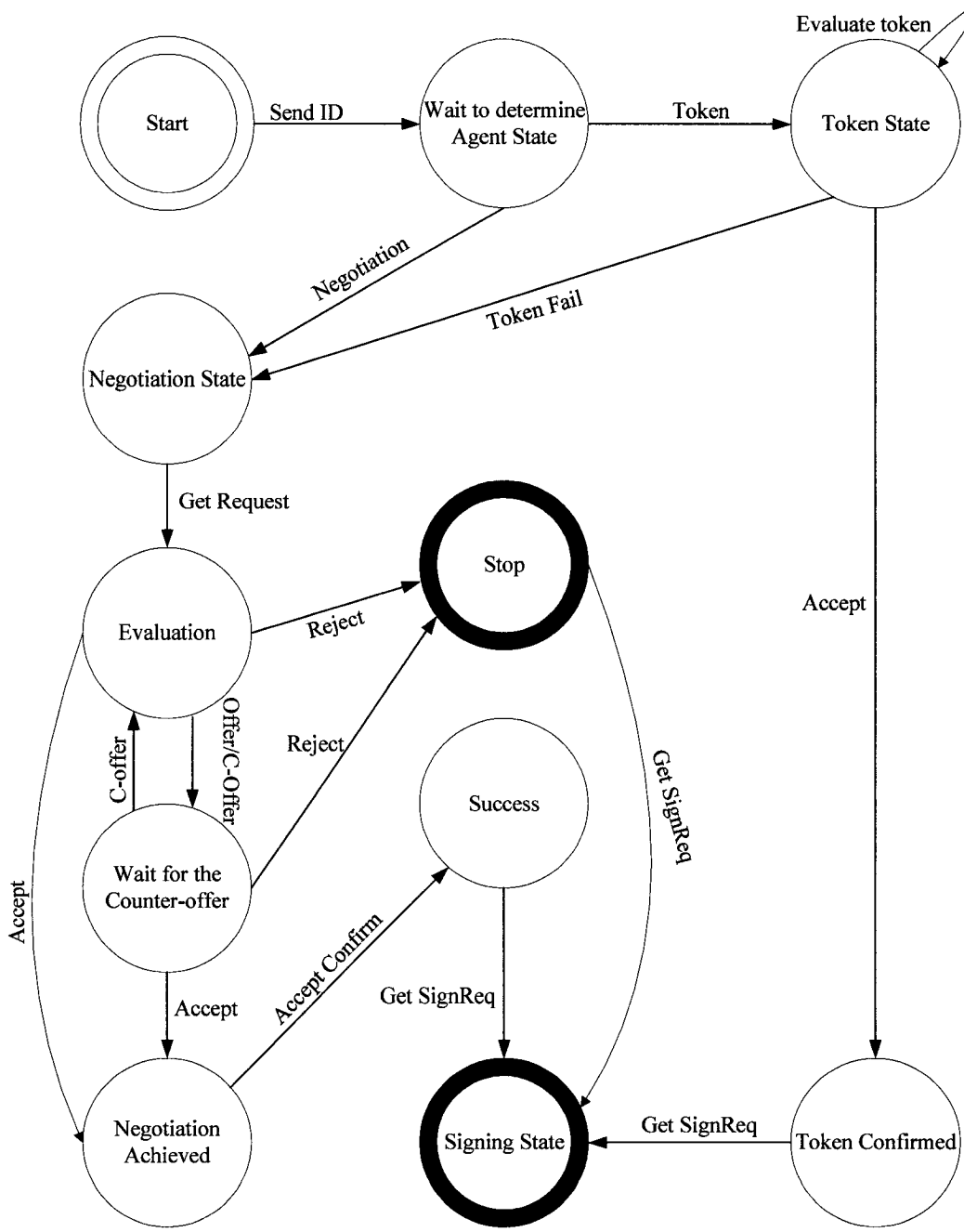

Figure 3.5: Host Agent State Transition

3.9.4 Agreement Confirmation Phase

After arriving at the selected host, the remote host provides a context where the agent starts execution. The mobile agent moves into the agreement-confirmation phase and asks the host

to sign and confirm the agent trace offer supplied in the first round of negotiation. When the mobile agent returns to the origin host, the monitoring agent retrieves the actual user preferences for the agent trace offer and matches these preferences with the privacy data offer supplied by the host. If the host privacy data offer matches the user preferences or is a subset of the user preferences, then the negotiation is said to be successful.

3.10 Summary

In this chapter we have presented the privacy negotiation architecture. This covers user preference collection and representation in the tree structure, its usage at a remote host during negotiation, and finally, how the user preferences are retrieved and used for evaluation at the origin host.

During the collection of the preferences, the user specifies the transaction type for which user preferences have to be collected. As well, he or she also ranks each preference collected, from 1 to 20. This ranking divides the user preferences into four slots: a critical preference slot, a primary preference slot, a secondary preference slot, and an irrelevant preference slot. Eventually, an OWL file is constructed that carries all user preferences for a single transaction. An OWL file is similarly constructed for all the transactions. These collected user preferences are converted into a tree structure using a parser, and are then provided to the mobile agent for negotiation purposes. During the conversion of the preferences from the OWL format, to the tree structure, the parser uses a transaction identifier and the ranking associated with each preference to generate the tree.

During negotiation with a remote host, the mobile agent uses the preference tree to evaluate an offer/counter-offer, to produce a counter-offer, and to produce an agent trace offer. During evaluation, the mobile agent takes each data element of the offer/counter-offer, compares it with the data element in the user preference tree, and checks if the conditions of release associated with that data element have been satisfied. If the conditions are satisfied, it accepts the data element request; otherwise, it chooses a reasonable alternative preference (condition and data element). This way, it evaluates each data element of an offer/counter-offer and responds with an appropriate message, depending upon the result of the evaluation.

When negotiation is achieved, the mobile agent uses label values to produce an agent trace offer of the user preferences on which negotiation is achieved. After visiting every remote host on its itinerary list, the agent returns to the origin host.

When agent returns to the origin host, the monitoring agent starts the evaluation process and determines the optimal offer. The monitoring agent first maps the label values of each data element of the offers (agent trace offer) with its actual score. The monitoring agent then categorizes each offer into three slots on the basis of its score. Once the categorization is complete, the monitoring agent starts the evaluation of offers, which is done in three stages. After the first evaluation, the three best offers are selected (more than three offers if two or more offers have same score and are among the top three offers). In the second stage evaluation, only the offers selected after the first stage evaluation are evaluated, for their second slot preferences. The best two host offers are selected (more than two offers if two or more offers have same score and are in the top two offers). Then, the evaluation moves to the third stage and the selected offers of the second stage are evaluated for their third slot preferences. Finally, the top offer is selected as an optimal offer. After selecting the optimal offer, the agent is dispatched to the selected host to confirm the negotiation over the agent trace offer and to collect the actual privacy data offer of the host on which negotiation was achieved.

Chapter 4: Trace Based Security Protocol

In this chapter, the trace based security protocol is described. First, an overview of the protocol, along with its terminology, are presented, followed by a description of the protocol and the security properties achieved. In the second part, an extension to the generic protocol proposed by Maggi and Sisto [55] is described. In order to enhance truncation resilience in this protocol, two extensions to Maggi and Sisto's protocol are proposed.

4.1 Overview

We propose a trace based security protocol to securely store the results of negotiation (i.e., the agent offer trace/dummy offer/privacy data offer). This protocol works in two rounds. In the first round, the mobile agent can be in either of two states (the negotiation state or the token confirmation state), and in the second round, it remains in the agreement confirmation state. This protocol is based on the work of Karjoth, et al. [44], who published a family of protocols to securely store the offers computed at each host by establishing a chaining relation between the previous host offer, the current host offer and the identity of the next host to be visited, as well as on the work of Maggi and Sisto [55], who extended the work of Karjoth, et al. [44], to address the limitations of their protocols and proposed a generic abstract protocol.

The trace based security protocol has been designed keeping in mind the desired security properties required, like truncation resilience and data integrity, and also considering the other requirements, like resource constrained devices and bandwidth saving characteristics. In this protocol, instead of collecting the remote host's privacy data offer, the mobile agent collects the agent trace offer after every negotiation (if negotiation is achieved). As discussed in the previous chapter, this reduces the size of the mobile agent and thereby saves bandwidth over the network. In the first round, the mobile agent collects signed agent trace offers at each host if negotiation is achieved; otherwise, it collects signed dummy trace offers. When the mobile agent returns to the origin host, the monitoring agent evaluates each agent trace offer and determines the optimal offer. The mobile agent is dispatched again to

visit the selected host to confirm the negotiation and to collect the remote host's privacy data offer. In the second round, after arriving at the selected host, the mobile agent presents the agent trace offer with the agent ID to the remote host. The remote host compares the agent trace offer and agent ID with the information in the temporary history file. If it finds a match, the host agent retrieves the privacy data offer from the temporary history file and digitally signs the privacy data offer, together with other pertinent information. If it does not find a match in the temporary file, it rejects the negotiation and responds with a rejection message. After returning to the origin host, the monitoring agent verifies the signature over the signed privacy data offer and retrieves the privacy data offer. If signature verification fails, the origin host discards the privacy data offer.

4.2 Protocol Representation

In a general way, the trace based protocol can be represented as follows:

$$M_n = D_n \,||\, O_n$$

$$D_n = D(d_n, i_n)$$

$$O_n = O(d_n, i_n, O_{n-1}, i_{n+1})$$

$$i_n \rightarrow i_{n+1} : \Pi_0\{M_0, M_1 \ldots\ldots M_{n-1}, M_n\}$$

M_n is the message or the complete set of data given by a remote host to a mobile agent. It consists of two main entities, i.e., D_n and O_n. D_n is the data supplied by the remote host which includes: i_n, current host identity; and d_n, the result of the negotiation. O_n is the digitally signed offer which includes four pieces of data: d_n, the result of the negotiation; i_n, the host identity; O_{n-1}, the previously signed offer; and i_{n+1}, the next host identity.

Before dispatching the mobile agent, the origin host will not have any previously signed offer and results of a negotiation. In order to create a chain (to generate the first offer, i.e., M_0), it generates a random number r_0 and uses it as O_{-1} (previous signed offer) and a dummy data d_0 (data supplied by the current host) which will act as the data offer given by an origin host. It also takes the hash h_0 of the agent code Π and uses it with a time stamp t to digitally sign the offer. This signed entity Π_0 is used by a remote host to validate the integrity of the

agent code. The following steps show the generation of message M_0. Here, || indicates concatenation.

$$d_0 = d_{dummy}$$

$$h_0 = h(\Pi)$$

$$\Pi_0 = \{h_0, t\}_{S_{i_0}^{-1}}$$

$$i_n \rightarrow i_{n+1} : \Pi_0\{M_0\}$$

where N = 0 and M_0 is the message given by the origin host which can be represented as follows:

$$M_0 = \left\{d_{dummy} \,||\, S_{i_0}^{-1}(d_0, r_0, \Pi_0, i_0, i_1)\right\}$$

In the same way, after negotiation at the first remote host the offer generated by a remote host will look like

$$i_n \rightarrow i_{n+1} : \Pi_0\{M_0, M_1\}$$

where N = 1 and M_1 is the message given by the origin host which can be represented as follows:

$$M_1 = \left\{(\{d_1, i_1\}_{Ek_1}, \{Ek_1\}_{E'k_0}) \,||\, S_{i_n}^{-1}(d_1, h_0, i_1, O_0, i_2)\right\}, \text{ here } (\{d_1, i_1\}_{Ek_1} \text{ indicates}$$

the encryption of the data d_n and i_n with symmetric key E_{k_1} of the host, $\{Ek_1\}_{E'k_0}$ indicates the encryption of the symmetric key with public key E'_{k_0} of the origin host and $S_{i_n}^{-1}(d_1, h_0, i_1, O_0, i_2)$ indicates the digital signature of the current host (O_1). Thus, in general, our protocol can be represented as follows:

$$D(i_n, d_n) = \begin{cases} d_{dummy} & \text{if } i_n = i_0 \\ (\{d_n, in\}_{Ek_n}, \{Ek_n\}_{E'k_0}) & \text{otherwise} \end{cases}$$

$$O(d_n, i_n, O_{n-1}, i_{n+1}) = \begin{cases} S_{i_0}^{-1}(d_0, r_0, \Pi_0, i_0, i_1) & \text{if } i_n = i_0 \\ S_{i_n}^{-1}(d_n, h_0, i_n, O_{n-1}, i_{n+1}) & \text{otherwise} \end{cases}$$

$$i_n \rightarrow i_{n+1} : \Pi_0\{M_0, \ldots\ldots, M_n\}$$

In the negotiation state, if negotiation is achieved, d_n is T_{mag}; otherwise, d_n is d_0. T_{mag} is the agent trace offer generated after negotiation, if it is achieved.

$$D(i_n, d_n) = \begin{cases} d_o & \text{if } i_n = i_0 \\ (\{T_{mag}, in\}_{Ek_n}, \{Ek_n\}_{E'k_0}) & \text{if negotiation is achieved} \\ (\{d_o, in\}_{Ek_n}, \{Ek_n\}_{E'k_0}) & \text{otherwise} \end{cases}$$

$$O(d_n, i_n, O_{n-1}, i_{n+1}) = \begin{cases} S_{i_0}^{-1}(d_0, r_0, \Pi_0, i_0, i_1) & \text{if } i_n = i_0 \\ S_{i_n}^{-1}(T_{mag}, h_0, i_n, O_{n-1}, i_{n+1}) & \text{if negotiation is achieved} \\ S_{i_n}^{-1}(d_0, i_n, h_0, O_{n-1}, i_{n+1}) & \text{otherwise} \end{cases}$$

In the token confirmation state, the above protocol is the same, except that d_n is T'_{mag} (the agent trace offer of the past negotiation). In this case, the remote host retrieves the agent trace offer from the permanent history file and digitally signs it.

$$D(i_n, d_n) = \begin{cases} (\{T'_{mag}, in\}_{Ek_n}, \{Ek_n\}_{E'k_0}) & \text{if token is confirmed} \\ \text{Agent switches to negotiation state} & \text{otherwise} \end{cases}$$

$$O(d_n, i_n, O_{n-1}, i_{n+1}) = \begin{cases} S_{i_n}^{-1}(T'_{mag}, h_0, i_n, O_{n-1}, i_{n+1}) & \text{if token is confirmed} \\ \text{Agent switches to negotiation state} & \text{otherwise} \end{cases}$$

It is clear that, if token confirmation fails for any reason, the agent enters the negotiation state and starts negotiation over user preferences with the host agent.

In agreement confirmation, d_n represents two types of data, the privacy data offer (i.e., PO_n) and either the agent trace offer generated during negotiation in the first round (i.e., T_{mag}), or the past agent trace offer (T'_{mag}) on which token confirmation has been achieved in the first round of the mobile agent. If more than one host offer is selected for the agreement confirmation phase, the protocol representation looks as follows:

$$D(i_n, d_n) = \begin{cases} d_0 & \text{if } i_n = i_0 \\ (\{PO_n, T_{mag}, in\}_{Ek_n}, \{Ek_n\}_{E'k_0}) & \text{if negotiation is successful} \\ d_0 & \text{otherwise} \end{cases}$$

$$O(d_n, i_n, O_{n-1}, i_{n+1}) = \begin{cases} S_{i_0}^{-1}(d_0, \Pi_0, i_0, r_0, i_1) & \text{if } i_n = i_0 \\ S_{i_n}^{-1}(PO_n, T_{mag}, h_0, i_n, O_{n-1}, i_{n+1}) & \text{if negotiation is successful} \\ S_{i_n}^{-1}(d_0, h_0, i_n, O_{n-1}, i_{n+1}) & \text{otherwise} \end{cases}$$

If only one host offer is selected, the mobile agent carries only the agent ID and the agent trace offer and there is no chaining of offers in the second round. In this study, we assume that only one host offer is selected. Thus, in the second round, the mobile agent only collects the privacy data offer from the selected host and returns to the origin host.

4.3 Description of the Protocol

The trace based protocol uses encapsulation of the offers to protect the offer supplied by each host during the agent's itinerary. Before being dispatched, the mobile agent retrieves the signed agent code from the code module, signs the filtered itinerary list it has received from the history module and creates an initial dummy encapsulated offer. The signed agent code is used by the mobile agent to confirm that the agent code has not been tampered with. It can also be used for source verification at the remote host. The signed itinerary is used to ensure that the mobile agent does indeed visit the hosts which were on the itinerary list when the agent started from the origin host. The initial encapsulated offer O_0 created is slightly different from the offers given by other hosts. The offer is a dummy offer, as the origin host is not offering anything; thus, to create the chain it uses dummy data and creates a dummy offer. Also, the origin host generates a random number to use as the previous signed offer, because there is no previous encapsulated offer. It signs the dummy data together with the generated random number and identity of the next host to create the encapsulated initial dummy offer. This offer is used by the next host to create the encapsulated offer. The offer created at the origin host and the other hosts is given by the following:

$$O_0 = S_{i_0}^{-1}\left(d_0, \Pi_0, i_0, r_0, i_1\right)$$

$$O_j = S_{i_j}^{-1}\left(d_j, h_0, i_j, O_{j-1}, i_{j+1}\right) \text{ Where j} = (0 < j \leq n)$$

Figure 4.1 shows how each host on the list appends the encapsulated offer. In the above figure an O_j can be the signed agent trace offer, the signed dummy trace offer, or the signed past agent trace offer. The protocol utilizes a chaining mechanism to create a set of encapsulated offers. This chaining mechanism prevents platforms from maliciously removing or replacing any of the encapsulated offers. Each encapsulated offer O_j computed by P_j contains a chaining relation, which is the encapsulated offer O_{j-1} received from the previous platform and the identity of the next host P_{i+1}. The previous encapsulated offer O_{j-1}

is used in computing the new encapsulated offer O_j to link the new encapsulated offer to it. Also, the inclusion of the identity of the next platform P_{i+1} in the chaining relation ensures that only the intended recipient host can make the next offer.

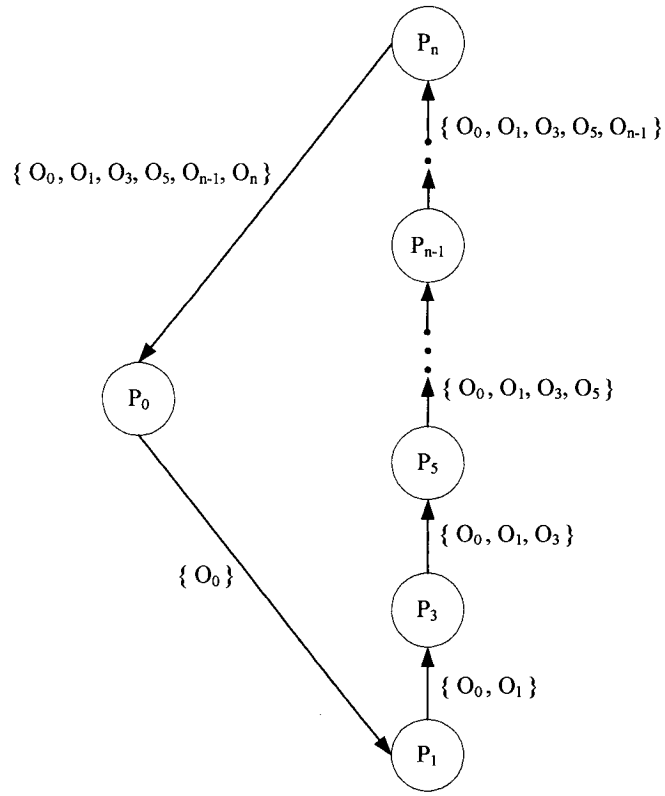

Figure 4.1: Encapsulation of Offer's

4.4 Working of the Protocol in different states

Overall, this protocol works in two rounds, in which the mobile agent switches among three states at the remote host, depending upon the various scenarios described below.

4.4.1 Negotiation State

In the negotiation state, the mobile agent does negotiation with the remote host, which may involve the exchange of a certain number of offers and counter offers.

i. The mobile agent enters the negotiation state if it does not have a token (supplied by the remote host if negotiation was successful) for the current host. This happens in the following situations:

- If the mobile agent is negotiating for the first time with the remote host (i.e., there is no past negotiation record with that host).
- The mobile agent has successfully negotiated with the remote host but the corresponding token has been removed from the user history folder (this can happen if the user preferences have been changed by the user). The user's priorities concerning personal information may have changed; thus, the token value may not be of any use. As well, the duration for which the negotiation agreement was signed may have expired and there may have been a change in the user privacy sensitivity level about his or her personal information.

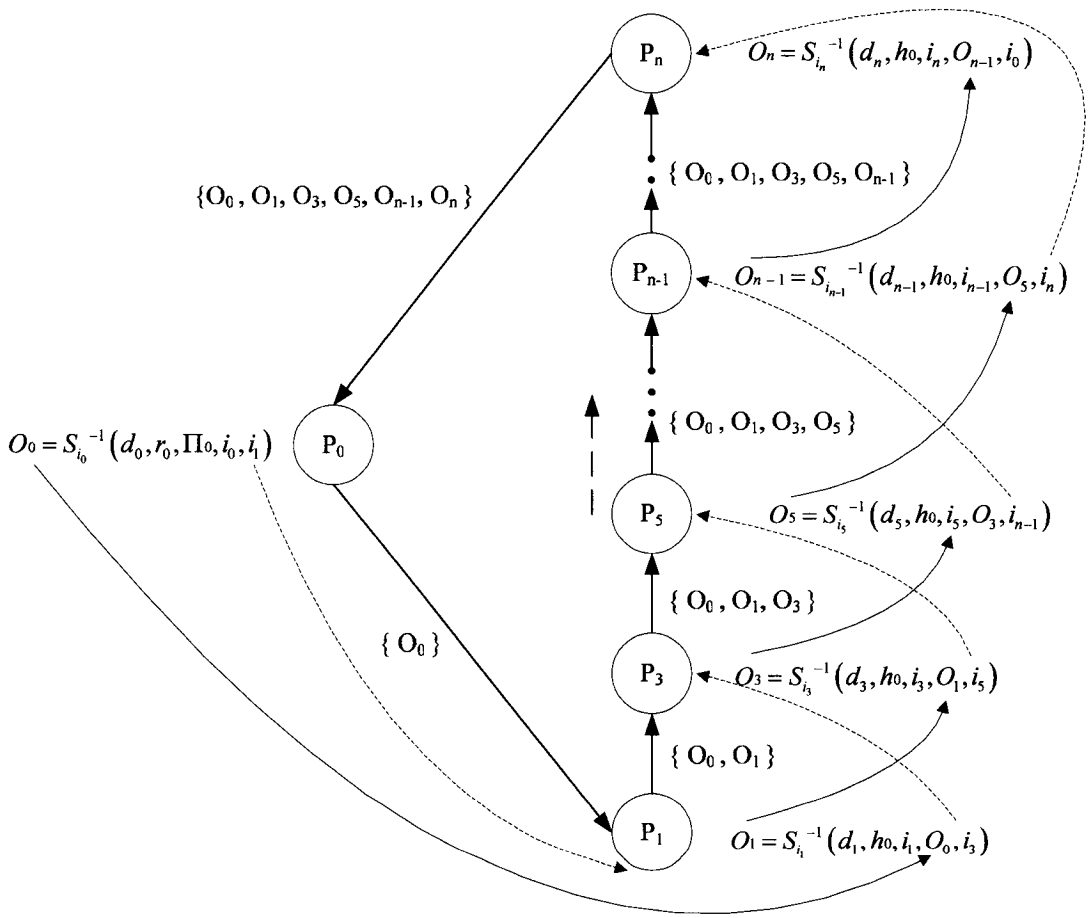

Figure 4.2: Chaining in the Encapsulated Offer's

- The mobile agent has negotiated with the host but the negotiation was not successful; thus, no token is found in the user history file.
- The mobile agent has negotiated with the host and the negotiation was successful but either the mobile agent or the host has changed its ID, due to which token confirmation fails, and the mobile agent switches to the negotiation state.

ii. When the mobile agent presents the token to the remote host and token confirmation fails, the host agent asks mobile agent to start negotiation. This happens in the following situations:

- The token presented by the mobile agent does not match with the token the remote host has in its permanent history file. This might occur if there has been an error in receiving the token or the mobile agent has presented the wrong token.
- The remote host does not have any past record of negotiation with the mobile agent. It is negotiating for the first time or has had no successful negotiation with the remote host in the past.
- The remote host does find a match in the permanent history database but there has been a change in the policy version; the host's privacy practices have been changed, and thus, the host cannot sign the contract on the same policy on which negotiation was successful in the past.

In any of the above situations the mobile agent negotiates with the host. If negotiation is achieved, the remote host is required to sign the agent offer trace on which negotiation was achieved. If negotiation is not achieved, the host needs to sign the dummy trace offer.

4.4.2 Token confirmation state

A mobile agent at a remote host is in token confirmation state if it is carrying a token and none of the above conditions apply. This means that the mobile agent has successfully negotiated with the host, and the token is validated by the remote host, and the host is ready to sign the offer trace on which previous negotiation was done. In this stage, after validating the token, the remote host signs the original agent trace offer on which negotiation was achieved in the past, together with the pertinent information and appends this to the chain of

offers. This way, the agent visits all the hosts on its itinerary list, and at each host, can be in either negotiation state or token confirmation state.

4.4.3 Agreement confirmation state

When the mobile agent returns to the origin host, the optimal offer is determined. The mobile agent is dispatched again to the selected host to confirm the negotiation. After arriving at the remote host, the mobile agent presents the agent trace offer and agent ID to the remote host, which validates the trace offer by looking into the temporary history file. After validating the agent trace offer, the remote host retrieves the privacy data offer from the temporary history file and signs the offer, together with the pertinent information. After signing the offer, the host makes an entry into the permanent history file. It enters six types of information into the file, namely, the agent trace offer, the token, the privacy data offer, the policy version, the date on which privacy data offer was signed, and the retention duration. The main advantage of including token confirmation in this protocol is the ability to avoid unnecessary negotiation, thereby increasing the efficiency of the protocol. The permanent history information is used during the token confirmation stage, where the host compares the token presented by the mobile agent with the token in its permanent history file and determines if it can sign the offer on the same agent offer trace and privacy data. It can be clearly seen that if the agent trace offer is validated and if the host platform cannot give a better offer than it gave before, the remote host agent gets ready to sign the same past offer. This saves time and computation power for both the mobile agent and the host platform.

4.5 Tasks performed by the protocol

A trace based protocol allows a mobile agent to securely collect the results of negotiations at various hosts during its agent itinerary. There are three distinct actions taken in the trace based protocol. First, the origin host retrieves the hash of the agent code, signs the host's itinerary list and creates the initial encapsulated offer. Next, when the mobile agent is received by the remote host, the remote host agent verifies the integrity of the code, performs negotiation, identifies the next host, and creates and submits an offer to the mobile agent. This second task is done at each host on the itinerary list until the mobile agent

reaches the origin host. Finally, at the origin host, the monitoring agent receives the results, processes them, chooses the optimal offer and dispatches the mobile agent to confirm the negotiation and to collect the privacy data offer of the host.

4.5.1 Creating and Dispatching the Mobile Agent

To begin the protocol, the mobile agent retrieves the hash of the agent code from the code module, signs the filtered itinerary list it got from the history module and creates an initial dummy encapsulated offer. To create the initial encapsulated offer O_0, the mobile agent generates a secret random number r_0 and decides who the first recipient i_1 will be. Then, the originator signs the dummy offer, the random value and the identity of the next host, to create the dummy offer O_0.

$$O_0 = S_{i_0}^{-1}(d_0, \Pi_0, i_0, r_0, i_1)$$

4.5.2 Receiving a Mobile Agent

After the mobile arrives at the remote host, the host first performs a couple of checks before providing context to it to start negotiation. First, the host verifies the digital signature over the agent code; if it is verified, the host verifies the itinerary list and checks if the host is the intended recipient or not. If not, the host simply forwards the agent to the correct recipient, after looking at the itinerary list. If both of the above conditions are verified, the host agent provides an execution context to the mobile agent to start negotiation.

As described above, in the first round, the mobile agent can be in either of two states, depending on whether the agent has negotiated before with the host and whether the negotiation has been successful. In both states, if negotiation is achieved, the host makes an offer, though if no offer can be made (i.e., negotiation is not achieved), a dummy offer is created. The host must submit an offer to the mobile agent in order to maintain the chaining relation in the set of encapsulated offers. Upon completion of negotiation, the next host address (which is used in creating the offer of the current host) is determined. Next, the host platform creates the encapsulated offer, by first taking the previous host's encapsulated

offer, hash of the static agent code, and the identity of the next platform, and signs this information together with its offer and its ID i_j:

$$O_j = S_{i_j}^{-1}\left(d_j, h_0, i_j, O_{j-1}, i_{j+1}\right) \text{ where j} = (0 < j \leq n)$$

This encapsulated offer is then appended to the current set of offers. After submitting an agent trace offer, the host makes an entry into its temporary history file and enters three types of data into the temporary history file. These data are the agent ID, the agent offer trace (trace produced by agent) and the privacy data offer of the host. Then, it sends the mobile agent to the next host platform. This way, the agent visits every host in its itinerary list and returns to the origin host.

4.5.3 Receiving the mobile agent at the origin host and processing the results

When the originator P_0 receives the mobile agent, the monitoring agent performs a series of checks. First it checks whether or not every host has been visited by the mobile agent; second, it verifies the hash over the agent code in each offer given by the hosts, and sees if the code has been tampered with or not. In the first case, the monitoring agent checks whether or not every host on the itinerary list has given an offer. If verification fails, it suspects a truncation attack and starts execution tracing to identify the malicious host. In the second case, it suspects that the agent code has been tampered with and discards all offers. If everything is verified, the monitoring agent starts retrieving the chain of encapsulated offers.

The monitoring agent evaluates each offer trace and determines the optimal offer. The mobile agent is dispatched again to visit the selected host to confirm the negotiation and to collect the privacy data offer of the host. In the second round, after arriving at the selected host, the mobile agent presents the agent trace offer with the agent ID. The remote host compares these entities with the entry it has in its temporary history file and determines its validity. If both entities match with the entry it has, the host agent retrieves the privacy data offer from the temporary history file and digitally signs it. If the remote host does not find a match in the file, it rejects the negotiation confirmation. After returning to the origin host, the monitoring agent verifies the signature on the signed offer and retrieves the data offer. If the signature verification fails, it discards the privacy data offer.

4.6 Security Analysis of the Protocol

A malicious host can mount several types of attacks on a mobile agent, such as altering the current state of the mobile agent, substituting the agent code in order to collect signed data which are then attached to the original set of offers, truncation of better offers provided by other hosts, and appending new encapsulated offers to the truncated set of encapsulated offers by modifying or replacing the agent code. In the following section, various security properties achieved by the proposed protocol are described.

4.6.1 Truncation Resilience

In general terms, a truncation attack is the truncation of a host offer by one or more colluding malicious hosts. If two hosts are malicious and if they collude, they can truncate the list of hosts visited between them simply by saving the state when the mobile agent visits the first malicious host and restoring the same state when it visits the second malicious host. A truncation attack can be easily detected in the protocol under discussion, since the mobile agent follows a partially fixed itinerary (i.e., the list of hosts to be visited is fixed and known to the origin host at agent departure from the origin host). Thus, when the monitoring agent finds an offer for every host in the list, it concludes that no host offer was truncated, and thus, the itinerary list contains no malicious hosts. If any host offer is missing, it suspects truncation of host offers.

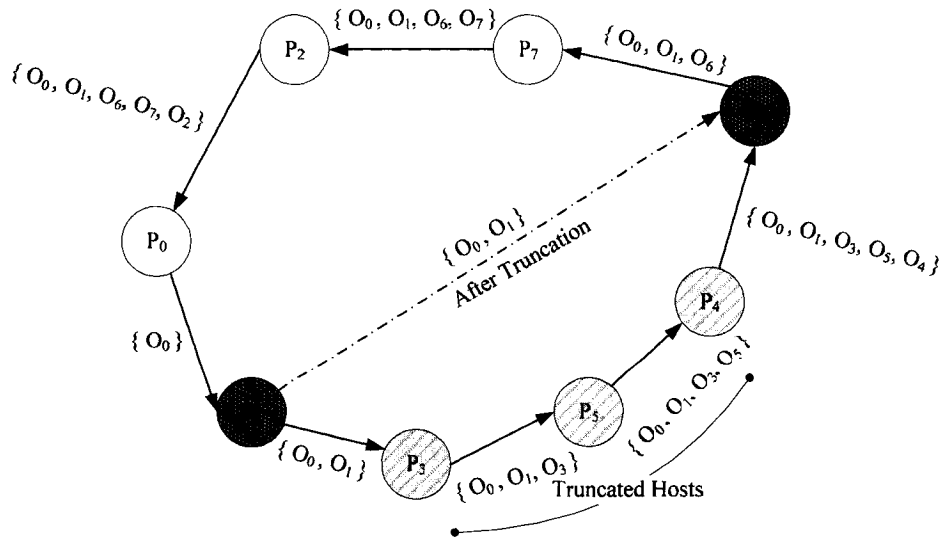

Figure 4.3: Truncation of host P_3, P_5, P_4

The agent starts from origin host P_0 and visits all the hosts in its itinerary list, for example, $\{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_0\}$. From Figure 4.3, it can be observed that the actual order of agent visits to the hosts on its itinerary list was $\{P_1, P_3, P_5, P_4, P_6, P_7, P_2, P_0\}$. In this example, P_1 and P_6 are malicious. They truncate the offers provided by P_3, P_5, and P_4 just by saving the state when the mobile agent visits the first malicious host P_1 and then restoring the same state when it visits the second malicious host P_6. This action again establishes a chaining relation between the host offers. If the mobile agent was to be free roaming, it would have been very difficult to detect a truncation attack, but in this case, the agent follows a partially fixed itinerary. Therefore, the monitoring agent can easily detect all the hosts which have been truncated, though not the malicious hosts who have truncated the host offers. To identify these malicious hosts, an execution tracing mechanism proposed in the next section for the free roaming mobile agent can be deployed, with slight modification (by sending a request message to all the hosts whose offers are suspected to be truncated).

During execution tracing, the monitoring agent picks all the hosts suspected of having not been visited or of being truncated by malicious hosts and sends a common message to them. All the hosts respond with the execution trace of the mobile agent. The execution trace is the offer generated after negotiation, when the agent visited the remote host. The monitoring agent extracts three types of information from each response it received, namely, the actual data offer d_n, the previous signed offer (i.e., O_{n-1}, the offer from the host visited before the agent arrived at the current host), and the next host address (i_{n+1}, the host that the agent is to visit next). Then, the monitoring agent establishes a chain by combining the O_{n-1} and i_{n+1} received from each host execution trace. Then, it observes where the chain has been truncated. It then matches the last and the first host of this chain with the chain it received originally from the mobile agent. In our example, the monitoring agent sends a common message to P_3, P_5, and P_4. All three hosts respond with the execution trace. After examining their responses, the monitoring agent makes a chain and sees from which point the chain got broken. In this case, the chain would look like $P_1 \rightarrow P_3 \rightarrow P_5 \rightarrow P_4 \rightarrow P_6$.

After comparing the chain it has received from the agent and the above chain established for the truncated hosts, the users can discover the malicious hosts. In our example, the

monitoring agent finds that hosts P_1 and P_6 are malicious. It enters this record into the user history module to avoid any further visits by the mobile agent to these hosts.

The main advantage of this mechanism is improved truncation resilience by identification of malicious hosts through observation of the chain and discovery of where the chain has been broken. On the other hand, its drawbacks are increased computational costs and the requirement that the host send the execution trace if requested by the monitoring agent during execution tracing.

4.6.2 Interleaving Attack

An interleaving attack is mounted by the remote host by suspending the execution of the mobile agent. The remote host copies the code and data of the mobile agent and creates its own mobile agent. Then, it sends that mobile agent to get offers from other hosts using characteristics of the origin agent. After getting all the offers, it appends these offers into the suspended original mobile agent and allows it to start its normal journey. In general, in interleaving attacks, a malicious host uses parts of an agent. The new agent is used to calculate certain results, using characteristics of the original agent. These results are incorporated into the chain of results, as if the original agent had produced them. This way an attacker can modify or truncate the partial chain of offers. For example, replacing the offers collected at previously visited hosts from the partial chain by the arbitrary results collected by the new mobile agent.

In our protocol if a malicious platform intends to modify or replace the code, it also has to update the hash value of the agent code. When the replaced code or modified code is executed on another host, during the signing of the offer, the current host also includes the hash of the agent code with the offer. The agent trace offer or dummy trace offer provided by the host is signed together with the hash h_0 of the agent static data Π. This is done to bind the agent trace offer/dummy offer to the unique agent identity, thus making sure that T_{mag} is really the data produced by the agent instance identified by h_0 on host i_n. This ensures that the agent having static code Π and hash h_0 has really been executed on host i_n producing data T_{mag}. Thus, when the agent returns to the origin host, the monitoring agent verifies the

hash of the code it has received with each offer with the original hash of the agent code. If it finds a mismatch between them, it suspects tampering of the agent code and discards all offers.

4.6.3 Data Authenticity

Data authenticity determines the origin of data or where the data was generated. The easiest way to achieve data authenticity is to ask the remote host to sign the offered data with its private key. In the proposed protocol, the host signs the offer data at each state, as during the negotiation state and token confirmation. If negotiation is achieved, it signs the agent trace offer with its private key; this ensures that the agent had visited the particular host and the corresponding offer has been generated at that host. If negotiation is not achieved, a dummy trace offer given by the agent is signed by the host. This confirms that the agent has visited the host and negotiation was not achieved (negotiation failed). As in the agreement confirmation state, the remote host platform signs the privacy data offer together with the agent trace generated during the first round. This verifies the identity of the host and confirms that the agent has visited a particular host and that the corresponding offer supplied has been generated at that host.

4.6.4 Data Confidentiality

Data confidentiality is achieved by transforming the data in such a way that no party other than the intended receiver can interpret it correctly. In the proposed protocol, this is achieved by encrypting the data with a symmetric key of the remote host and then encrypting that key with the public key of the origin host, so that no other intermediate host can see it. When the mobile agent returns to the origin host, the monitoring agent decrypts the key with the origin host private key and then decrypts the data with the symmetric key provided by the remote host.

4.6.5 Data Integrity

The architecture described in the previous chapter shows that the itinerary list is already known before the mobile agent is dispatched. Therefore, data integrity can be achieved by establishing a chaining relationship between the hosts by giving the immediate previously

signed offer and next host identity to the current host to sign its offer [44, 55]. Because of this chaining relationship, deletion, insertion, and substitution of host attacks become very difficult without detection.

i. Deletion Attack: In a deletion attack, a data item is eliminated from the set representing the protected area contents. If a single host is malicious, it cannot delete a host without violating the chaining relation because, after deleting the host, it also needs to change the previously signed offer of the host from which it wants to re-establish the chain (i.e., a single host cannot delete the host's offer). For example, as shown in Figure 4.4, if host P_4 is malicious, and wants to truncate the offers of hosts P_3 and P_5, it needs the support of host P_1 to include its identity as the next host in the offer given by host P_1. Thus, a single malicious host cannot delete the host offers without violating the chaining relation. But if more than one host is malicious, the hosts can collude and delete the hosts visited between them (as explained in the truncation attack section). As the itinerary is partially fixed, the monitoring agent can suspect that a host may have been deleted from the protected container but cannot detect which hosts have been malicious. To detect that, the execution tracing discussed in the previous section is used.

ii. Insertion Attack: In an insertion attack, a new data item is added to the set representing the protected area contents. If a single host is malicious, it cannot insert offers without violating the chaining relation (i.e., it needs the support of at least one host to re-establish the chain), but if more than one host is malicious, the hosts can collude and insert offers, and can also easily establish the chaining relation by starting the chain from the first malicious host and ending the chain at the second malicious host. The insertion of the offers can be done in two ways; the first is by simply inserting fake offers for the unvisited hosts. In this case, the offer for every unvisited host needs to be signed by the host private keys, and if a colluding host inserts fake offers, the host will get detected at the origin host during signature verification (the signature verification on those offers will fail). If signature verification fails, the origin host suspects an insertion attack and discards all offers. The second way to insert offers is by modifying the code of the agent, and creating a new mobile agent, sending the mobile agent to the unvisited hosts, collecting the offers, and appending these offers into the actual mobile

agent in such a way that the chain does not get violated. This attack can also be called as an interleaving attack (as discussed in previous section), and the execution tracing described above also holds true in detecting an insertion attack. Again, on detection, all offers are discarded.

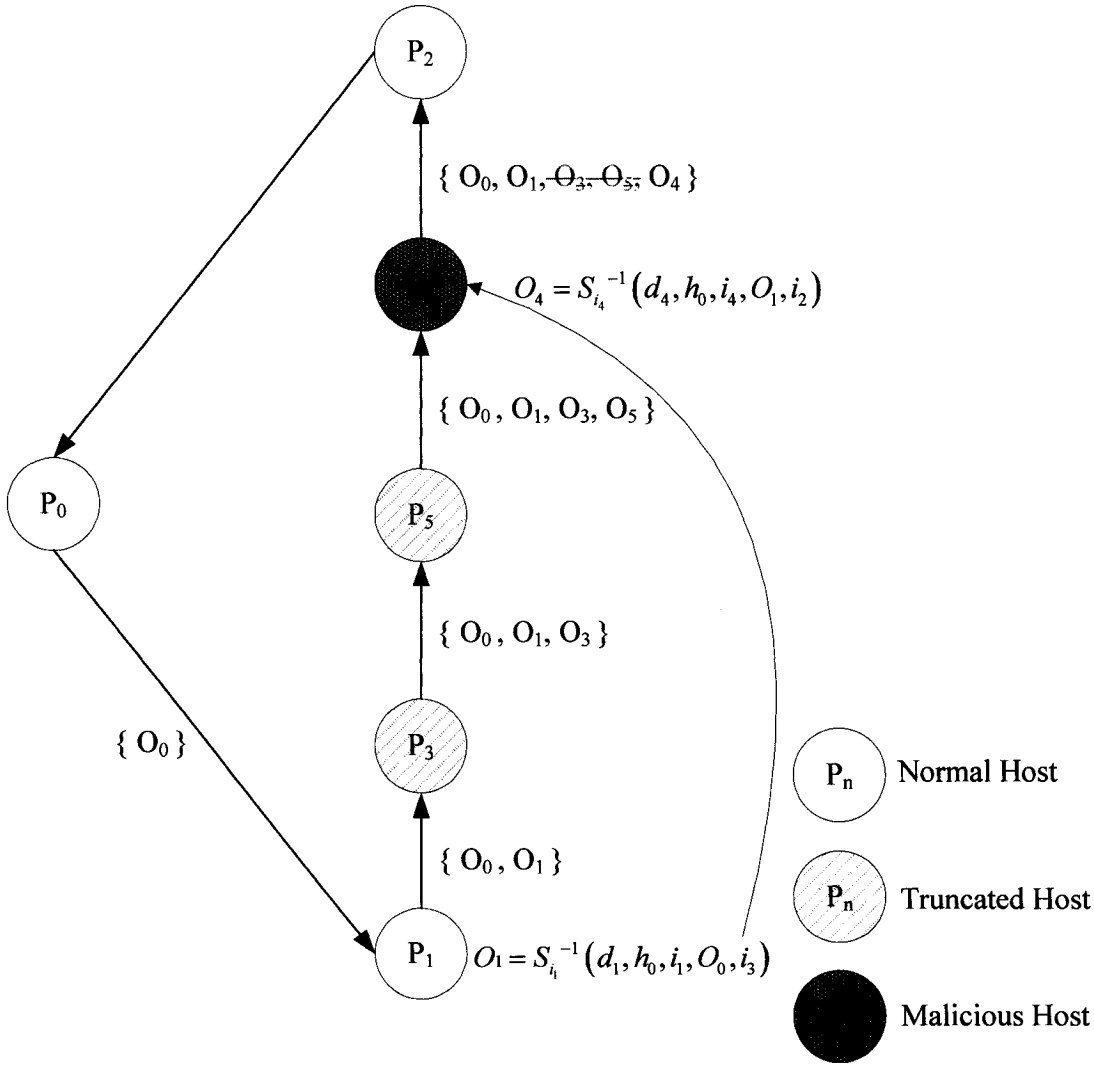

Figure 4.4: Deletion Attack

iii. Substitution Attack: A substitution attack is a deletion attack followed by the insertion of a data item different from the deleted one. For the reason described above, a single malicious host cannot mount a substitution attack without violating the chaining relation. If more than one host is malicious, the hosts can collude and mount a

substitution attack by truncating the offers of the hosts visited between them and then substituting fake offers (insertion attack), or by modifying the code of the agent, creating a new mobile agent, and sending the mobile agent to the same visited host, collecting the offers, and then appending these offers into the actual mobile agent in such a way that the chain of offers does not get violated. The execution tracing scheme also works in detecting the substitution attack. Again, on detection, all offers are discarded.

4.7 Truncation Protocol

A malicious host [44, 55] can mount several types of attacks on a mobile agent, such as altering the current state of the mobile agent, substituting the agent code in order to collect signed data which are then attached to the original agent code, and truncation of better offers provided by other hosts. Among all attacks [55], a truncation attack has been argued as the most difficult to solve for a free roaming mobile agent. Roth [68] pointed out that most of the mechanisms proposed in the literature are vulnerable to truncation attacks. In general terms [44], a truncation attack is the truncation of a host offer by one or more colluding malicious hosts. A truncation attack can be easily detected for a fixed itinerary mobile agent or a partially fixed mobile agent. Although a truncation attack mounted by a single host can be detected easily by establishing a chaining relation in a free roaming mobile agent, if two or more hosts collude and truncate the chain of offers between them, it is very difficult to detect all the hosts which have been truncated and to find the malicious hosts who have truncated offers from the chain of offers.

Extending the work of Yee [78] and Karjoth [44], Maggi and Sisto [55] proposed a generic protocol that could be configured according to security requirements and application. This protocol defends against most of the major attacks for free roaming mobile agents. Maggi and Sisto [55] argued that limited truncation resilience could be achieved by forcing the agent to securely store the addresses of the next hosts to be visited. In this study, we extend this work by proposing two traced based mechanisms which improve truncation resilience. These mechanisms are applied when the mobile agent returns after visiting all the hosts on list.

In the following section, we first present an overview of the generic protocol proposed by Maggi and Sisto, followed by a brief description of the security properties achieved by their protocol. Finally, an analysis of the protocol is done, with respect to truncation resilience, followed by the proposed extensions to the protocol, which improves truncation resilience.

$$M_n = D_n \| C_n$$

$$D_n = D(i_n, C_n)$$

$$C_n = C(i_n, d_n, C_{n-1}, i_{n+1})$$

$$C_{-1} = \in$$

$$\Pi_0 = \{\Pi, t\}_{S_{i_0}^{-1}}$$

$$i_n \rightarrow i_{n+1} : \Pi_0, \{M_0, ..., M_n\}$$

$$D(i_n, < d_n, p_n >) = \begin{cases} p_0 & \text{if } i_n = i_0 \\ < \{d_n\}_{K_{i_0}}, p_n > & \text{otherwise} \end{cases}$$

$$C(i_n, < d_n, p_n > C_{n-1}, i_{n+1}) = \begin{cases} S_{i_0}^{-1}(p_0, \Pi_0, i_1) & \text{if } i_n = i_0 \\ S_{i_n}^{-1}(< d_n, p_n >, \Pi_0, C_{n-1}, i_{n+1}) & \text{otherwise} \end{cases}$$

Maggi and Sisto's [55] protocol defends against most major attacks on free roaming mobile agents, providing limited truncation resilience. In their protocol, they showed how data confidentiality, data authenticity, non-repudiation, forward data integrity, and freedom from interleaving attacks could be achieved by using some cryptographic operations. For example, data confidentiality is achieved by encrypting data offer d_n using the originator's public key (K_{i0}); the forward integrity property is achieved by establishing a chaining relation between the previous collected signed offer (C_{n-1}), the current offer d_n, and the identity of the next host (i_{n+1}); data authenticity and non-repudiation are achieved by digitally signing the data d_n and the static piece of data Π_0 (Π_0 is the digitally signed static code provided by the origin host) by host i_n. This protocol avoids interleaving attacks by asking the host to sign the data d_n with the agent static piece of data Π_0. This ensures that the

mobile agent having static signed code Π_0 has really been executed on i_n, producing the offer d_n. This protocol [55] achieves a weak form of truncation resilience by forcing the agent to securely store the addresses of the hosts with the offer. In practice, on each visited host i_n the agent gathers two pieces of data: a set P_n of hosts to be visited (eventually empty if the visited host does not want to add new hosts to the agent's itinerary) and the actual data d_n (eventually a dummy value if host i_n does not want to or is not able to provide data).

4.7.1 Truncation Resilience in the Original Protocol

In this section, the protocol is analyzed with respect to the truncation resilience achieved for a free roaming mobile agent. As stated above, only a weak form of truncation resilience is achieved in Maggi and Sisto's protocol [55]. For example, let $< P_0, P_1, P_3, P_5, P_9, P_7 >$ be the initial itinerary list (the list of hosts when the agent is dispatched from the origin host P_0). The itinerary list is the list of hosts the agent should visit during its itinerary. The order or route in which these hosts are visited is not fixed. The agent is free roaming and can freely choose which host it will visit next from all the hosts on the itinerary list. Let's say host P_1 adds $<P_4, P_8 >$ to the itinerary list of the agent. The itinerary list thus becomes $< P_0, P_1, P_3, P_5, P_9, P_7, P_4, P_8 >$. After visiting host P_1, the agent moves to host P_3, then to host P_4 and then P_5, as shown in Figure 4.5. Host P_5 adds one host P_{12} to the itinerary list of the agent. Now, the itinerary list is $< P_0, P_1, P_3, P_5, P_9, P_7, P_2, P_4, P_{12}>$. In this way, the agent visits all the hosts on its list, and when it finally arrives at the origin host the itinerary list looks like $< P_0, P_1, P_3, P_5, P_9, P_7, P_4, P_8, P_{12}, P_{14}, P_{15} >$ (with the addition of five hosts during agent itinerary). The route taken by the mobile agent is illustrated in Figure 4.5.

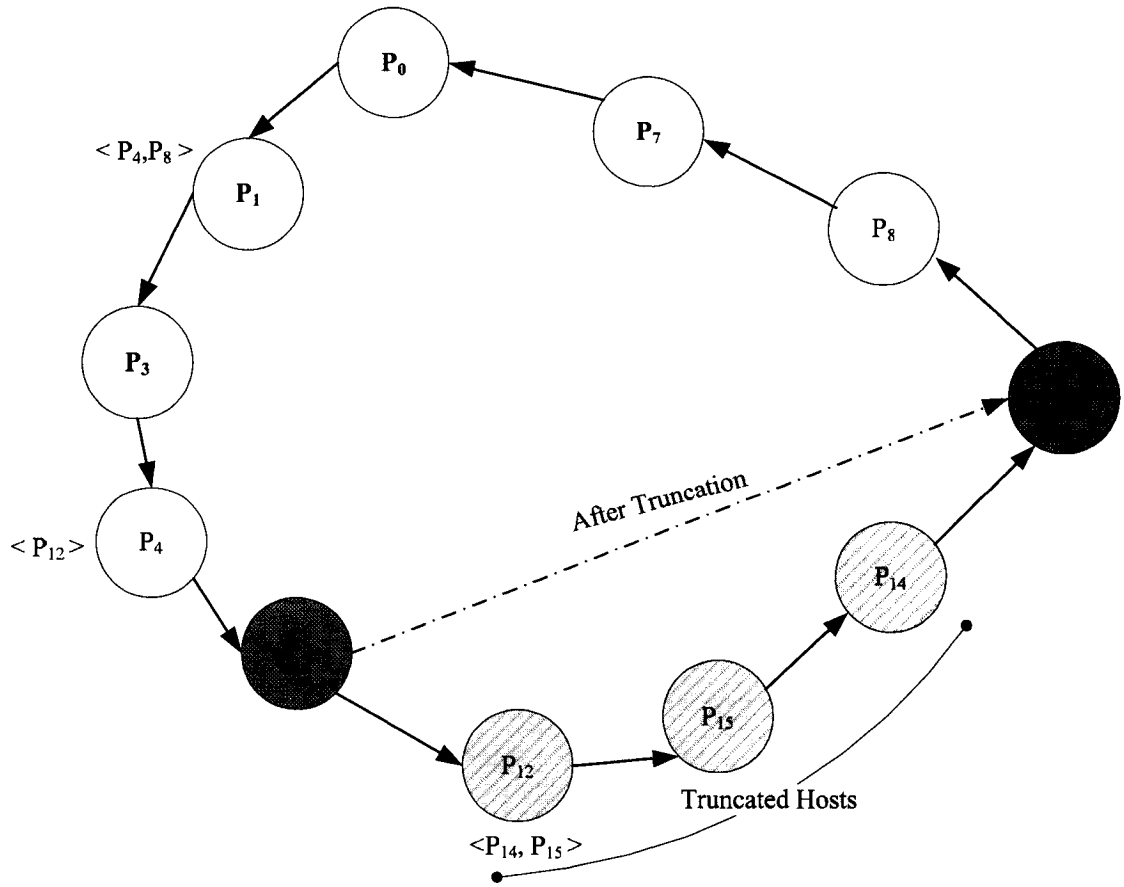

Figure 4.5: Agent route before truncation and after truncation

When the user agent returns to the origin host, the monitoring agent checks whether or not every host in the itinerary list has given an offer. If the monitoring agent host finds an offer for every host on the list, it concludes that no host has been truncated and thus, no malicious host has been found in the itinerary list. The monitoring agent then starts the evaluation of offers and selects the optimal offer.

If a host offer is missing, the monitoring agent suspects truncation of host offers. From Figure 4.5, it can be observed that host P_1 has added P_4 and P_8, host P_4 has added P_{12}, and P_{12} has added P_{14}, P_{15}. If two of these hosts are malicious and if they collude, they can truncate the list of the hosts visited between them (simply by saving the state when the mobile agent visits the first malicious host and restoring the same state when it visits the second malicious host). This again establishes the chaining relation between the hosts. If no

proper truncation detection mechanism is in place, the monitoring agent will not detect all the hosts which have been truncated. In Figure 4.5, hosts P_3 and P_9 are malicious and they truncate the hosts visited in between them (they truncate the offers provided by hosts P_{12}, P_{14}, and P_{15}).

Maggi and Sisto's protocol achieves a weak form of truncation resilience by forcing the agent to securely store the addresses of the next hosts to be visited. For example, when host P_5 adds P_{12}, the host P_5 supplies the address of host P_{12} with its offer. In the same way, host P_1 adds P_4 and P_8, and P_{12} adds P_{14} and P_{15} with their offers. In the protocol [55], the monitoring agent can only detect the truncation of host P_{12} but not of host P_{14} and P_{15}, because host P_{12} was added by host P_5, which stored the address of P_{12} while adding the host to the itinerary list. Therefore, when it returns, the monitoring agent can easily observe that the offer from P_{12} is missing. If the offer from P_{12} is truncated, the monitoring agent will not know whether host P_{12} has added any hosts or not. Therefore, it will not be able to detect the truncation of P_{14} and P_{15}.

4.7.2 Proposed Extension

Our approach is based on two assumptions: the mobile agent must visit all hosts on its itinerary list, and the mobile agent should not visit the same host twice in its journey. Our mechanism is used when the mobile agent returns after visiting all the hosts on the list. The monitoring agent examines all the results of the execution. First, it checks whether or not all the trusted hosts which were in the original itinerary list have been visited (have provided offers). It then starts examining all the hosts added by the trusted hosts (hosts which were in the list when agent starts its journey) during the itinerary (in this example P_4, P_8, and P_{12}). Then, it examines the hosts added by P_4, P_8, and P_{12} (hosts P_{14} and P_{15}). This process goes on until all the hosts in the itinerary are examined. If the monitoring agent observes that any of these hosts have not been visited, it starts the execution tracing mechanism; otherwise it stops and concludes that no host has been malicious and that all hosts have been visited by the mobile agent.

During execution tracing, the monitoring agent picks all the hosts suspected of not having been visited or having been truncated by malicious hosts and sends a common message to them. All the hosts respond with the trace of the execution of the mobile agent and with a list of hosts added by them (if any). This execution trace specifically includes the list of host added by them (if any), the offer they made and the identity of the next host where it sent the agent after execution. After receiving the trace from all the hosts, the monitoring agent checks whether any host has been added by them. If it finds any other hosts have been added by these hosts, it re-sends the common message to the newly added hosts. These hosts also respond with the trace of the execution of the mobile agent. After receiving the trace from these hosts, the monitoring agent re-checks whether any host has been added by them. If it finds any other hosts have been added by these hosts, it re-sends the common message to the newly added hosts added by them. This process goes on until all the hosts added directly or indirectly by the hosts are traced or until the monitoring agent gets the complete list of the truncated hosts. For example, in Figure 4.5, the monitoring agent first sends a message to P_{12} to check if any host has been added by it. The host P_{12} responds with its offer and also the list of hosts added by it (P_{14}, and P_{15},). After getting a list of hosts, the monitoring agent sends the common message to P_{14} and P_{15} to check if they have added any hosts to the list. After getting responses from both of these hosts, it observes that all hosts have been tracked down and no other host has been detected or truncated. This way, it concludes that a total of three hosts have been truncated, from the chain of offers.

This mechanism also enables users to identify the malicious hosts which have truncated offers from the chain. In our protocol, every host supplies four types of information with its offer, namely, the actual data offer d_n, the list of hosts added by it P_n, the previous signed offer (C_{n-1}, the offer from the host visited before arriving at the current host), and the next host address (i_{n+1}, the host that the agent is to visit next). After finding all the truncated hosts, the monitoring agent establishes a chain by combining C_{n-1} and i_{n+1} received from each host offer. Then, it determines where the chain has been truncated. Subsequently, it matches the last and the first host of the chain with the chain it receives from the mobile agent. For example, after finding hosts P_{12}, P_{14}, P_{15}, it checks the data provided by them. The data from P_{12} includes the offer, the signed offer of the previous host (in our example, host P_4), the hosts added by it (P_{14}, P_{15}), and the next host address (P_{14}). From this data, it is

clear that the agent visited P_4 before visiting P_{12}, and it will next visit P_{14}. In the same way, after looking at the offer of P_{14}, the monitoring agent gets the information that the mobile agent visited P_{12} before visiting P_{14} and it visited P_{15} after visiting P_{14}. In this way, all offers are examined (in our example, P_{12}, P_{14}, P_{15}) and the chain is established. In our example, the chain looks like $P_4 \rightarrow P_{12} \rightarrow P_{14} \rightarrow P_{15} \rightarrow P_9$.

After comparing the chain it receives from the agent (Figure 4.5) and the above chain established for truncated hosts, the user concludes that hosts P_4 and P_9 are malicious. The monitoring agent enters this record into the history module to avoid any further visits of the agent to these hosts. The main advantages of this mechanism are improved truncation resilience and identification of malicious hosts by observing the chain and finding where the chain was broken. On the other hand, its drawbacks are increased computational cost and forcing the remote host to store the trace of execution and send the execution trace if asked by the monitoring agent during execution tracing.

4.7.3 Tree based tracing mechanism

This mechanism is used when the mobile agent returns after visiting all the hosts on the list. The monitoring agent examines all the results of the execution. First, it checks whether or not all the trusted hosts which were on the original itinerary list have been visited (have provided offers). It then starts examining all the hosts added by the trusted hosts (hosts which were on the itinerary list, when agent started its journey) during the itinerary. If it finds that any of the hosts' (trusted or untrusted) offers are missing, then it suspects a truncation attack and sends a message to the trusted hosts to know the complete list of the hosts added directly or indirectly by them.

This mechanism follows tree based tracing, as shown in Figure 4.7, in which the root node sends a message to all its children, these children send messages to their children, and this continues until the leaf node is reached. Then they follow backward processing, that is, the leaf node gives some data to its immediate parent node. That parent node supplies the collected data and its data to its immediate parent node. This backward processing goes on

until the root node is reached. This way, the root node holds all the data from all the nodes in the tree.

In our mechanism, the monitoring agent acts as the root node and sends a message to the trusted hosts who were on the itinerary list when the agent started its journey and who have added hosts to the list of the agent. After getting the tracing message from the monitoring agent, each trusted host sends a message to the hosts added by them (untrusted hosts). If these untrusted hosts have added any other hosts to the list, they send a message to these hosts. If the receiving hosts have added any hosts to the list, they send a message to the hosts added by them; otherwise, they stop. This process goes on until the last host (the leaf node, which has not added any host) is reached. Now, the backward processing starts, that is, the last node in the tree responds with the trace of the mobile agent execution to the immediate parent host (the host who added it to the list). This execution trace includes the list of host added by them (if any), the offer they made, and the previous offers the host got during backtracking and the identity of the next host where it sent the agent after execution. Now, this parent host gives the collected trace to its immediate parent host. This processing continues until the root host is reached (i.e., the monitoring agent). Eventually, the root host gets the complete list of hosts. The main advantage of this approach over execution tracing mechanism is less message computation cost on the origin host. The drawback of this mechanism is dependency on host's platform.

The agent is free roaming, and thus can freely choose which host it will visit next from all the hosts on the itinerary list. Let's say <P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_7> is the initial itinerary list and when the agent visits host P_2, it adds <P_8, P_9 > to the itinerary list of agent. The itinerary list becomes <P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9>. After visiting host P_2, the agent moves to the other hosts on the list. When the agent visits host P_4, it adds host P_{10} to the itinerary list. Now, the itinerary list becomes <P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}>. In this way, the agent visits all the hosts in its list. Finally, when it arrives at the origin host, the itinerary list looks like <P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}, P_{12}, P_{13}, P_{14}, P_{15}>. There has been an addition of 8 hosts in total (hosts P_8, P_9, P_{10}, P_{11}, P_{12}, P_{13}, P_{14}, P_{15}) to the itinerary list, as shown in Figure 4.6. The figure above shows the order of visits of the mobile agent to

all these hosts. The mobile agent starts from P_0, moves to P_1 then moves to host P_2, and so on, as shown in the Figure 4.6. From the figure it is clear that host P_2 added P_8 and P_9, host P_4 added P_{10}, host P_8 added P_{11}, host P_{11} added P_{12}, P_{10} added P_{13}, and P_3 added P_{14}. When the agent visits all the hosts on its itinerary, each host supplies the offer it made (eventually a dummy offer is supplied if no offer is made). In this example, as shown in Figure 4.6, hosts P_7 and P_{12} are malicious; they truncate the offers of the hosts visited between them and again establish a chain from host P_7 to P_{12}. Thus, the offers of hosts P_4, P_9, P_8, P_{11}, P_5, P_{10} are truncated. When the agent returns to the origin host, the monitoring agent examines all offers and sees if any host offer is missing, by looking at the final itinerary list. If it finds that any host offer is missing, it suspects a truncation attack.

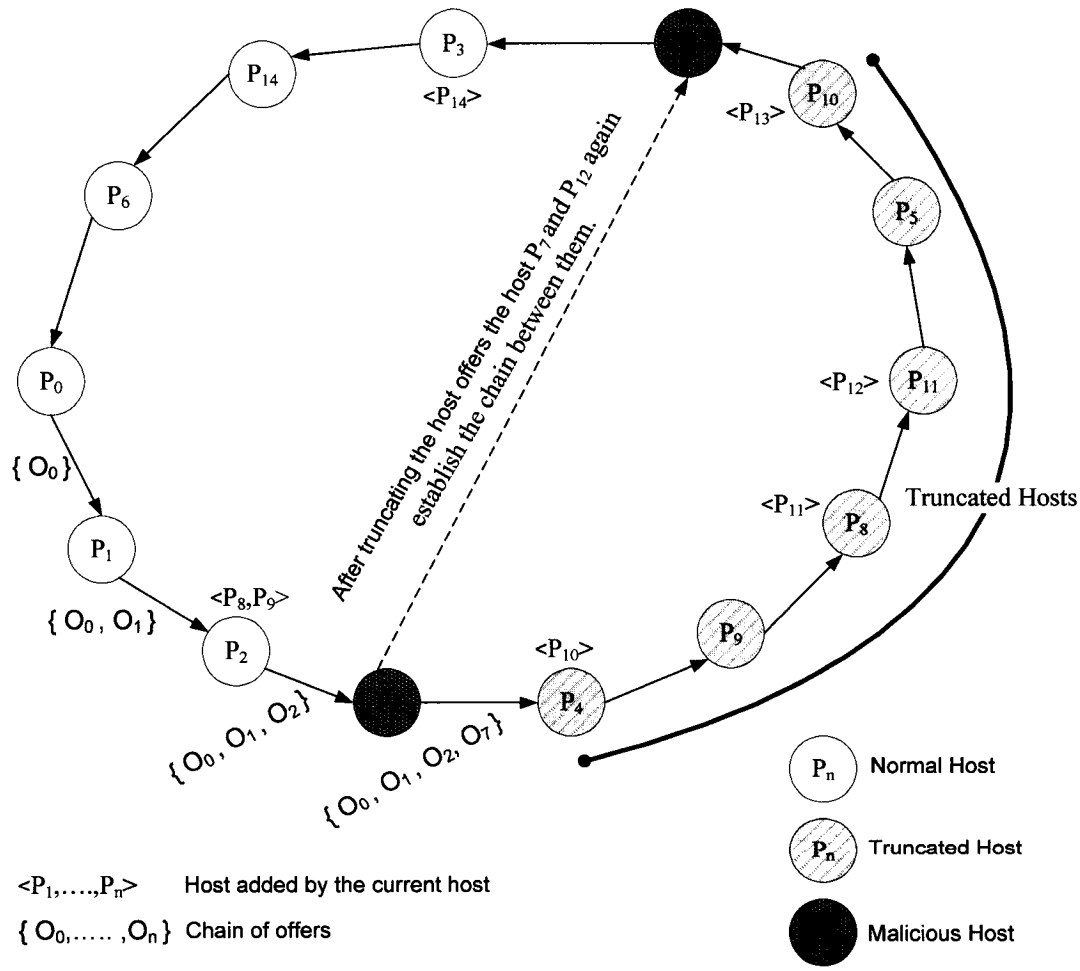

Figure 4.6: Truncation of host P_4, P_9, P_8, P_{11}, P_5, P_{10}

In this example, as shown in Figure 4.6, it finds that offers from the hosts P_4, P_9, P_8, P_{11}, P_5, and P_{10} are missing and it suspects a truncation attack and starts the tree based tracing. At first, it sees which of the trusted hosts (the hosts which were on the list when agent started its journey) have added other hosts to the itinerary list. In this example, it finds hosts P_2 and P_3. Then it looks to see if there is any trusted host whose offer is missing. In this example, it finds one, host P_4. After identifying these three hosts it sends a common message to them, asking for the complete list of hosts added by them directly or indirectly. These three hosts see if they have added any hosts to the list and send a common message to them if they have. In this example the hosts added by the three hosts are P_8, P_9, P_{10}, and P_{14}. Then, these four hosts send a message to hosts they have added, in this case P_{11} and P_{13} (hosts P_9 and P_{14} have not added any host). This process goes on until the last host who has not added any host to the list is reached, as shown in Figure 4.7. Then, the backtracking process starts, the last host responding with the trace to its immediate parent host and the parent host responding to its immediate parent. This process goes on until the root host is reached. As shown in Figure 4.7, hosts P_{12}, P_{13}, and P_{14} give traces (including the offer and hosts added, as well as their own ID's) to their respective immediate parent hosts (i.e., to hosts P_{11}, P_3 and P_4). Hosts P_{11}, P_3 and P_4 give traces to their immediate parent hosts and this backward processing goes on until the monitoring agent receives an offer from each trusted host to whom it sent the tracing message. After receiving all the messages, the monitoring agent examines all offers (as discussed in the previous section) and the list of host added by the trusted hosts and determines the total number of hosts truncated, from the chain of offers. This mechanism also enables the origin host to identify the malicious hosts which have truncated offers from the chain.

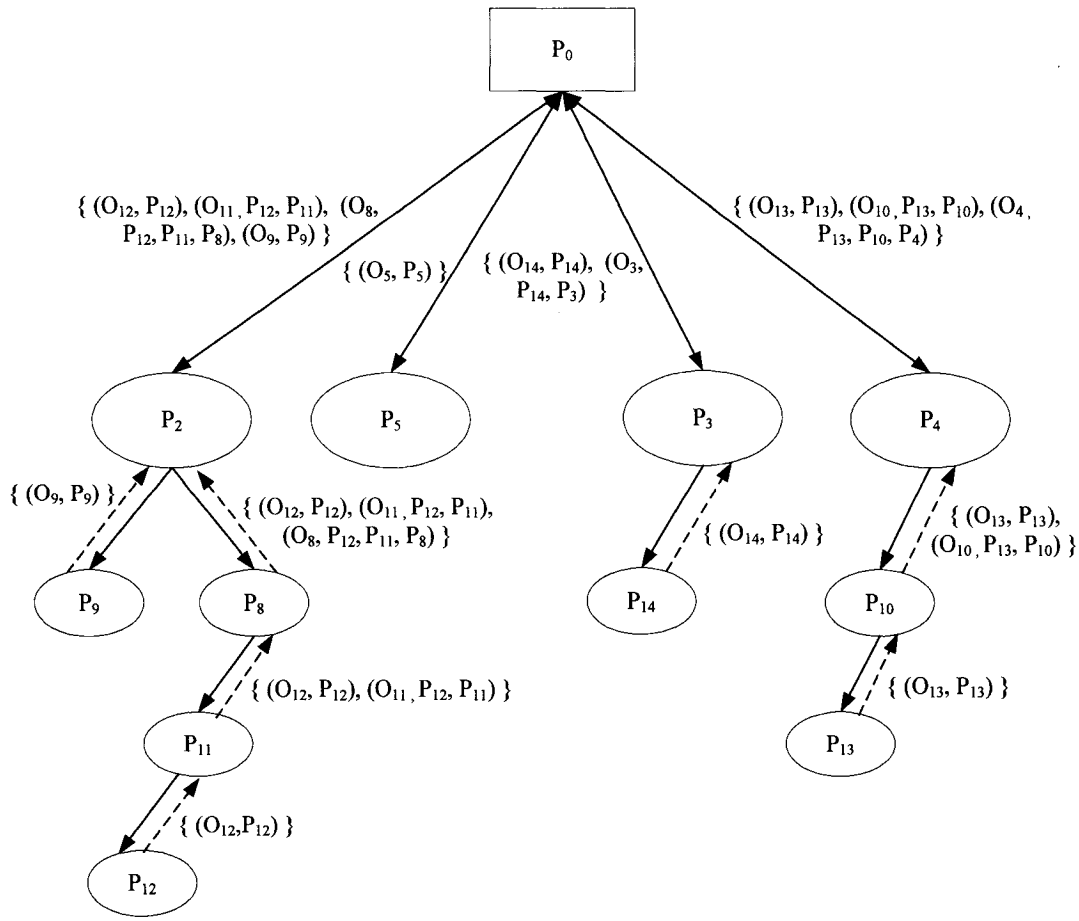

Figure 4.7: Tree based tracing mechanism

The number of message transmissions to detect truncated hosts depends on the number of trusted hosts who have added any other hosts to the list. In the above example, only six messages (sending and receiving) are required.

4.8 Summary

In this chapter, we presented a trace based security protocol to securely store the results of negotiation. This protocol works in two rounds. In the first round, the mobile agent collects signed agent trace offers at each host if negotiation is achieved; otherwise, it collects signed dummy trace offers. At the remote host, in the first round, the mobile agent can be in either of two states (negotiation state or token confirmation), depending on whether the agent has negotiated before with the host and whether the negotiation has been successful. The main advantage of including token confirmation in this protocol is the ability to avoid unnecessary

negotiation, thereby increasing the efficiency of the protocol. In the second round, after determining the optimal offer, the mobile agent is dispatched again to the selected host to confirm the negotiation. This protocol has been designed keeping in mind the desired security properties required, such as truncation resilience and data integrity, and also considering the other requirements, like resource constrained devices and bandwidth saving characteristics. In this protocol, instead of collecting the remote host's privacy data offer, the mobile agent collects the agent trace offer after every negotiation (if negotiation is achieved). This reduces the size of the mobile agent and thereby saves bandwidth over the network. The other advantage of storing the agent trace offer instead of the privacy data offer of the host is that it makes offer evaluation easier and faster at the origin host.

In this chapter, we also showed how the proposed protocol achieves desired security properties such as data confidentiality, data authenticity, non-repudiation, forward data integrity, freedom from interleaving attacks, and truncation resilience by using some cryptographic operations. Finally in the last section, we extended a protocol proposed by Maggi and Sisto [55] by proposing two mechanisms to improve truncation resilience, which is argued as one of the major problems with most of the mechanisms that have been proposed in the literature. These mechanisms also enable users to identify the malicious hosts which have truncated offers from the chain.

Chapter 5: Implementation

This chapter describes the implementation of the privacy negotiation architecture prototype. This prototype is used for negotiation over user preferences with the host privacy policy, and for securely store the offer provided by each host agent. In this architecture, three different types of agents are implemented, a monitoring agent, a mobile agent, and a host agent. The monitoring agent and host agent are static agents and cannot move to any other host, while the mobile agent can move from host to host. The monitoring agent resides on the origin host and acts as the master of the mobile agent. The host agent resides on the remote host, handles all incoming mobile agents and does the negotiation over the host privacy policy with these mobile agents. The mobile agent is the major entity of the architecture; it moves around and performs the negotiation over user preferences tasks on the user's behalf at each host.

The agents in this study are implemented using Java, Java-Security and IBM Aglets SDK. Aglets are a type of mobile agent with the ability to be transported from host to host. The agents are written in Java and can access databases at each host to get required information. To demonstrate the protocol, we will only consider flat files containing data which is stored in a delimited format. The aglets support message based communication; the agents in the architecture communicate with one another using an Aglet based messaging system.

5.1 Agents Description

The following section presents the brief description about the agents and their functionality in our architecture:

5.1.1 Host Agent

The host agent is implemented as a static agent and runs in a Java Virtual Machine (JVM) JDK 1.5.0_06 and Aglets 2.0.2 on a Windows XP machine. It resides on the host platform and handles all incoming mobile agents over the host platform. It utilizes an Aglets messaging framework to communicate with the mobile agents. It does negotiation with the mobile agents according to the privacy policy of the host.

5.1.2 Mobile Agent

The mobile agent runs in a Java Virtual Machine (JVM) JDK 1.5.0_06 and Aglets 2.0.2 on a Windows XP machine. It can move from host to host to do negotiation over user preferences. On creation, the mobile agent is initialized with a unique constant ID. This ID identifies the user and remains same for all negotiations pursued by the mobile agent.

After arriving at the host platform, it utilizes the Aglets messaging framework to communicate with the host agent and to commence negotiation. In the first round, if negotiation or token confirmation is achieved, the mobile agent collects the signed agent trace offer and travels to another host in its itinerary list. When the mobile agent returns to the origin host, the optimal offer is determined and mobile agent is dispatched again to the selected host to get the final confirmation of the negotiation and to collect the privacy data offer.

5.1.3 Monitoring Agent

The monitoring agent is implemented as a static agent and runs in a Java Virtual Machine (JVM) JDK 1.5.0_06 and Aglets 2.0.2 on a Windows XP machine. It acts as the master of the mobile agent. It utilizes the Aglets messaging framework to communicate with the mobile agent. It is created after the mobile agent is dispatched. When the mobile agent arrives at the origin host, the monitoring agent first identifies it and then collects the offers carried by it.

In the first round, when the agent returns to the origin host the monitoring agent collects all the agent trace offers generated at each host by the mobile agent. After collecting the agent trace offers, it validates the offers. Figure 5.1 shows the validation process. This is done simply by comparing the number of offers in the chain with the number of hosts in the itinerary list of the mobile agent when it was dispatched from the origin host. If it finds that any host offer is missing, it suspects a truncation attack and starts execution tracing (described in Chapter 4); otherwise, if it is verified that no host offers are missing from the chain, it checks the integrity and authenticity of each offer by verifying the digital signature

on it. This ensures that the result of the negotiation has not been tampered with and that the offer has been produced at the host who had signed the trace offer. After verification, the monitoring agent removes the offers of the hosts with whom negotiation has not been achieved (i.e., the hosts who have signed the dummy trace) and starts evaluating the remaining agent trace offers. After selecting the optimal offer, it provides this offer to the mobile agent, who is dispatched to collect the privacy data offer from the selected host.

In the second round, when the mobile agent returns to the origin host, the monitoring agent collects the privacy data offer. It checks the integrity and authenticity of the offer by verifying the digital signature on the offer, and then compares the data offer with the agent trace offer. If the privacy data offer matches the agent trace offer or if it is a subset agent trace offer, the negotiation is said to be successful.

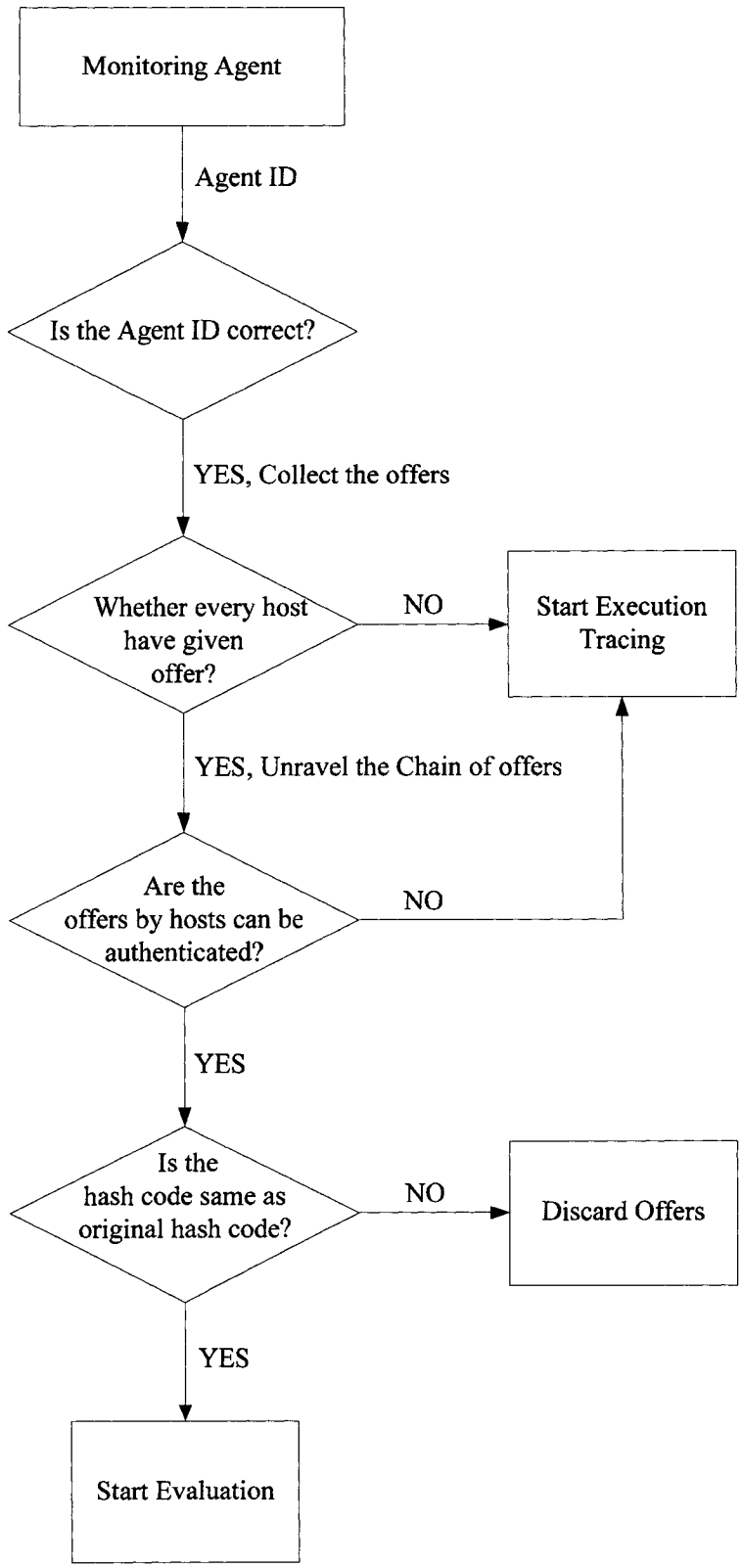

Figure 5.1: Monitoring Agent Validation Process

5.2 Implementation Description

The following section present the implementation description of the agent implemented in our architecture:

Eclipse 3.1.2 was the development environment of choice for the development of basic negotiation framework and an implementation of trace based security protocol. It is a very powerful integrated development environment (IDE). Figure 5.2 and Figure 5.3 shows the UML diagrams for the java classes and Java Methods implementing the mobile agent and stationery agent on the user end. Figure 5.4 and Figure 5.5 shows the UML diagram for the java classes and methods implementing the server agent on the remote host side.

The base of all the classes is the Aglet class. This is an abstract class which provides the basic functionalities required for the implementation of an agent. All three agents (i.e., mobile agent, stationery agent and host agent) extend the Aglet class. The main method of the Aglet is onCreation which allows custom initialization of the agent created. The User class extends the Aglet base class and implements the mobile agent functionalities. It uses keyload class to retrieve the keys from the keystore. The class Stationery Agent implements the functionalities of Stationery Agent. This is created after dispatching the mobile agent and used for validating and collecting the offers.

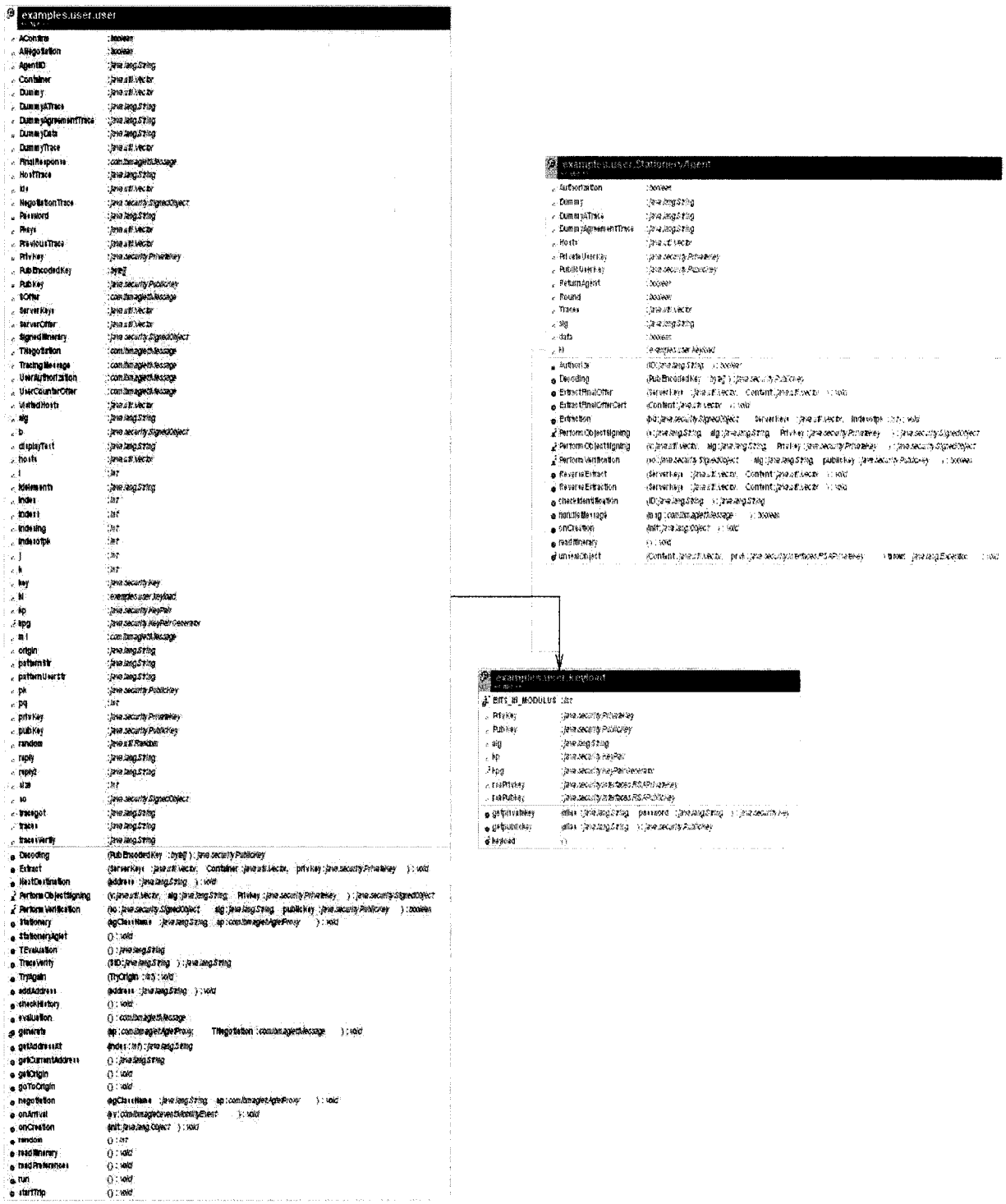

Figure 5.2: UML Class Diagram for User and Stationary Agent

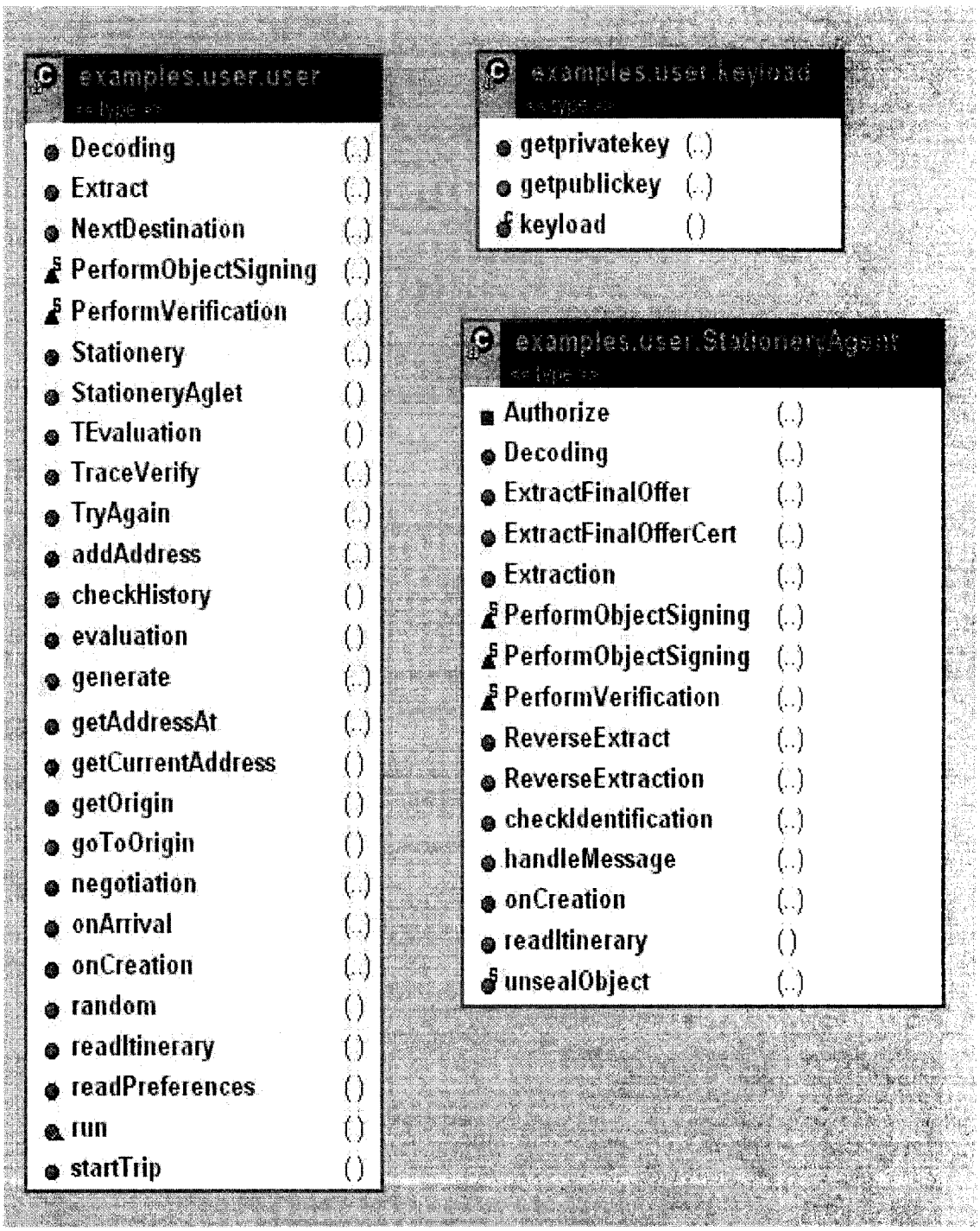

Figure 5.3: UML Method Diagram for User and Stationary Agent

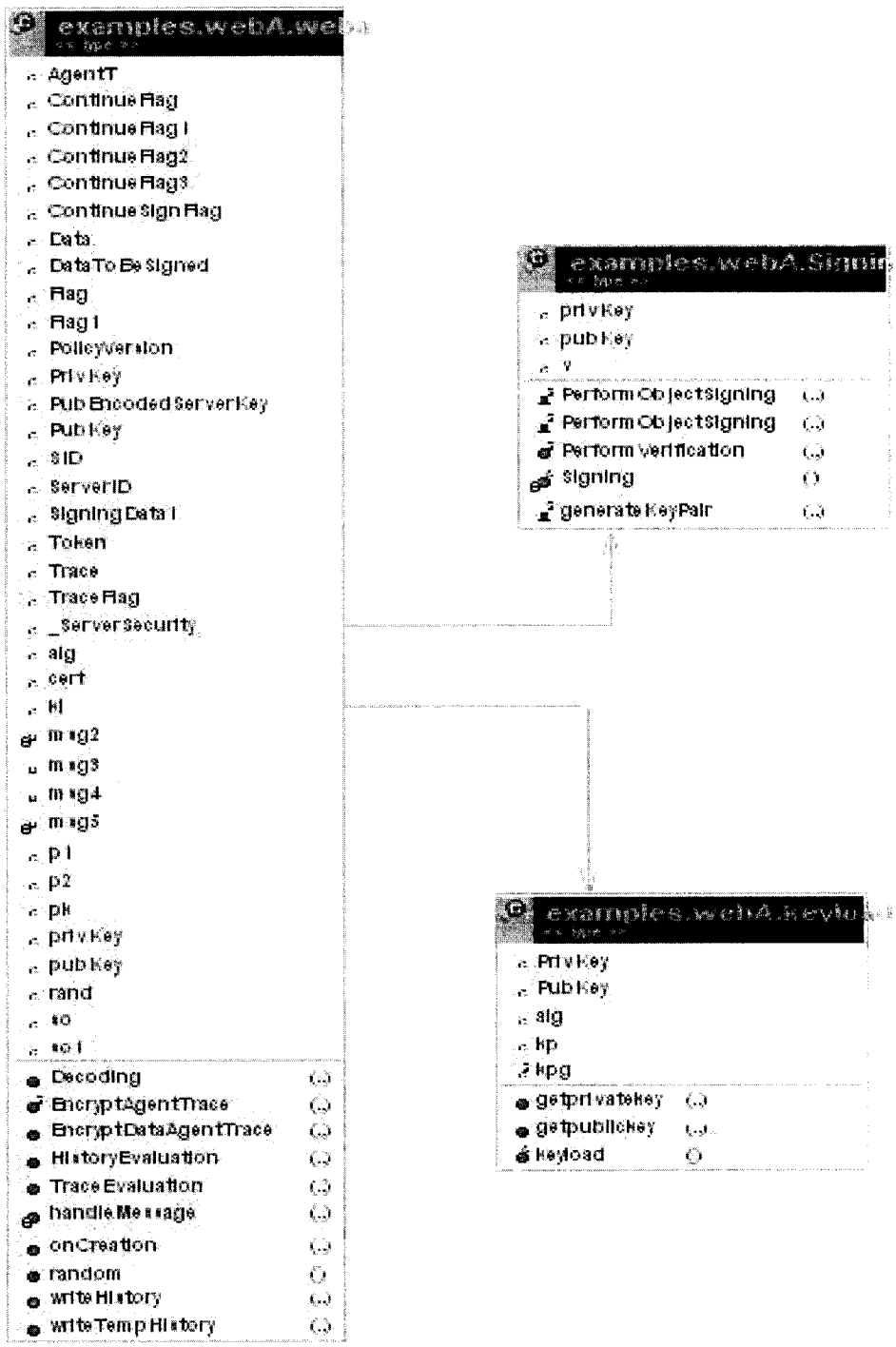

Figure 5.4: UML Class Diagram for Host Agent

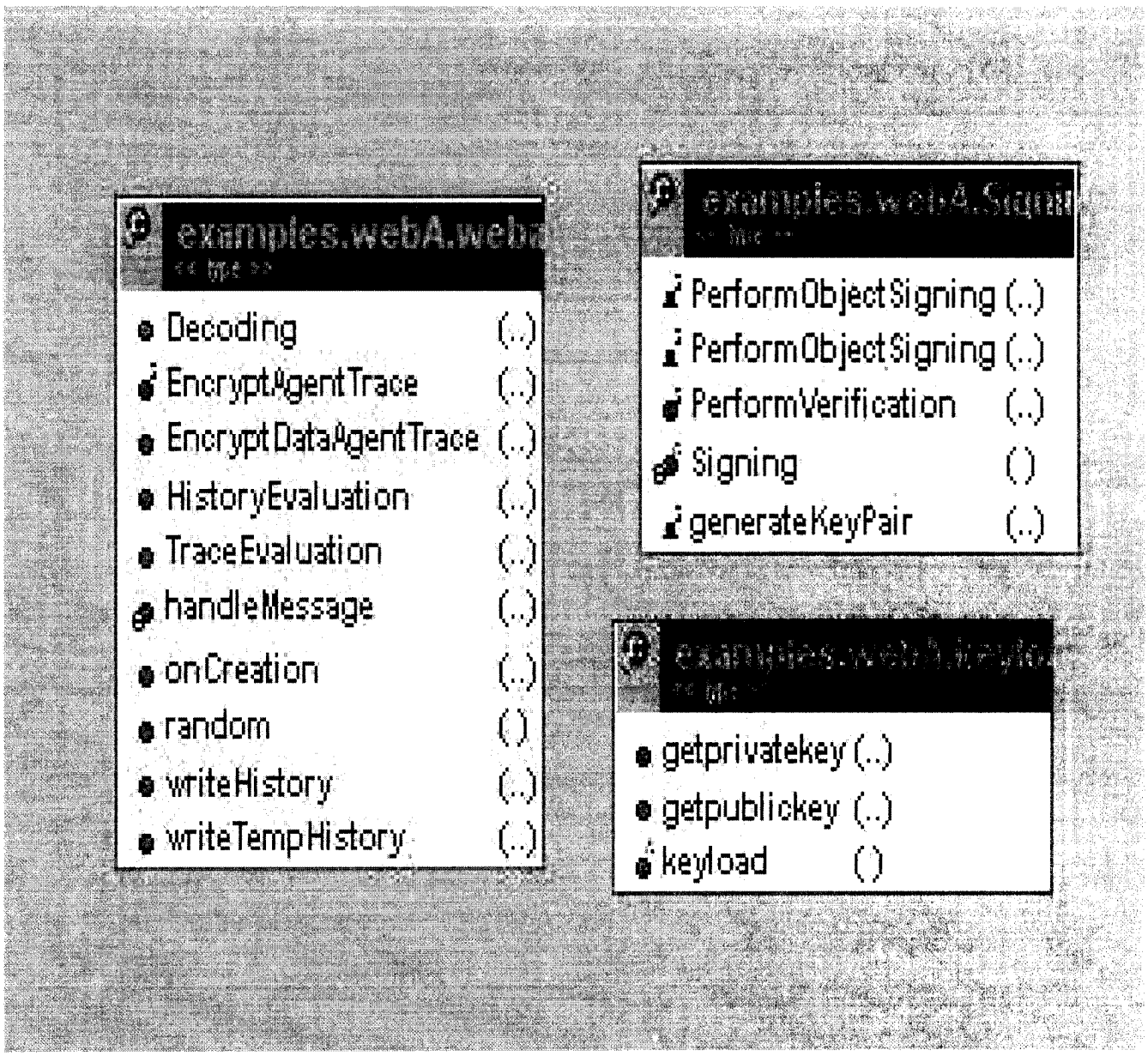

Figure 5.5: UML Method Diagram for Host Agent

In this section we describe important java methods used to implements the base functionality of the negotiation framework and a trace based security protocol using flow control diagram. First is the onCreation method which is the main method of all the agents. In case of mobile agent implementation, It at first initializes the agent with a unique ID. Figure 5.6 shows the flow control diagram of the onCreation method. Which starts with an initialization with a unique ID, then it retrieves the public and the private key from the Keyload class. After

retrieving the keys, it read the list of hosts to be visited by the agent from the itinerary file After getting the list of hosts, the agent sends the list to the check history methods which examines the list of host t determine if there is any hosts which has been malicious in the past negotition or there is any host with whom negotiation was successful. The Figure 5.7 shows the functionality of the Checkhistory function. After examining the list of host, the Checkhistory method supplies the final list of hosts to be visited with the token of the hosts with whom the negotiation was successful. After getting the final list of hosts, it read the preference from preference file which consists of hard coded preferences. After reading the preferences, it performs signing of the dummy offer to create a chain of encapsulated offers. Then it calls the starttrip method. This method checks the address of the next host to be visited and calls the dispatch method which dispatches the agent to the remote host. Figure 5.8 shows the functionality of the starttrip method.

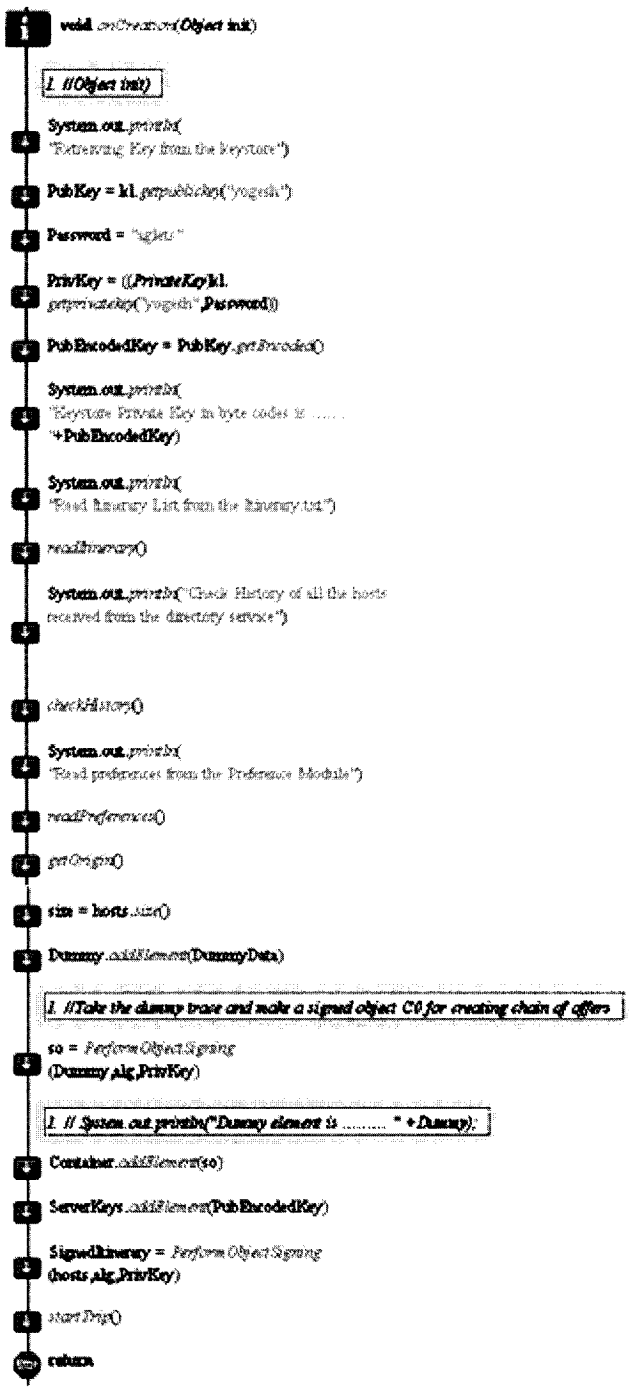

Figure 5.6: Flow control Diagram of onCreation Method of Mobile Agent

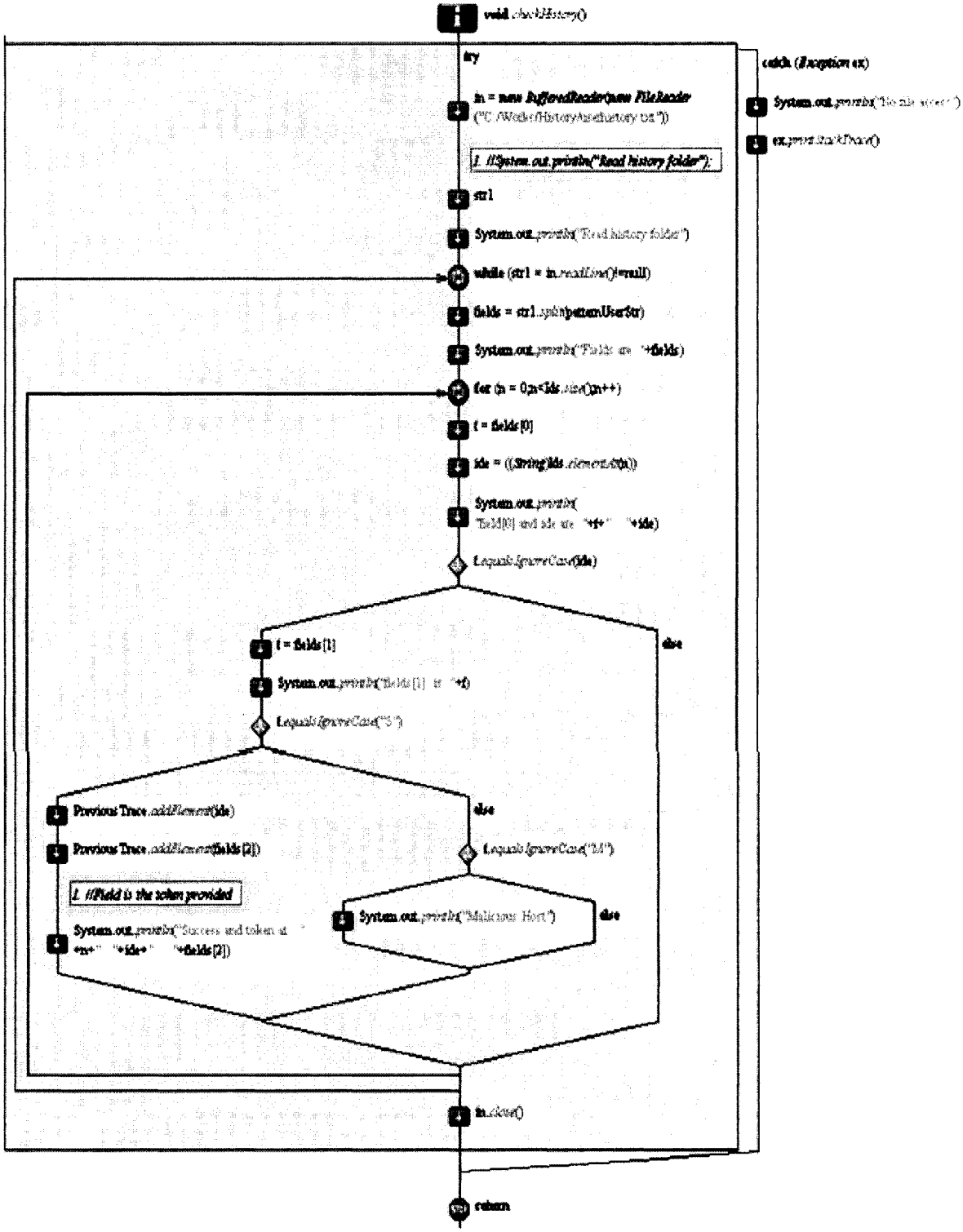

Figure 5.7: Flow Control Diagram of checkHistory Method of Mobile Agent

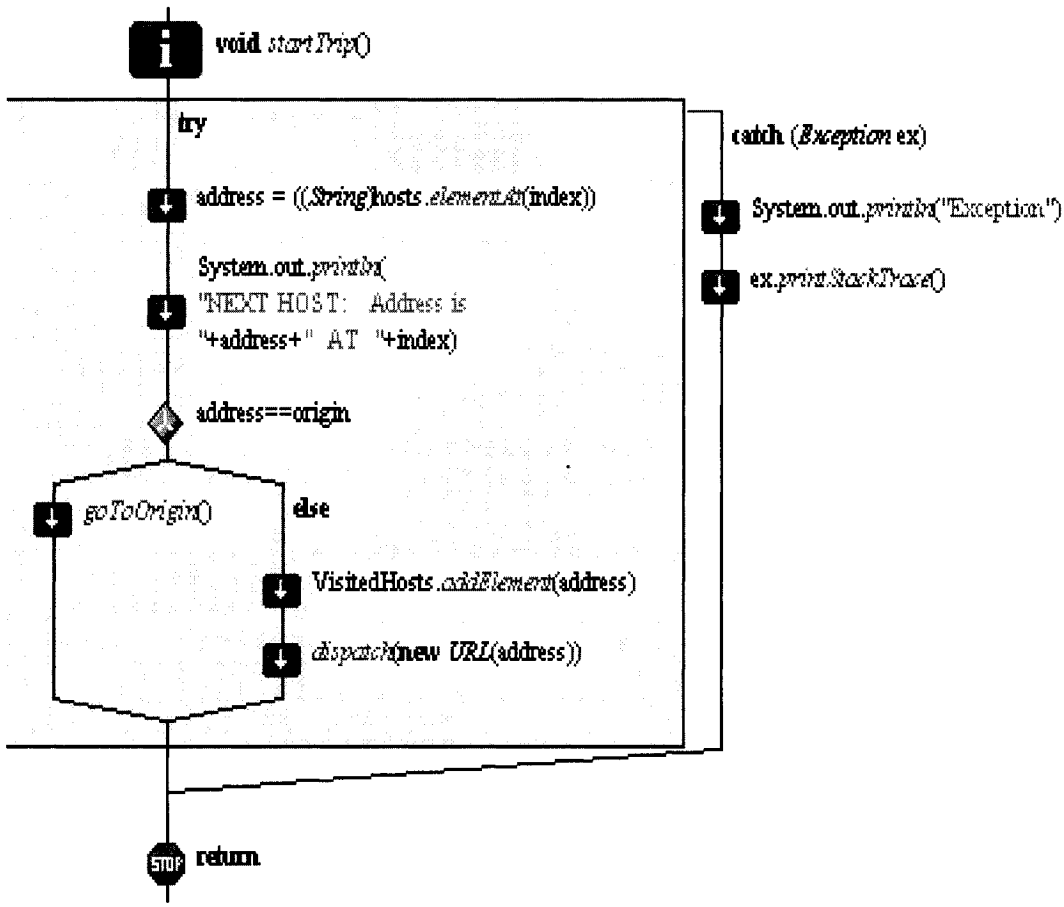

Figure 5.8: Flow control diagram of startTrip method of Mobile Agent

Upon arrival on the remote host, the mobile agent calls on onarrival method and starts running in a new thread from its starting point. It calls run method shown in the Figure 5.9. In run method mobile agent first checks the current host address, if it is same as origin address it calls stationary agent message where it gives all the collected offers to the stationary agent. Otherwise, it calls the negotiation method where it does negotiation with the remote host. Negotiation method covers all the scenarios described in section 5.4. After negotiation, it moves to generate methods where it takes the offer supplied by a remote host and request the host to sign. After appending the signed offer to the offer container it dispatches to the other remote host in the itinerary list. Upon arrival on the next remote host, the mobile agent calls on onArrival method and starts running in a new thread from its starting point. The same way it moves to all the remote hosts in the itinerary list and does negotiation and collects the result of the negotiation.

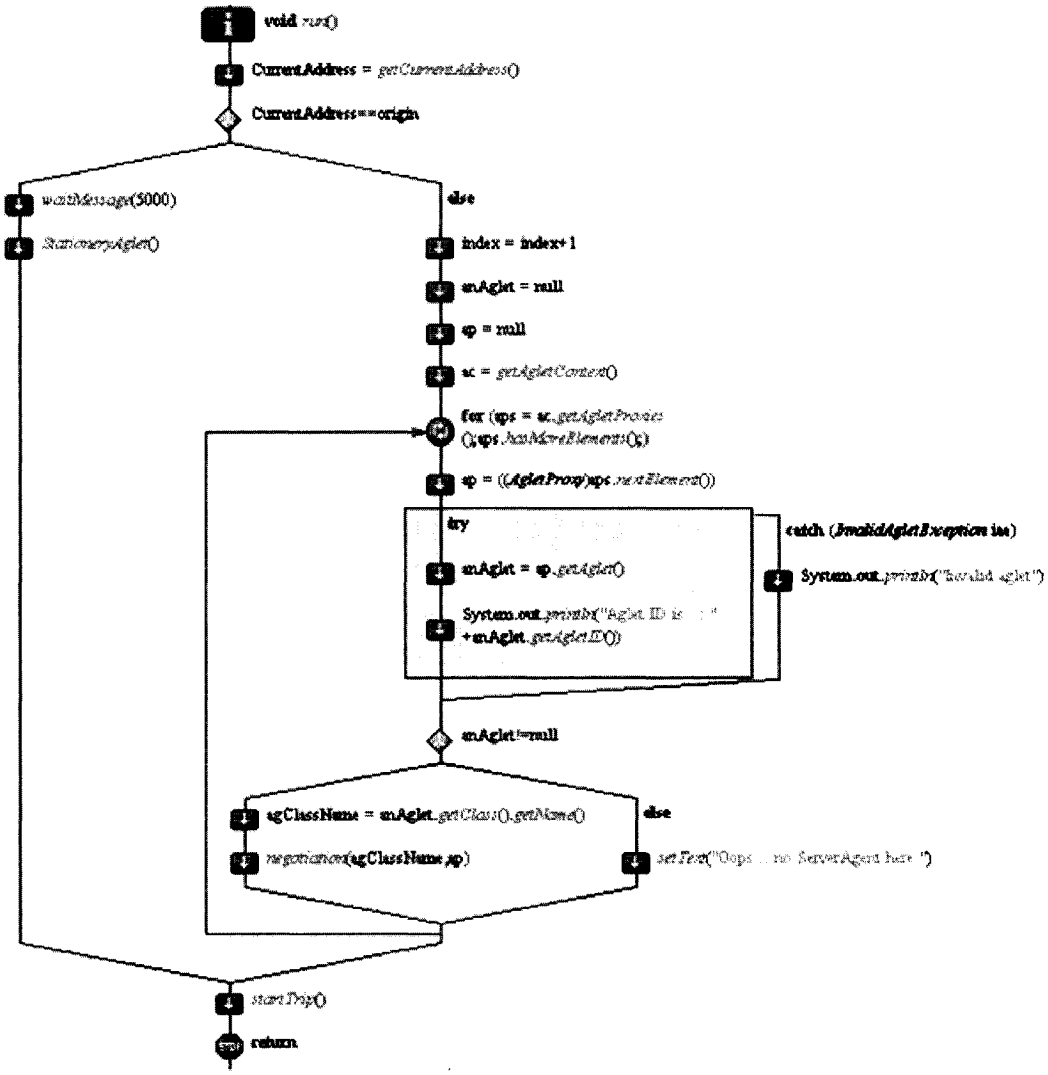

Figure 5.9: Flow control Diagram of run method of Mobile Agent

Figure 5.10 shows the Flow control diagram for the java method extract of the stationary agent which extracts the offered collected by the mobile agent during its first round of itinerary. Figure 5.11 shows the Flow control diagram for the java method extract of the stationary agent which extracts the offered collected by the mobile agent during its first round of itinerary

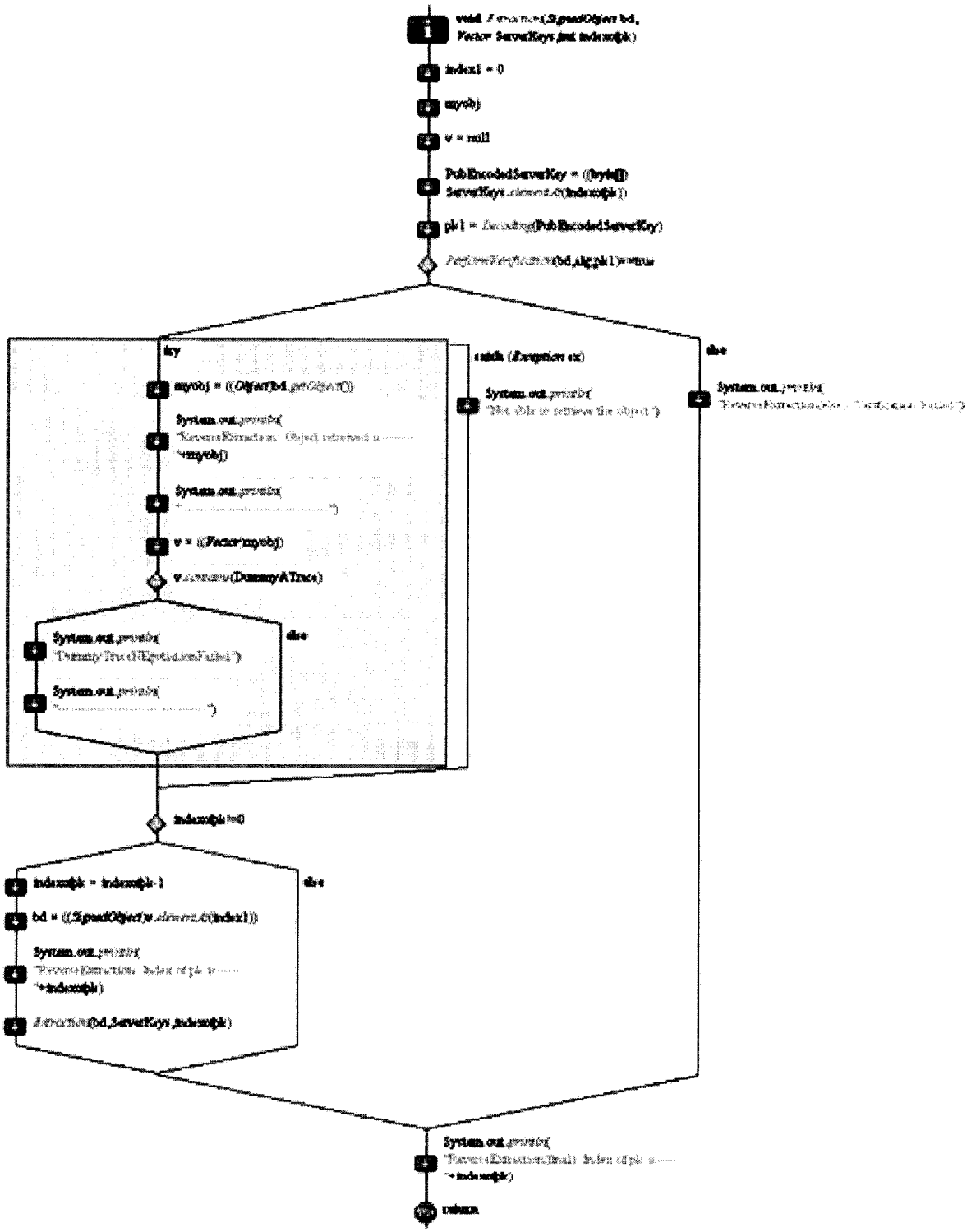

Figure 5.10: Flow control diagram of extraction method of Mobile Agent

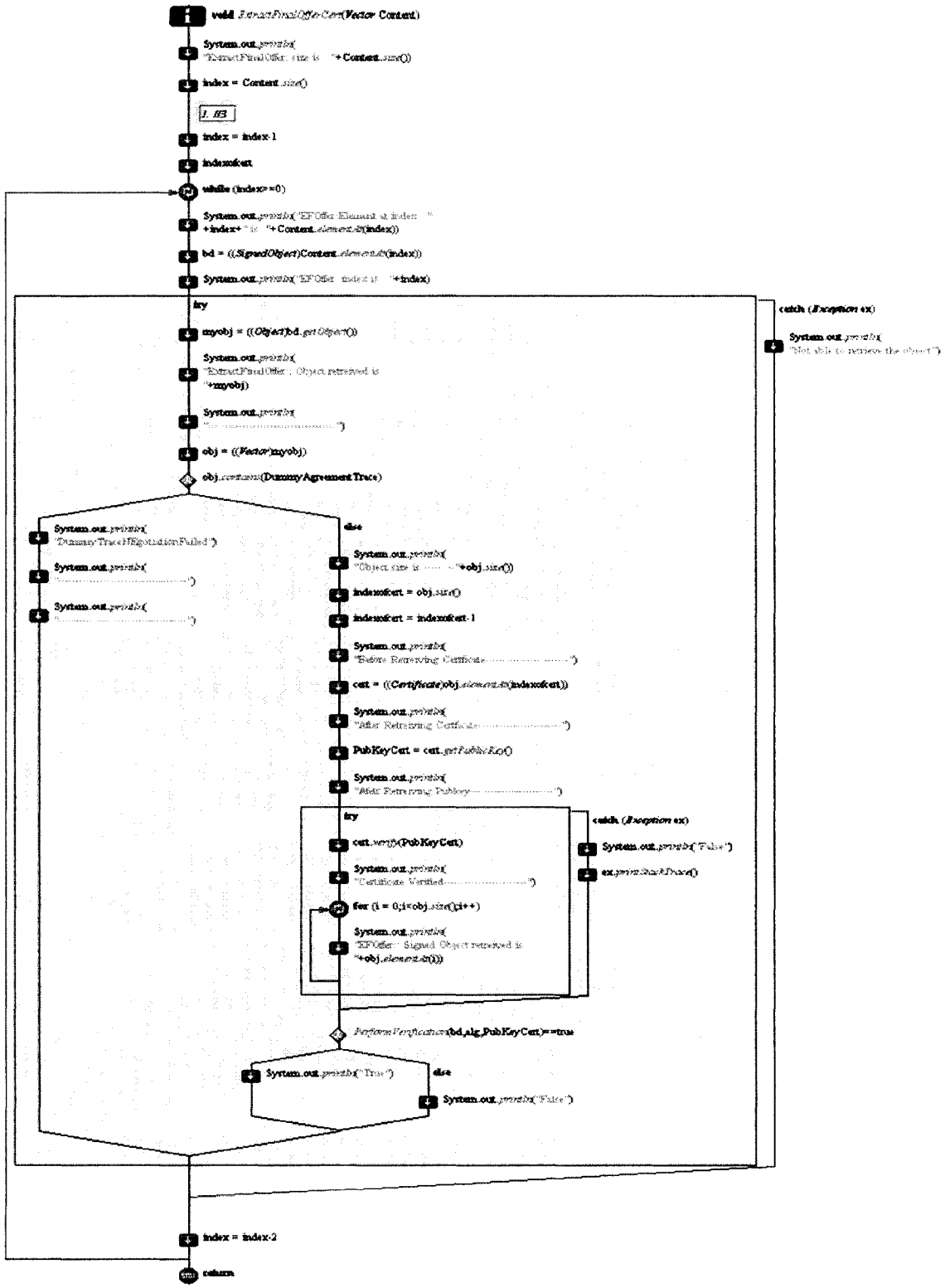

Figure 5.11: Flow control diagram of extraction method of Mobile Agent

Figure 5.12 indicate a sequence diagram representation of the mobile agent processing at origin host and remote host in the first round of its itinerary. This diagram does not shows the second round of the mobile agent on which mobile agent visits the selected host for agreement confirmation. As well token confirmation state is also not covered in this diagram.

Figure 5.12: Sequence diagram of a mobile agent at the origin host and at the remote host (Negotiation Phase)

Figure 5.13 indicate a sequence diagram representation of the mobile agent processing at origin host and remote host in the first round of its itinerary. This diagram shows the token confirmation pahse at the remote host. The user negotiation pahse has been shown in Figure 5.12. This diagram does not shows the second round of the mobile agent on which mobile agent visits the selected host for agreement confirmation.

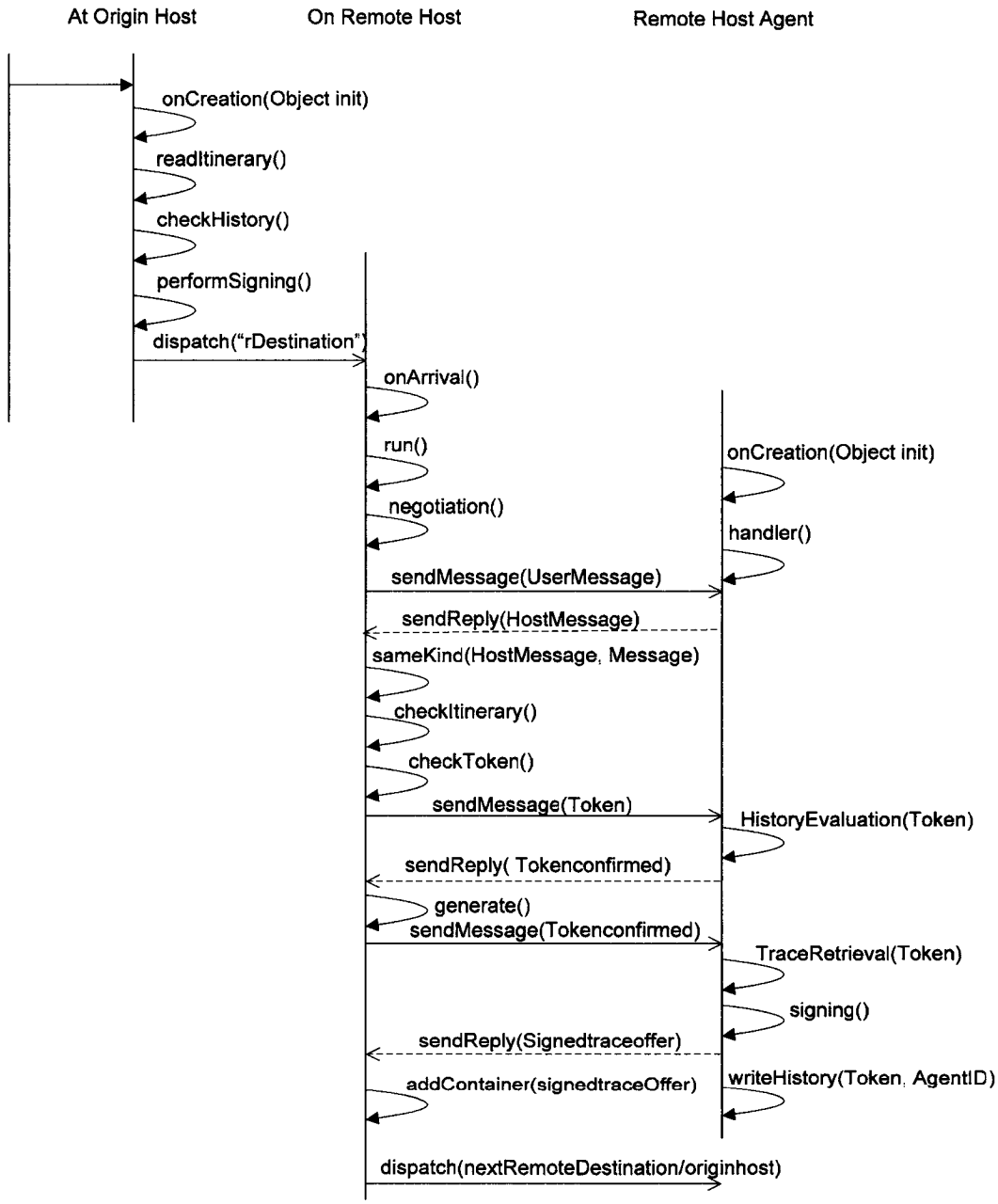

Figure 5.13: Sequence diagram of a mobile agent at the origin host and at the remote host (token confirmation phase)

5.3 Role of History Files

A simple text file is created on the origin host, called user history file. The user history module maintains this user history file, which stores the records of past negotiations of the mobile agent. The information in the history file is updated at two stages of the negotiation process, after the first round, when mobile agent returns to the origin host, and after the second round of the mobile agent. After the first round, the monitoring agent enters information for hosts which have been identified as malicious (if any). It enters the host ID and result of the negotiation (i.e. malicious) into the history file. In the second round, it enters the information for host with whom the negotiation has been successful. It enters the host ID, the result of the negotiation, the transaction identifier, the token value, and the signed privacy data offer, into the history folder. The user history module uses this information for two purposes: first, to filter the list of hosts to be visited by the mobile agent by removing hosts with whom past negotiation experience has been negative, and secondly to provide tokens to the mobile agent for the hosts with whom negotiations have been successful in the past.

On the remote host, two simple text files are created. These are the permanent history file and the temporary history file. The permanent history file is a simple file encompassing information about the previous successful negotiations of the host with mobile agents. The information in the permanent history file is mainly used for two reasons: firstly, to avoid unnecessary negotiation in the future with the same mobile agent, and secondly, to check and delete personal data from the permanent history file if the retention duration of any negotiation agreement is over.

The temporary file stores information for a short period of time. It stores basically four types of information: the agent ID, the agent trace offer, the privacy data offer of the host, and the retention duration. This information is updated in the temporary file if the negotiation is achieved between the mobile agent and the host. The host agent stores the agent trace offer and agent ID in the temporary history file to identify the mobile agent in the second round of negotiation (the agreement confirmation phase). During agreement confirmation, the host agent compares the agent trace offer and agent ID with the data in the temporary history file.

If it finds a match, it retrieves the actual privacy data offer from the temporary history file and digitally signs the data offer. If it does not find a match, it rejects the request for confirmation and the negotiation is terminated.

5.4 Communication between the Agents

Communication among the agents is implemented using a set of messages, utilizing the Aglets messaging framework. The Aglets messages can be synchronous, asynchronous, multi-cast, or single cast, and may or may not require a reply. The Aglets have three types of messages, namely, *Message*, *Future Reply* and *Reply Set*. We use the *Message* type to communicate between the agents (i.e., synchronous messaging where the sender waits for a reply and can only send one message at a time to only one agent). In the following section we present the important methods which are used during communication between the agents.

boolean handleMessage(Message)

This is a key method for receiving messages. It returns a Boolean value. A return of true indicates that the message has been dealt with. A return of false indicates that the agent does not understand the message.

boolean sameKind(String)

This method allows the receiving agent to distinguish among messages the aglet wants to understand and respond to. The argument is a message key to be compared with keys of incoming messages.

void sendReply (Object)

The easiest way to reply to a message is to use the Message object itself to carry a reply to the sender of the message.

Sample code showing the use of these three methods is shown below.

```
public boolean handleMessage (Message msg) {
 if (msg.sameKind("Negotiation Agent") )
```

```
        Message ServerM = new Message("PurposeofVisit");
        msg.sendReply(ServerM);
 } else if (msg.sameKind("AgreementConfirmation")){
        Message ServerMessage = new Message("AgreementConfirmed");
        msg.sendReply(ServerMessage);
} else {
   System.out.println("Invalid Agent");
}
```

5.5 Interaction of the Agents

From the Figure 5.14, the mobile agent starts its itinerary from the origin host and then visits all hosts on its itinerary list. At each host, it interacts with the host agent, does negotiation, and collects the result of the negotiation.

After visiting all the hosts in the list the mobile agent return to the origin host. In the first round, when mobile agent returns to the origin host, the mobile agent authenticates itself to the monitoring agent by providing required identification information. This information includes the agent ID, the agent static signed code, and dummy data the mobile agent received when it was dispatched from the origin host.

After authenticating the mobile agent, the monitoring agent collects the agent trace offers and validates every offer. After validation, the monitoring agent evaluates the agent trace offers and determines the optimal offer.

The number of host offers to be selected for agreement confirmation depends upon the evaluation methodology. In this study we assume that only one host is selected for agreement confirmation, but this protocol once implemented, can also be used if more than one host is selected for agreement confirmation. After selecting the host offer, the monitoring agent supplies the desired information (i.e., the selected agent trace offer and the host address) to the mobile agent to start the agreement confirmation phase. After receiving the information, the mobile agent is dispatched again to the selected host (in the Figure 5.14,

host 3 is selected) for agreement confirmation. At the selected host, the mobile agent asks the host agent to confirm the negotiation. If negotiation is confirmed, the negotiation offer is signed by the host agent; otherwise, a rejection message is provided by the host agent. After agreement confirmation, when mobile agent returns to the origin host, the mobile agent again authenticates itself to the monitoring agent by providing required identification information. This includes the agent ID, the agent static signed code, and agent trace offer.

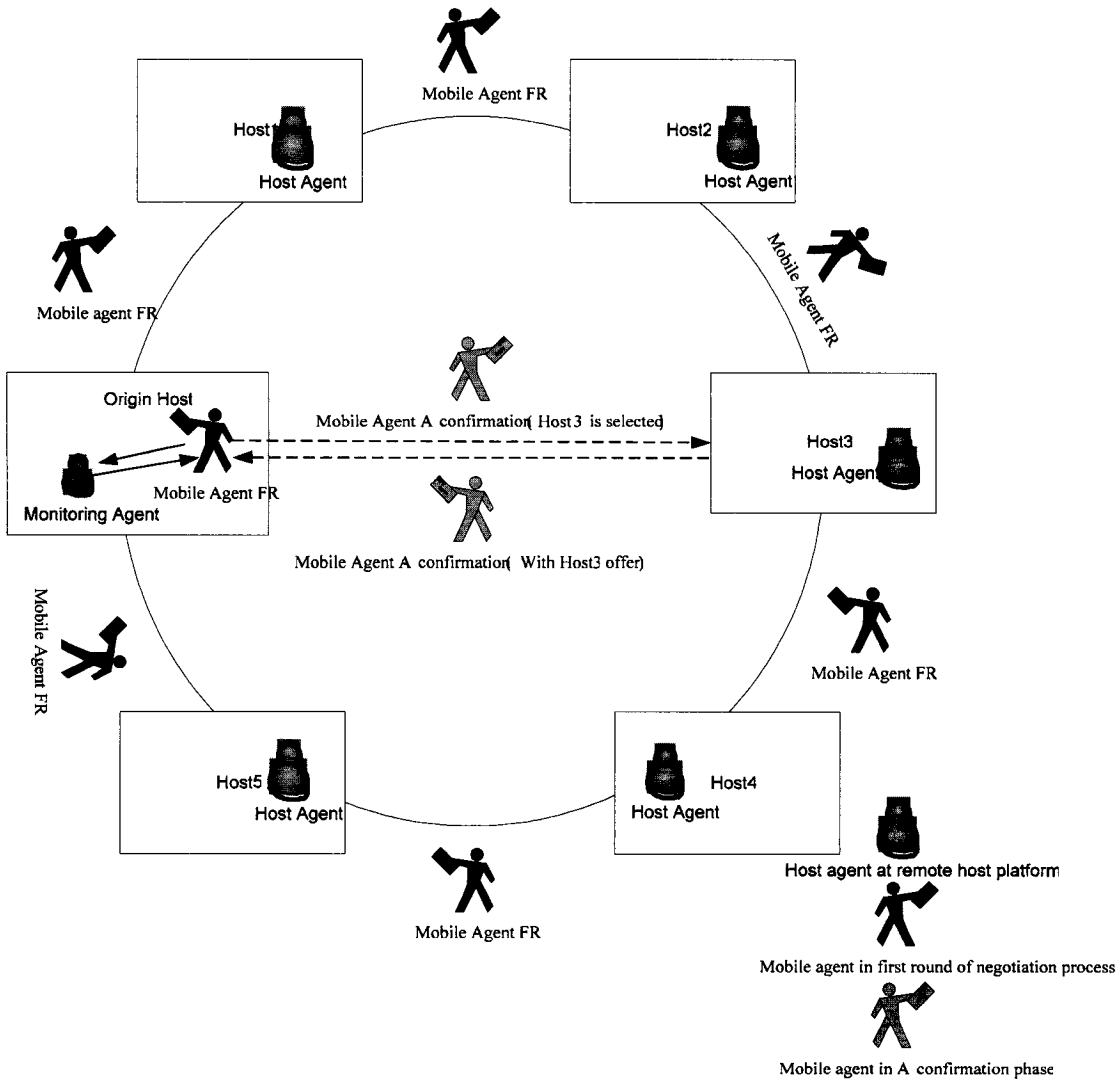

Figure 5.14: Interaction of Mobile Agent, Host Agent, and Monitoring Agent

After authenticating, the monitoring agent collects the signed privacy data offer. The monitoring agent then verifies the digital signature over the privacy data offer and retrieves the privacy data offer. If negotiation has been confirmed, and matches (or subset) the user

preferences and the agent trace provided by the host agent in the first round of the negotiation. If these are validated the negotiation is said to be successful and it enters the host ID, the result of the negotiation, the transaction identifier, the token value, and the signed privacy data offer in the history folder.

5.6 Testing Environment

This protocol has been tested in a local environment. A description of the local testing environment is as follows:

- One Laptop, Sony VAIO, Pentium 2.8 GHZ with 512 MB RAM on it.
- Operating System: Windows XP.
- Six Tahiti servers were running on the same machine, one represented the origin host and the other five represented the host platforms.
- Three types of agents were used over these servers: one monitoring agent, one mobile agent at the origin host, and a host agent was at each remote host platforms. Thus, a total of seven agents were used. Each remote host had one host agent which interacts with incoming mobile agents.
- The host agents on different hosts are not same; they might have different code and host policy on which negotiation is to be done.
- The host agent always remains in run state (i.e., always available).

Currently, our implemented mobile agent encompasses basic functionality, such as the ability to move to different hosts on the network, sending messages to and receiving messages from the remote host including negotiation, token confirmation, and agreement confirmation phase where agent switched to appropriate state and responds with an appropriate message, for example, if token is confirmed, the agent remains in token confirmation state, skips the negotiation and collects the offer from the remote host agent. Currently, the size of the implemented mobile agent is 24 KB while the total lines of code are 926, and it will grow as negotiation functionality is added to it. The size and total lines of code for the monitoring agent is 13 KB and 448, respectively. Similarly, the size and total lines of code for the host agent is 15 KB and 546, respectively. It takes between 8 seconds and 12 seconds to perform the above functionalities on a single remote host. The above estimate shows the ideal scenario and the time required to perform the required operations

will vary depending on the traffic over the network, remote host availabilities, and other constraints which should be considered to determine the actual time required to perform the negotiation at each remote host.

Our current implementation of the mobile agent does not carry any user preferences; it carries basic fixed data to make negotiation decisions at each remote host. When user preferences are added, the size of the mobile agent will grow. As well, including the evaluation function to evaluate offers and counteroffers during negotiation, along with a function to choose reasonable alternative preferences, will also increase the size of the agent. Each server ran on different ports on the local machine. In our experiment, origin host ran on default port number i.e., 4434 and other remote host platforms on 4435 to 4439. All five hosts in the experiment simulated the five different scenarios of the protocol. The scenarios which our protocol can work are as follow:

1. **Negotiation achieved:**

   In this scenario, the mobile agent and the host negotiate over the user preferences and host privacy policy, and the negotiation is achieved. The mobile agent collects the signed agent trace offer.

   In Figure 5.15, it can be seen that, host agent sends a message together with its ID to the mobile agent, to know, if the mobile agent has negotiated with the remote host in the past. The mobile agent checks received host ID and determines if it has any past negotiation record. If it finds a token for the current host then it sends the token and its ID to the remote host; otherwise, it sends "token fail" message and starts negotiation over user preferences with the host agent. In this scenario token confirmation is failed and agent does negotiation with the host and the negotiation is achieved.

2. **Negotiation fails:**

   In this scenario, the mobile agent and the host negotiate over the user preferences and host privacy policy, and the negotiation is not achieved. In this scenario the agent collects the signed dummy trace offer.

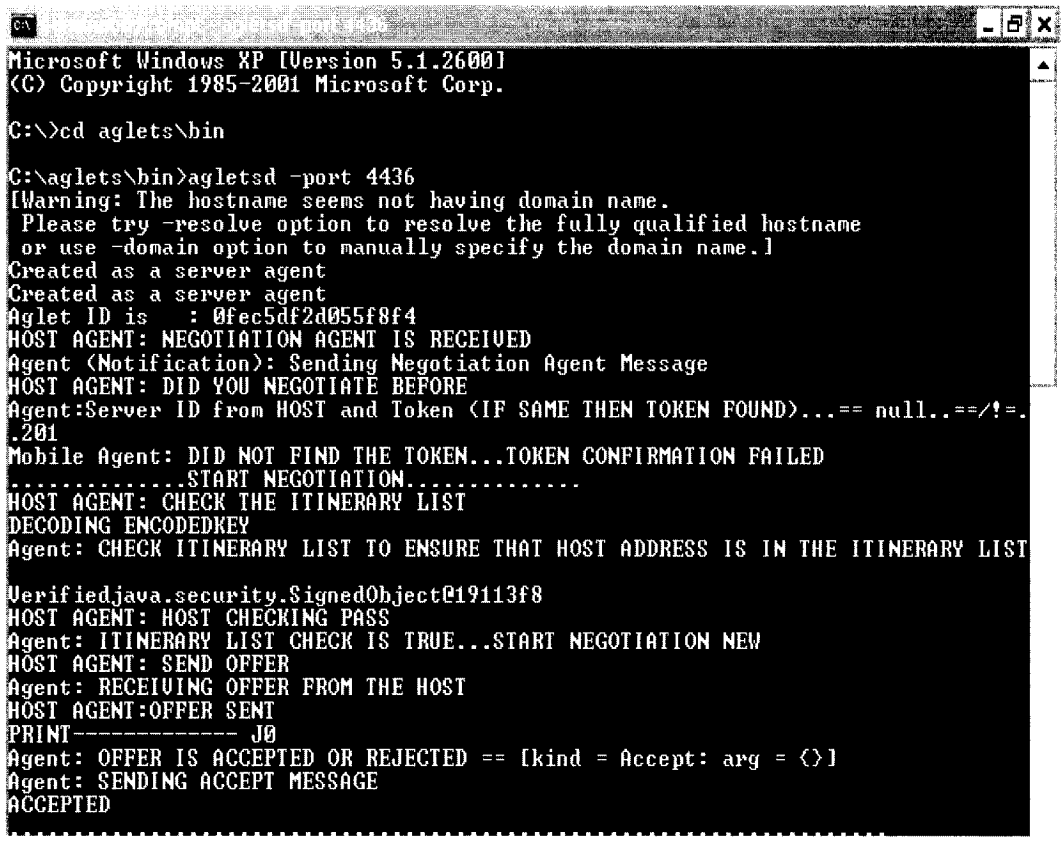

Figure 5.15: Negotiation Achieved

3. **Token confirmation pass:**

In this scenario, the mobile agent and the host agent confirm the token. This is possible if the mobile agent is carrying a token for the host platform and this token match with the token the host has in its permanent history file. In this scenario, the agent collects the signed agent trace offer. This agent trace offer is the trace which was generated in past negotiations between the mobile agent and the host.

From Figure 5.16, it can be seen that, the host agent sends a message together with its ID to the mobile agent, to know if the mobile agent has negotiated with the remote host in the past. The mobile agent checks the received host ID and determines if it has any past negotiation record. If it finds a token for the current host then it sends the token and its ID to the remote host. The remote host checks its permanent history file to determine if it has any past record with the agent ID given and with the corresponding token. If it finds a match, the token confirmation is achieved and the

host confirms the token and indicates readiness to sign the same agent trace offer. In this way token confirmation is achieved.

Figure 5.16: Token Confirmation Scenario

4. Token confirmation fails, negotiation passes:

In this scenario, the token confirmation between the mobile agent and the host agent fails, which starts a negotiation. The negotiation over the user preferences and host

privacy policy is achieved between the mobile agent and the host agent. The agent collects the signed agent trace offer.

A token confirmation fails in two situations:

4.1 Token confirmation fails by mobile agent

This occurs when the mobile agent does not have the token for that host (the agent is negotiating for the first time or the token has been deleted). This scenario is the same as the previously mentioned scenario 1 ("negotiation achieved), as at each host, the mobile agent checks whether it has the token for the current host. If it does not find the token for current host, it starts negotiation. In this scenario, the mobile agent collects the signed agent trace offer.

4.2 Token confirmation fails by host agent

This is possible if the mobile agent is carrying a token for the host platform but this token does not match with the token the host has in its permanent history file. In this scenario, the token confirmation by the host agent fails and the mobile agent starts negotiation. The mobile agent collects the signed agent trace offer.

5. Token confirmation fails, negotiation fails

In this scenario, the token confirmation between a mobile agent and a host agent fails and the negotiation is also not achieved over the user preferences and host privacy policy. The agent collects the signed dummy trace offer. This can happen in two situations:

5.1 Token confirmation fails by mobile agent

This occurs when the mobile agent does not have a token for the host (the mobile agent negotiating for the first time or the token has been deleted). This scenario is the same as second scenario (negotiation fails). In this scenario, the mobile agent and the host negotiate over user preferences and the host's privacy policy but the negotiation is not achieved. The mobile agent collects the signed dummy trace offer.

5.2 Token confirmation fails by host agent

This is possible if the mobile agent is carrying a token for the host platform and this token does not match with the token the host agent has in its permanent history file. In this scenario, the token confirmation by the host agent fails and the mobile agent starts negotiation. The mobile agent and the host agent negotiate over user preferences and the host privacy policy, but the negotiation is not achieved. The agent collects the signed dummy trace offer.

6. **Agreement confirmation pass**

   This scenario takes place in the second round of the mobile agent. In this scenario, the negotiation between a mobile agent and host was achieved (negotiation pass or token confirmation) in the first round and the agent trace offer of the host has been selected as the optimal offer. When agent visits the selected host, the host agent confirms the negotiation and provides a signed privacy data offer, together with other relevant information such as token value and agent trace offer.

7. **Agreement confirmation fails**

   This scenario takes place in the second round of the mobile agent. In this scenario, the negotiation between a mobile agent and host was achieved (negotiation pass or token confirmation) in the first round and the agent trace offer of the host has been selected as the optimal offer. When agent visits the selected host, the host agent finds any discrepancies (such as the agent trace offer not being valid or the agent got was late in arriving at the remote host) and rejects the negotiation and simply responds with a rejection message.

Discussion

The trace based protocol implemented in this thesis achieves the desired level of security, including data confidentiality, data authenticity, data integrity, freedom from interleaving attack, improved truncation resilience, and identification of malicious hosts (if they have truncated the offer from the chain of offers). Overall, the proposed security protocol provides security to the mobile agent from most of the major attacks that can be mounted by

a malicious host, except for a denial of service attack (which is not addressed in this study). However, it comes at the cost of a number of cryptographic operations which slow down the negotiation process and put overhead on the origin host (to verify the digital signature or to decrypt each offer supplied by the host). For example, to achieve data confidentiality, each host encrypts the offer with the originator's public key so that only the originator can retrieve the protected offer. If an agent visits "n" number of hosts, the origin host has to run the decrypt operation "n" times to get the clear text offer for each host. In the same way, data authenticity and non-repudiation are achieved by each host digitally signing the data and the static piece of agent data. This also incurs overhead on the origin host, as the origin host has to verify the signatures on each offer provided.

On the other hand, the proposed architecture gives us several advantages over the client server architecture: bandwidth problem over the network is reduced (no need to exchange messages between the client and the server); there is no need for any user interference during the entire negotiation process; the user device can go offline after dispatching the mobile agent and can reconnect later to collect the results; the origin host don't need to make a connection to each Web site to retrieve privacy policy and compare them with the user preferences; the origin host does not need to do any evaluation of offers and counter-offers during negotiation on its platform; and most importantly, if a device is resource constrained (i.e., connectivity is poor) this architecture is the optimal choice to achieve the negotiation with the remote host.

Currently, our implemented mobile agent encompasses basic functionality, such as the ability to move to different hosts on the network, sending messages to and receiving messages from the remote host and to collect a signed offer from the remote host. It works in all three scenarios, namely, negotiation state, token confirmation state, and agreement confirmation state. During collecting the signed offer, the agent utilizes a chaining mechanism to create a set of encapsulated offers. This chaining mechanism prevents platforms from maliciously truncating, replacing or inserting any of the encapsulated offers without getting detected at the origin host. At the origin host, the monitoring agent uses the

chain to retrieve the offers to determine the collected offer, to check if there was any malicious host who had truncated, replaced, or inserted offer from the chain.

Our current implementation of the mobile agent does not carry any user preferences; it carries basic data to make negotiation decisions at each remote host. When user preferences for desired transactions are added, the size of the mobile agent will grow. As well, including the evaluation function to evaluate the offer and counter-offer during negotiation, along with a function to choose reasonable alternative preferences as the counter-offer will also increase the size of the mobile agent.

We have a partially implemented architecture; more work is required on the negotiation side. It will not be very clear whether the performance of the mobile agent based architecture will be better than the existing client server architecture (in the literature, no working client server architecture) until complete implementation is finished, and performance analysis is made for the proposed architecture with respect to the existing client server architecture over the network. In theory, our architecture does incur overhead to the user side in the form of a number of cryptographic operations, but it also provides several major advantages for the origin host. If the user device is resource constrained, this architecture becomes is the optimal choice, but less constrained devices may also benefit from this work.

Chapter 6: Conclusions and Future Work

This chapter discusses the conclusions of the study. With the growing popularity of e-Commerce, the privacy concerns of users are also increasing. Privacy negotiation may play a major role in enhancing the user's confidence by giving him or her more control over the release of their personal information.

6.1 Summary

In this study, a privacy negotiation architecture using a mobile agent has been proposed and a security protocol used to store the result of the negotiation at each host has been implemented. This study has addressed two broad aspects of privacy negotiation, the design of a privacy negotiation architecture using a mobile agent, and the implementation of a trace based security protocol which is used to collect the results of the negotiation at each host.

With regard to the first aspect of privacy negotiation, we have described how the mobile agent is used to do privacy negotiation with the remote host. In our architecture, it has been indicated how OWL can be used to represent the user preferences. It has been shown by an example how an OWL files can be used to construct user preference trees for each desired transaction. This means each transaction will have a separate tree of user's preferences. The advantage of taking separate user's preferences for each transaction is to accurately capture the user's strictness about his or her personal information, since user privacy sensitivity about personal information may vary according to each transaction. The other benefit is that a mobile agent only has to carry the required preferences for a transaction instead of carrying the preferences for all transactions together. The draw back is redundancy of personal information, as users may have the same level of strictness for certain personal information for all the transactions.

Preference ranking is used to explicitly cover the user's strictness about his or her personal information. Privacy sensitivity for a data element may vary from user to user. During the setting of preferences, the user gives each preference a score of 1-20. This score range sub-divides user preferences into four slots: critical preferences, primary preferences, secondary

preferences, and irrelevant preferences. During conversion, an identifier is labeled with each preference corresponding to its score value. This is used for two reasons. First, it is used to produce the agent trace offer, when the agent trace offer is collected instead of the privacy data offer. This reduces the size of the agent, thereby saving bandwidth over the network. Secondly, it facilitates the evaluation of the agent trace offers when the agent returns after the first round of visits to all the hosts on the itinerary list.

The architecture works in four processing phases: a pre-processing phase, a negotiation phase, a post-processing phase, and an agreement confirmation phase. These four processing phases take place in two rounds of a mobile agent. In the first round, the mobile agent collects the results of the negotiations at each host. When the mobile agent returns to the origin host, the optimal offer is determined and the mobile agent is dispatched again to the selected host to get the final confirmation of the negotiation and to collect the privacy data offer.

At each remote host, if negotiation or token confirmation is achieved, the mobile agent collects the signed agent trace offer; otherwise, it collects a signed dummy trace offer. In the agreement confirmation phase, it collects the signed privacy data offer.

With regard to the second aspect of the privacy negotiation, we have discussed and implemented a trace based security protocol to securely collect the results of the negotiations at each host. This protocol achieves the desired security properties, including data confidentiality, data authenticity, and data integrity, freedom from interleaving attack, improved truncation resilience, and identification of malicious hosts.

In the last section of this study, we extended a generic protocol proposed by Maggi and Sisto [55], to enhance truncation resilience. We have proposed two extensions to the generic protocol. Both mechanisms take place when the mobile agent returns after visiting all the hosts on the list. Both are based on two assumptions: the mobile agent must visit all hosts on its itinerary list, and the mobile agent should not visit the same host twice on its journey. These mechanisms improve truncation resilience by detecting all truncated hosts' offers

during the agent's itinerary, and by enable the origin host to identify malicious hosts which have truncated offers from the chain. In first mechanism, the origin host sends a message to all the hosts suspected of having been truncated, but in second mechanism it only sends a message to the hosts which were on the itinerary list and have either added other hosts to the itinerary list or whose offer has been truncated from the chain of offers. The first mechanism places more overhead on the origin host because origin host is required to send messages to the hosts identified as having been truncated, while second mechanism distributes the overhead among all the remote hosts, following a tree based tracing. Thus, with the second mechanism, an origin host needs to send very few messages compared with the first. The only drawback of the second mechanism is dependency on the remote host platform to follow the tracing mechanism.

6.2 Results

The main results of this study can be summarized as follows:

1. This study proposes a mobile agent based privacy negotiation architecture, which aims to enhance user control over personal information, so that users can decide what personal information they wish to release to a Web site.
2. In order to address the security concerns posed by a malicious agent, a security protocol has been proposed in this study. This protocol provides a defense against most of the major attacks that can be mounted by a malicious host against the mobile agent.
3. Finally, in order to improve the truncation resilience of the free roaming mobile agent, Maggi and Sisto's generic protocol [55] has been extended. We have proposed two extensions to the original protocol. These mechanisms provide freedom from two hosts colluding truncation attack and also allows the origin host to identify malicious hosts who have truncated host offers.

6.3 Recommendations

In this study, we choose one optimal offer, i.e., only one best host offer among the offers provided by all hosts is selected after evaluation. However, this architecture can also be

deployed in selecting multiple host offers. For example, selecting the offer of the top five hosts, rather than selecting only the offer of the top host. In the future, when a user has to do some transaction, rather than sending the mobile agent again for the negotiation to all different hosts, it could just send the agent to these five pre-selected hosts.

This architecture is suited to a scenario in which the devices have limited resources and cannot be online all the time (i.e., poor connectivity). Another mechanism which can replace or can be a good choice for such a scenario is the use of a proxy agent. This proxy agent will sit on a trusted host and act on behalf of the user. In such an architecture, the user makes an initial request to the proxy agent with some relevant data. The proxy agent then does privacy negotiation with all the hosts and supplies the final result of the negotiation to the user. This way, the user device does not have to do anything other than to make a request and collect the final result of the negotiation. The only problem with this approach is the need for support from another device or entity which can be trusted (i.e., operating on a trusted host). It might cause network overhead, as it is somewhat similar to the client-server architecture.

6.4 Future Work

1. In this study, we have not incorporated the actual negotiation between the mobile agent and the host agent. To demonstrate the functioning of the architecture, the protocol stores fixed data rather than the actual result of negotiation. Thus, more work is needed to implement the actual negotiation between the mobile agent and the host agent.

2. Furthermore, in our architecture, we have suggested that during counter-offer generation, the mobile agent chooses reasonable alternatives which depend upon the results of the host agent offer/counter-offer evaluation. However, there is not enough description in this study of how this offer/counter-offer is selected. More research is needed to address this topic. The distance based approach proposed in [56] provided a possible solution to generate reasonable counter-offers.

3. This study has only addressed the negotiation of user preferences with host privacy practices represented in P3P. It does not shed any light on one other aspect of privacy, supply chain negotiation, where different users wish to release personal information to the Web site and which may lead to different prices for products or services.

4. In the present study, it is assumed that P3P provides an alternative to specify the alternative preferences and that during offer/counter-offer these preferences are converted into a generalized representation (as described in the chapter 3) and then supplied to the mobile agent. Not much work has been done on the remote host side; more work is required, especially on how to generate counter-offers and offers during negotiation.

5. The time required for the complete negotiation process with all security measures in place needs to be determined. This would allow us to measure how much time the mobile agent based architecture takes in negotiation with "n" number of hosts, as compared to the client server architecture. As well, this would allow us to determine how much time an origin host is takes to perform certain cryptographic operations.

6. In the present study, we have partially implemented our architecture. More work is required on the negotiation side, as there is no existing network which supports the mobile agent paradigm and P3P negotiation. Therefore, the relative performance of the architecture will not be clear until complete implementation is finished, and a performance analysis is done of the proposed architecture in comparison to the existing client server architecture over the network. Currently, the proposed architecture is ideally suited for a scenario where a device is bandwidth restricted, that is, it cannot be online all the time and might not be able to do the negotiation. If the security of the mobile agent is not considered, this architecture clearly has an edge over client server architecture. However, it seems there might be differences in the performance of the architecture after security is included in the system.

7. Another question yet to be answered is the size of the mobile agent. After complete implementation of the mobile agent, its size should be reasonable, so that it will not place extra overload on the network. Currently, our implemented mobile agent encompasses basic functionality, such as the ability to move to different hosts on the network, sending messages to and receiving messages from the remote host, collecting the signed offer from the remote host, etc. Our implemented mobile agent is 24 KB, and it will grow as negotiation functionality is added to it. Our current implementation of the mobile agent does not carry any user preferences; it carries basic fixed data to make negotiation decisions at each remote host. When user preferences are added, the size of the mobile agent will grow. As well, including the evaluation function to evaluate offers and counteroffers during negotiation, along with a function to choose reasonable alternative preferences, will also increase the size of the agent. The size of the mobile agent must be considered in our future research.

8. The implemented mobile agent in this study takes between 8 seconds and 12 seconds to perform the functionality on a Sony VAIO Pentium 4, 2.8 GHz machine with 512 MB RAM on it. As this architecture is not yet completely implemented, we have not tested it on a resource constrained device. Future work should include testing the mobile agent on resource constrained devices, and determining the minimum resources required to run the system on the device and to measure its efficiency.

References

[1] Agrawal, R., Kiernan, J., and Srikant, R., "Xpref: A Preference Language for P3P," *Proceedings of the 12th International World Wide Web Conference*, 2003.

[2] Agrawal, R., and Wimmers, E.L., "A Framework for Expressing and Combining Preferences," *Proceedings of the International Conference on Management of Data*, Vol. 29, 2000, pp. 297-306.

[3] Baldi, M., and Picco, G. P., "Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications," *Proceedings of the 20th International Conference on Software Engineering*, 1998, pp. 146–155.

[4] Bamasak, O., and Zhang, N., "A Secure Method for Signature Delegation to Mobile Agents," *Proceedings of the Symposium on Applied Computing*, Vol. 1, 2004, pp. 813-818.

[5] Bennicke, M., Langendorfer, P., "Towards Automatic Negotiation of Privacy Contracts for Internet Services," *Proceedings of the 11th International Conference on Networks*, 2003, pp. 319 – 324.

[6] Berthold, O., Köhntopp, M., "Identity Management Based on P3P," *Proceedings of the International Workshop on Designing Privacy Enhancing Technologies: Design Issues*, 2001, 141-160.

[7] Bodorik, P., and Jutla, D., "Architecture for User-Controlled E-Privacy," *Proceedings of the Symposium on Applied Computing*, 2003, pp. 609-616.

[8] Borrell, J., Robles, S., and Serra, J., "Securing the Itinerary of Mobile Agents through a Non-Repudiation Protocol," *Proceedings of the 33rd Annual International Carnahan Conference on Security Technology*, 1999, pp. 461-4.

[9] Borselius, N., "Mobile Agent Security," *Electronics & Communication Engineering Journal*, Vol. 14, 2002, pp. 211-218.

[10] Byers, S., Cranor, L.F., and Kormann, D., "Automated Analysis of P3P-Enabled Web Sites," *Proceedings of the 5th International Conference on Electric Commerce*, Vol. 5, 2003, pp. 326-338.

[11] Chavez, A., Maes, P., "Kasbah: An Agent Marketplace for Buying and Selling Goods," *Proceedings of the 1st International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, 1996, pp. 75–90.

[12] Cheng, J., and Wei, V., "Defenses against the Truncation of Computation Results of Free-Roaming Agents," *Proceedings of the 4th International Conference on Information and Communications Security*, Vol. 2513, 2002, pp. 1–12.

[13] Chess, D., and Harrison, C., and Kershenbaum, A., "Mobile Agents: Are they a Good Idea?," *IBM Research Report RC 19887 (88465)*, 1994.

[14] Claessens, J., Preneel, B., and Vandewalle, J., "(How) Can Mobile Agents do Secure Electronic Transactions on Untrusted Hosts?: A Survey of the Security Issues and the Current Solutions," *ACM Transaction on Internet Technology*, Vol 3, Issue 1, 2003, pp. 28-48.

[15] Corradi, A., Montanari, R., and Stefanelli, C., "Mobile Agents Integrity in E-Commerce Applications," *Proceedings of the 19th International Conference on Distributed Computing Systems*, 1999, pp. 59-64.

[16] Cranor, L., Langheinrich, M., Marchiori, M., Presler-Marshall, M., and Reagle, J., "The Platform for Privacy Preferences 1.1 (P3P1.1) Specification," *W3C Recommendation*, Available online at: http://www.w3.org/TR/P3P.

[17] Cranor, L.F., and Reidenberg, J., "Can User Agents Accurately Represent Privacy Notices?" *A Discussion Draft*. Available online at: http://tprc.org/papers/2002/65/ tprc2002-useragents.pdf.

[18] Cranor, L.F., and Resnick, P., "Protocols for Automated Negotiations with Buyer Anonymity and Seller Reputations," *Netnomics*, Vol. 2, 2000, pp. 1-23.

[19] Cranor, L.F. et al, "A P3P Preference Exchange Language," Available online at: http://www.w3.org/TR /P3P-preferences.

[20] Cranor, L. F., Arjula, M., and Guduru, P. "Use of a P3P User Agent by Early Adopters," *Proceedings of the ACM Workshop on Privacy in the Electronic Society*, 2002, pp. 1-10.

[21] Dadon-Elichai, A., "RDS: Remote Distributed Scheme for Protecting Mobile Agents," *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-agent Systems*, 2004, pp.354 – 361.

[22] Das, A., Gongxuan, Y., "A Secure Payment Protocol Using Mobile Agents in an Untrusted Host Environment," In: *Lecture Notes in Computer Science*, Vol. 2040, 2001, pp. 33.

[23] Dias, P., Ribeiro, C., and Ferreira, P., "Enforcing History-Based Security Policies In Mobile Agent Systems," *Proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks*, 2003, pp. 231-234.

[24] Dikaiakos, M. D., and Samaras, G., "Performance Evaluation of Mobile Agents: Issues and Approaches," In: *Performance Engineering, State of the Art and Current Trend, Lecture Notes in Computer Science*, Vol. 2047, 2001, pp. 148-166.

[25] Domingo Ferrer, J., "Mobile Agent Route Protection through Hash-Based Mechanisms," *Proceedings of the 2nd International Conference on Cryptology in India: Progress in Cryptology*, Vol. 2247, 2001, pp. 17 – 29.

[26] El-Khatib, K., "A Privacy Negotiation Protocol for Web Services," *Proceedings of the Workshop on Collaboration Agents: Autonomous Agents for Collaborative Environments*, 2003.

[27] Esparza, O., Fernandez, M., Soriano, M., Muñoz, J.L., and Forne, J., "Mobile Agent Watermarking and Fingerprinting: Tracing Malicious Hosts," In: *Lecture Notes in Computer Science*, Vol. 2736, 2003, pp. 927 – 936.

[28] Esparza, O., Soriano, M., Muñoz, J.L., and Forne, J., "Punishing Manipulation Attacks In Mobile Agent Systems," *IEEE Global Telecommunications Conference*, Vol. 4, 2004, pp. 2235-2239.

[29] Farmer, W.M., Guttman, J.D., and Swarup, V., "Security for Mobile Agents: Issues and Requirements," *Proceedings of the National Information System Security Conference*, 1996. Available Online at: http://csrc.nist.gov/nissc/1996/ papers/NIS SC96/ paper033/SWARUP96.PDF

[30] Fischmeister, S., Vigna, G., and Kemmerer, R.A., "Evaluating the Security of Three Java-Based Mobile Agent Systems," *Proceedings of 5th International Conference on Mobile Agents*, 2001, pp. 31-41.

[31] Gray, R., Kotz, D., Nog, S., Rus, S., and Cybenko, G., "Mobile Agents: The Next Generation in Distributed Computing," *Proceedings of the* 2nd International Symposium on Parallel Algorithms/Architectures Synthesis, 1997, pp 8-24.

[32] Gray, R.S., Kotz, D., Peterson, R., Barton A., "Mobile-Agent versus Client/Server Performance: Scalability in an Information-Retrieval Task," *Proceedings of the 5th International Conference on Mobile Agents*, Vol. 2240, 2002, pp. 229-243.

[33] Grimshaw, D., *Overview of Aglets*, Available online at: http://www.ryerson. ca /~dgrimsha/courses/cps720.

[34] He, Y., *An Arbitrator Agent for e-Privacy*, Master's Thesis, Available from the National Library of Canada, Ottawa, Canada, 2004.

[35] He, Y., Jutla, D., "Contextual e-Negotiation for the Handling of Private Data in e-Commerce on a Semantic Web," *Proceedings of the 39thAnnual Hawaii International Conference on System Sciences*, 2006.

[36] Hogben, G., "Describing the P3P base data schema using OWL," *Proceedings of PM4W at WWW2005*, 2005.

[37] Hohl, F., "Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts," *Mobile Agents and Security, Lecture Notes in Computer Science*, Vol. 1419, 1998, pp. 92-113.

[38] Hung, P.C.K., Li, H., and Jeng, J., "WS-Negotiation: An Overview of Research Issues," *Proceedings of the Hawaii International Conference on System Sciences*, 2004, pp. 493-502.

[39] Jansen, W.A., "Countermeasures for Mobile Agent Security," *Computer Communications*, Vol. 23, 2000, pp. 1667-76. Available online at: http://csrc.nist.gov /mobilesecurity/Publications/ppcounterMeas.pdf

[40] Jansen, W., and Karygiannis, T., *NIST Special Publication 800-19 – Mobile Agent Security*, Available online at: http://csrc.nist.gov/publications/nistpubs/ 800-19/sp800-19.pdf.

[41] Jutla, D., Zhang, Y., "Maturing E-Privacy with P3P and Context Agents," *Proceedings of the International Conference on e-Technology, e-Commerce and e-Service*, 2005, pp. 536- 541.

[42] Karnik, N. M., and Tripathi, A. R., "A Security Architecture for Mobile Agents in Ajanta," *Proceedings of 20th International Conference on Distributed Computing Systems*, 2000, pp 402–409.

[43] Karnik, N.M., and Tripathi, A.R., "Design Issues in Mobile Agent Programming Systems," *IEEE Concurrency*, Vol. 6, No. 3, 1998, pp. 52-61.

[44] Karjoth, G., Asokan, N., and Gülcü, C., "Protecting the Computation Results of Free-Roaming Agents," *Proceedings of the 2nd International Workshop on Mobile Agents*, Vol. 1477, 1998, pp 195- 207.

[45] Karjoth, G., Lange, D.B., and Oshima, M., "A Security Model for Aglets," In: *Mobile Agents and Security, Lecture Notes in Computer Science*, Vol. 1419, 1998, pp. 188-205.

[46] Karjoth,G., Schunter, M., Herreweghen, E.V., Waidner, M., "Amending P3P for Clearer Privacy Promises," *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, pp. 445, 2003.

[47] Lange, D.B., Oshima, M., *Programming And Deploying Java Mobile Agents with Aglets*, Addison-Wesley, New York, U.S.A., 1998.

[48] Lange, D. B. and Oshima, M., "Seven Good Reasons For Mobile Agents," *Communications of the ACM*, Vol. 42, No. 3, 1999, pp. 88-89.

[49] Langendotfer, P., Kraemer, R., "Towards User Defined Privacy in Location-aware Platforms," *Proceedings of the 3rd International Conference on Internet Computing*, 2002.

[50] Lategan, F.A., and Olivier, M.S., "A Chinese Wall Approach to Privacy Policies for the Web," *Proceedings of 26th Annual International Computer Software and Applications*, 2002, pp. 940-4.

[51] Menzes, A. J., Oorschot, P.C., Vanstone, S.A., *Handbook of Applied Cryptography*, CRC Press, Boca Raton, U.S.A., 1996.

[52] Necula, G., Lee, P., *Proof-Carrying Code*, Available Online at: http://www.cs.cmu. edu /~petel/papers/pcc/pcc.html

[53] Leroy, X., "Java Bytecode Verification: An Overview," *Proceedings of the Conference on Computer Aided Verification*, July 2001, pp. 265-85.

[54] Maaser, M., Langendoerfer, P., "Automated Negotiation of Privacy Contracts," *Proceedings of the 29th International Conference on Computer Software and Applications*, 2005, pp. 505 – 510.

[55] Maggi, P., and Sisto, R., "A Configurable Mobile Agent Data Protection Protocol," *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*, 2003, pp 851– 858.

[56] Meyer, J., "How to Manage, Negotiate, and Transfer Personal Information on the Web," *A Technical Publication of the Almaden Research Center, International Business Machines*, 1999.

[57] "OWL Web Ontology Language," *A W3C Recommendation*, 2006. Available online at: http://www.w3.org/TR/owl-features.

[58] Parsons, S., Sierra, C., and Jennings, N., "Agents that Reason and Negotiate by Arguing," *Journal of Logic and Computation*, Vol. 8, No. 3, 1998, pp. 261-292,.

[59] Patrick, A.S., "Building Trustworthy Software Agents," *IEEE Internet Computing*, Vol. 6, No. 6, 2002, pp. 46-53.

[60] Preibusch, S., "Implementing Privacy Negotiation Techniques in E-Commerce," *Proceedings of the 7th IEEE International Conference on E-Commerce Technology*, 2005, pp. 387-390.

[61] "Privacy Survey Results," *Centre for Democracy and Technology*, Available Online at: http://www.cdt.org/privacy/survey/findings.

[62] "Public Opinion on Privacy," *Electronic Privacy Information Centre*, Available Online at: http://www.epic.org/privacy/survey.

[63] Puliafito, A., Riccobene, S., and Scarpa, M., "An Analytical Comparison of the Client-Server, Remote Evaluation and Mobile Agents Paradigms," *Proceedings of the 1st International Symposium on Agent Systems and Applications*, 1999.

[64] Rahul J., Sridhar I., "Performance Evaluation of Mobile Agents for E-commerce Applications," *Lecture Notes in Computer Science*, Vol. 2228, 2001, pp. 331.

[65] Riordan, J., Schneier, B., "Environmental Key Generation towards Clueless Agents," Ed.: Vigna, *Mobile Agents and Security, Lecture Notes in Computer Science*, Vol. 1419, 1998.

[66] Roth, V., "Empowering Mobile Software Agents," *Proceedings of the 6th Mobile Agents Conference*, Vol. 2535, 2002, pp. 47–63.

[67] Roth, V., "Secure Recording of Itineraries Through Cooperating Agents," *Proceedings of the ECOOP Workshop on Distributed Object Security and 4th*

Workshop on Mobile Object Systems: Secure Internet Mobile Computations, France, 1998, pp. 147-154.

[68] Roth, V., "On the Robustness of some Cryptographic Protocols for Mobile Agent Protection," *Mobile Agents, Lecture Notes in Computer Science*, Vol. 2240, 2001.

[69] Sander, T., and Tschudin, C., "Protecting Mobile Agents against Malicious Hosts," *Mobile Agents and Security, Lecture Notes in Computer Science*, Vol. 1419, 2001, pp. 44–60.

[70] Schneider, F.B., "Towards Fault-Tolerant and Secure Agent," *Proceedings of the 11th International Workshop on Distributed Algorithms*, 1997.

[71] Sierra, C., "Agent-Mediated Electronic Commerce," *Autonomous Agents and Multi-Agent Systems*, Vol. 9, No. 3, 2004, pp. 285 – 301.

[72] Smith, R. and Shao, J., "Preserving privacy when preference searching in e-commerce," *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society (WPES)*, Washington, DC, 2003, pp. 101-110.

[73] Thibadeau, R., *A Critique of P3P: Privacy on the Web*, A Technical Report available from The E-Commerce Institute, School of Computer Science, Carnegie Mellon University, U.S.A.

[74] Theilmann W. and Rothermel K., "Optimizing the Dissemination of Mobile Agents for Distributed Information Filtering," *IEEE Concurrency*, Vol. 8, No. 2, 2000, pp. 53-61.

[75] Veit, D., Muller, J.P., and Schneider, M., "Matchmaking for Autonomous Agents in Electronic Marketplaces," *Proceedings of the 5th International Conference on Autonomous Agents*, Montreal, Canada, 2001, pp. 65-66.

[76] Vigna, G., "Protecting Mobile Agents through Tracing," *Proceedings of the 3rd ECOOP Workshop on Mobile Object Systems*, Finland, 1997.

[77] Wong, D., Paciorek, N., and Moore, D., "Java-Based Mobile Agents," *Communications of the ACM*, Vol. 42, No. 3, 1999, pp. 92-95.

[78] Yee, B.S., "A Sanctuary for Mobile Agents," In: *Secure Internet Programming: Security Issues for Mobile and Distributed Objects, Lecture Notes in Computer Science*, Springer-Verlag, Vol. 1603, 1999, pp. 261– 273.

[79] Yee, G., Korba, L., *Privacy Policies and their Negotiation in Distance Education*, Publication of the National Research Council of Canada, 2004. Available online at: http://iit-iti.nrc-cnrc.gc.ca/publications/nrc-46555_e.htm

[80] Yee, G., Korba, L., "Semi-automated Derivation of Personal Privacy Policies," *Proceedings of the Information Resources Management Association International Conference*, 2004.

[81] Zhang, S., Ye, S., and Makedon, F., "A Hybrid Negotiation Strategy Mechanism In An Automated Negotiation System," *Proceedings of the 5th ACM Conference on Electronic Commerce*, 2004, pp. 256-257.