2003

# Mobile Agent Protection with Data Encapsulation and Execution Tracing

Anna Suen

THE FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

# MOBILE AGENT PROTECTION WITH DATA ENCAPSULATION AND EXECUTION TRACING

By

ANNA SUEN

A Thesis submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Master of Science

Degree Awarded:
Spring Semester, 2003

The members of the Committee approve the thesis of Anna Suen defended on April 30, 2003.

_____
Alec Yasinsac
Professor Directing Thesis

_____
Mike Burmester
Committee Member

_____
Lois Hawkes
Committee Member

Approved:

_____
Sudhir Aggarwal, Chair, Department of Computer Science

The Office of Graduate Studies has verified and approved the above named committee members.

# ACKNOWLEDGEMENTS

I would like to thank my major professor, Dr. Alec Yasinsac, who mentored me these last two years. Through our many discussions, he helped me form and solidify ideas. I would also like to thank my wonderful colleagues for listening to my ideas, asking questions when they did not make sense, and providing feedback and suggestions for improving my ideas. I thank my parents, for giving me the loving support and encouragement I needed to get through this. And I thank my sister, Tina, for letting me use her as a guinea pig all these years. I would like to thank all my friends for the continual encouragement and the positive support. Finally, I would like to thank the Department of Defense Information Assurance Scholarship Program for providing the funding for my final year of graduate school.

# TABLE OF CONTENTS

LIST OF FIGURES ..... vi

LIST OF SYMBOLS ..... vii

ABSTRACT ..... viii

1. INTRODUCTION ..... 1

    1.1. A Closer Look at a Mobile Agent ..... 1
    1.2. Mobile Agent Applications ..... 2
    1.3. Mobile Agent Advantages ..... 4
    1.4. Thesis Structure ..... 4

2. MOBILE AGENT SYSTEM SECURITY ..... 5

    2.1. Mobile Agent System Model ..... 5
    2.2. Security Threats ..... 6
    2.3. Security Tools ..... 7
    2.4. Security Requirements ..... 8
        2.4.1. Confidentiality ..... 8
        2.4.2. Integrity ..... 8
        2.4.3. Availability ..... 9
        2.4.4. Accountability ..... 9

3. RELATED WORK ..... 10

    3.1. Publicly Verifiable Chained Digital Signature Protocol ..... 10
        3.1.1. Pitfalls ..... 12
    3.2. Execution Tracing ..... 14
        3.2.1. Pitfalls ..... 17

4. THE ENCAPSULATED OFFER PROTOCOL ..... 19

    4.1. Overview of the Encapsulated Offer Protocol ..... 19
    4.2. The Parts of a Message in EOP ..... 20
        4.2.1. The Encapsulated Offers and Chaining ..... 21
    4.3. Protocol Description ..... 23

4.3.1. Creating and Dispatching the Initial Mobile Agent ..... 23
4.3.2. Receiving a Mobile Agent ..... 23
4.3.3. Receiving and Processing the Results ..... 25
4.3.4. Execution Trace File ..... 27
4.3.5. Notice Message ..... 27
4.4. Security Evaluation ..... 27
4.4.1. Security Evaluation of the Code and State ..... 27
4.4.2. Security Evaluation of the Encapsulated Offers ..... 28
4.4.3. Security Properties ..... 31
4.4.4. Other Security Concerns ..... 32

5. CONCLUSION ..... 33

5.1. Future Work ..... 33

REFERENCES ..... 35

BIOGRAPHICAL SKETCH ..... 36

# LIST OF FIGURES

Figure 1.1. Mobile Agent Migration ..... 1

Figure 1.2. Mobile Agent (MA) for Personal Digital Assistants Example ..... 3

Figure 2.1. Mobile Agent System Model ..... 5

Figure 3.1. Publicly Verifiable Digital Signature Protocol ..... 11

Figure 3.2. Fault in PVCDSP ..... 13

Figure 3.3. Execution Tracing Protocol ..... 16

Figure 4.1. Description of the fields of a message ..... 20

Figure 4.2. Each platform appends his offer to the set of encapsulated offers ..... 22

Figure 4.3. Chaining in the encapsulated offers ..... 23

Figure 4.4. Checking process completed by an intermediate platform ..... 24

Figure 4.5. Checking process completed by originator upon receipt of results ..... 26

Figure 4.6. Notice Message ..... 27

Figure 4.7. Cannot modify code and state ..... 28

Figure 4.8. Cannot collect and append offers ..... 29

Figure 4.9. Cannot truncate the set of encapsulated offers ..... 30

Figure 4.10. Cannot insert an offer after truncation ..... 31

# LIST OF SYMBOLS

| Symbol | Label | Description |
|---|---|---|
| C | Code | mobile agent code |
| S | State | mobile agent state |
| $P_0$ | Originator | owner of the mobile agent |
| $P_{i>0}$ | Platform | computational environment for the mobile agent |
| $o_i$ | Offer | the piece of data that platform $P_i$ submits |
| $O_i$ | Encapsulated offer | the offer $o_i$ encrypted and/or hashed together with some other information |
| $r_0$ | Secret random number | the secret random number used in creating the encapsulated offer $O_i$ |
| $t_{P_i}$ | Timestamp | the timestamp provided by platform $P_i$ |
| $I$ | Identifier | the unique identifier of the mobile agent session |
| $ENC_{P_i}(m)$ | Encryption | encryption of message $m$ with the public key of platform $P_i$ |
| $SIG_{P_i}(m)$ | Signature | signing of message $m$ with the secret key of platform $P_i$ |
| $h(m)$ | Hash function | one-way hash of message $m$ |

# ABSTRACT

Mobile agent systems provide a new method for computer communication. A mobile agent can migrate from platform to platform, performing a task or computation for its originator. Mobile agents are a promising new technology; however, there exist many security issues that need to be addressed. Security issues consist of protecting the agent platform and protecting the mobile agent. The toughest task is protecting the mobile agent, who is subject to attacks from the platform it is operating on. This thesis is concerned with protecting a mobile agent who collects data on behalf of its originator. A new mobile agent protection protocol, the data encapsulation protocol, is presented in this thesis.

# CHAPTER 1

# INTRODUCTION

A new and emerging technology for computers to communicate is via mobile agents. A mobile agent is a piece of code with the property of mobility, allowing the code to freely roam the network to perform a task described by its owner. As with many computing technologies, protecting mobile agents has received a significant amount of attention, and is the focus of this thesis.

## 1.1. A Closer Look at a Mobile Agent

A mobile agent consists of the code and state information needed to perform some computation. More specifically, the state information consists of the execution state and data state. A mobile agent migrates between agent platforms, who execute the mobile agent code and provide data as a result of the execution. The platform from which a mobile agent originates is called the home platform, and the user who creates the mobile agent is referred to as the originator.

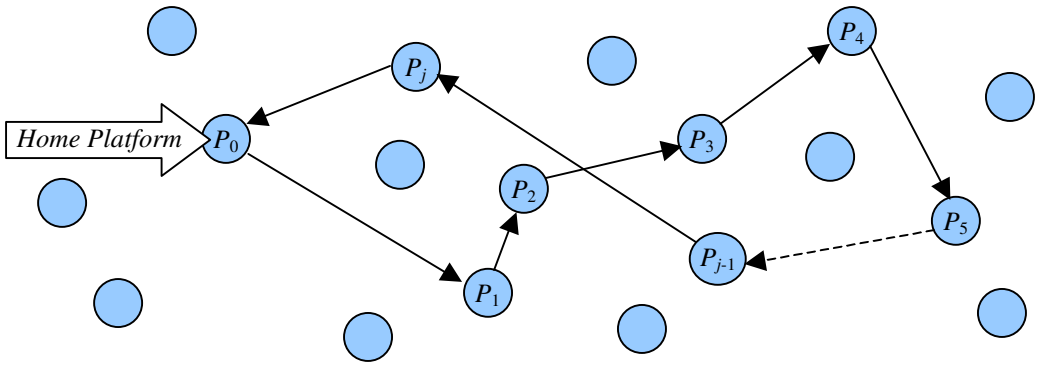


**Figure 1.1. Mobile Agent Migration**

Since a mobile agent can only migrate between agent platforms, these platforms are notably important to the mobile agent [3]. Not only do they execute the mobile agent code, but they also provide functionality, such as migration. When the mobile agent requests to migrate to another platform, the current platform bundles the mobile agent into a transportable form and sends it to the next platform.

Different types of itineraries can be chosen by a mobile agent [5]. The itinerary can be fixed, where the originator defines the platforms the mobile agent is to visit and the order they are to be visited. Alternatively, the itinerary can be partially fixed, where some of the platforms to be visited are known at the time of departure, but the mobile agent is free to decide the next hop. Or, the itinerary can be random, so the mobile agent has complete freedom as to which platform is visited next. This thesis is concerned with partially fixed mobile agent itineraries. The actual decision process of the itinerary is beyond the scope of this thesis.

## 1.2. Mobile Agent Applications

Mobile agents can be applied in many different areas. They can be used for electronic commerce, for network management, or as personal digital assistants [4].

Mobile agents for electronic commerce applications, such as contract negotiating, stock trading, and auctioning, may need access to a platform's database and other sensitive information. Also, the mobile agent may be collecting sensitive information that should be kept secret from a third party. This security requirement may limit the mobility of a mobile agent – the more sensitive the information collected, the less mobile the agent can be. Thus the level of security required for the task directly affects the mobility of the mobile agent.

Network management tasks, such as software updates and distribution as well as remote network management, can also be accomplished with mobile agents. With current network management devices, network administrators must adhere to the parameters defined by the manufacturer of the device. However, with mobile agents, administrators can have greater control of the parameters. Also, mobile agents are dynamic, so they can adaptively respond to network events. For example, a network administrator can dispatch a mobile agent to upgrade a piece of software. If the software does not exist on the machine, either the same mobile agent can install it, or it can initiate another mobile agent to install the software.

Mobile agents for personal digital assistants are becoming the focus of agent developers. Devices, such as cell phones and hand-held digital organizers, that are not continuously online, can take advantage of the mobile agent's ability to achieve a task autonomously. A popular example is that of a user dispatching a mobile agent from his personal digital assistant to collect the best airline ticket price offer from various airline agencies. For example (figure 1.2), if Alice wishes to find an airline ticket from Los Angeles to Chicago, she can create a mobile agent to perform this task for her. With the aid of this mobile agent, Alice will not have to spend countless hours hunting for the perfect ticket. She can specify exactly what time(s) she is willing to accept, the price range, seating preference, etc. Alice then dispatches the mobile agent, which roams the network, visiting various airlines. Each time the mobile agent queries the airline for its best offer. Upon reaching the final airline, the results are sent back to Alice.

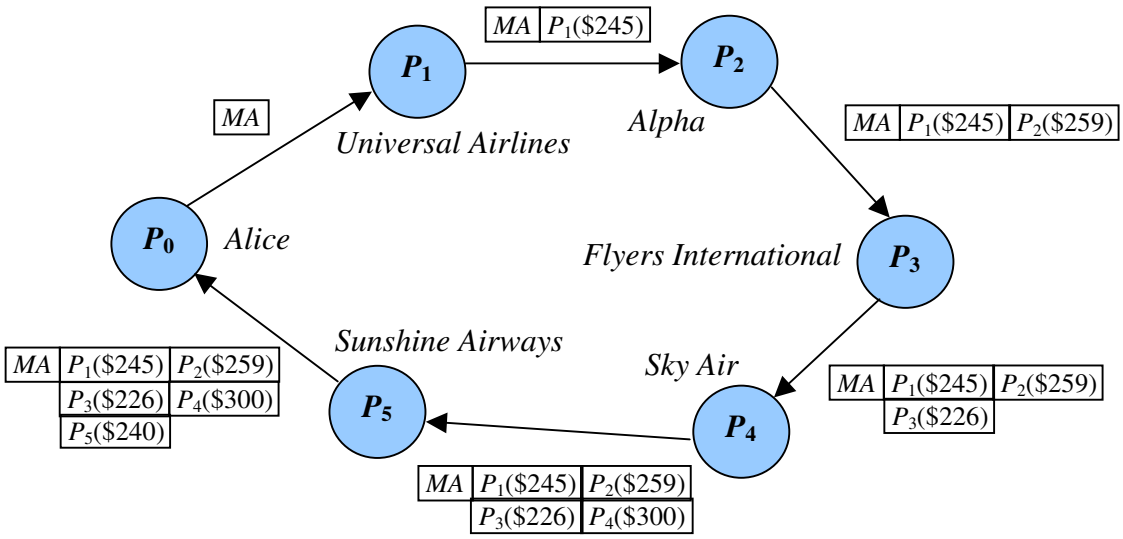


**Figure 1.2. Mobile Agent (MA) for Personal Digital Assistants Example**

Users of a personal digital assistant mobile agent most likely will not create their own mobile agent, but instead will use mobile agents purchased from a reputable vendor or download them. These users also are unlikely to serve as a host to other mobile agents because they are usually everyday people, not vendors of some product or service.

## 1.3. Mobile Agent Advantages

Many advantages result from the use of mobile agents, including elimination of network latency, reduction of network load, asynchronous processing, autonomy, dynamic adaptability, operation in heterogeneous environments, and robust and fault-tolerant behavior [4, 3].

Network latency is eliminated because the mobile agent resides on the platform when it is computing. The mobile agent execution does not depend on latencies in messages sent across slow and unreliable networks. Similarly, the network load is reduced because the mobile agent is sent to the data, rather than bringing large amounts of data to the local platform for processing.

Mobile agents have the ability to perform computations asynchronously. They do not have to be permanently connected to the mobile agent originator to fulfill a task. Also, a mobile agent is autonomous, i.e. it can continue its task without instructions from its originator.

Being autonomous, a mobile agent can dynamically adapt to its environment. For example, if the mobile agent arrives on a platform that is busy, it can sense this obstruction and migrate to another platform that can better serve its needs.

Since mobile agents work in the application layer, they have the ability to operate in heterogeneous computing environments. Mobile agent heterogeneity is commonly made possible by virtual machines or interpreters on the host platform.

Because a mobile agent has dynamic adaptability, it also has a robust and fault-tolerant behavior. For example, if a failure occurs in the system, the host platform can warn all its mobile agents, and they would be given the opportunity to migrate and continue their task on another platform.

## 1.4. Thesis Structure

The focus of this thesis is to protect mobile agents whose task is to collect data from various platforms and return the results to the mobile agent originator (in particular, a personal digital assistant). Chapter 2 discusses the security of mobile agents and their systems. In chapter 3, we describe the protocols that this thesis is based upon. Chapter 4 introduces and details our proposed encapsulated offer protocol. Finally, we conclude in chapter 5.

# CHAPTER 2

# MOBILE AGENT SYSTEM SECURITY

## 2.1. Mobile Agent System Model

Many models for a mobile agent system exist, but a simple one is sufficient for describing security issues. Here, the mobile agent system model consists of two main entities: the mobile agent and the agent platform. The mobile agent is the code and state information needed to perform some described computation. With the property of mobility, mobile agents hop from platform to platform. These agent platforms provide the computational environment for the mobile agent. Many platforms also consist of agents, called platform agents. Platform agents, however, are not mobile. Instead they are stationary on the agent platform and provide system-level services. The home platform is the agent's originator and is the most trusted environment.


**Figure 2.1. Mobile Agent System Model**

## 2.2. Security Threats

There are four categories of security threats on the mobile agent system: platform-to-agent, agent-to-platform, agent-to-agent, and other-to-platform. Platform-to-agent is where the mobile agent is attacked by the platform in which it is performing its computation. Agent-to-platform is just the opposite; the mobile agent attacks the platform. Agent-to-agent is where a platform agent attacks the visiting agent. Each of the previous three categories of attacks takes place within one agent system. Other-to-platform is where the agent platform is attacked by a mobile agent or a platform from a remote system.

The platform-to-agent threat category consists of masquerading, denial of service, eavesdropping, and alteration. A platform can masquerade as a trusted platform to extract sensitive information from a mobile agent. This masquerading harms both the mobile agent and the platform whose identity was assumed. A denial of service attack on the mobile agent can easily be accomplished – simply ignore, delay, or terminate the agent's requests. The platform can also continuously feed the agent with tasks, so that it never gets to the task it is supposed to perform. This attack can affect other agents that depend on the attacked agent, resulting in a deadlock. With access to all the mobile agent's data and instructions, the platform can eavesdrop on the agent's task. For example, if the mobile agent's task is to collect offers from hotels along the beach, the platform can infer that the agent's owner is taking a trip to the beach soon. The platform may share this information with a beachwear retailer, who in turn may solicit the mobile agent's owner to buy beachwear from them. Finally, a platform can modify an agent's code, state, or data, causing the agent to behave in a way that was not originally intended or to return incorrect results to the originator.

The agent-to-platform threat category consists of masquerading, denial of service, and unauthorized access. An unauthorized mobile agent can masquerade as an authorized agent to gain access to a platform's services. Or, an authorized agent can masquerade as an unauthorized agent to avoid being blamed for some mistake that it had made. This attack damages the trust and reputation of the assumed agent. The denial of service attack on the platform can be achieved by executing attack scripts or via unintentional program errors, such as a non-terminating loop. A mobile agent can also attempt to access unauthorized data on the platform.

The agent-to-agent threat category consists of masquerading, denial of service, repudiation, and unauthorized access. Like a malicious platform, a platform agent can masquerade as a reputable vendor in order to gain private information, such as credit card information, from a mobile agent. A platform agent can also launch a denial of service attack by spamming the mobile agent with useless messages and information. Repudiation happens when an agent claims that a transaction never took place. To solve this attack, platforms can keep records to resolve disputes. A platform agent can gain unauthorized access by invoking a mobile agent's public methods. It can modify a mobile agent's data or code, resulting in a change in the attacked mobile agent's behavior. Also, a platform agent can gain information on the mobile agent's activities by eavesdropping on it activities.

The other-to-agent threat category consists of masquerading, unauthorized access, denial of service, and copy and replay. These attacks come from remote platforms and agents. An agent on a remote platform can request services and resources. It can act in conjunction with a malicious platform, or it can act in solitary. A remote user, process, or agent can also gain unauthorized access to a platform's services and resources. Agent platforms are also susceptible to common denial of service attacks coming from a remote entity. Finally, a platform can clone a mobile agent and retransmit it.

## 2.3. Security Tools

The main security tools used in a mobile agent system are encryption, signatures, and hashing. The mobile agent system can operate in either a public key infrastructure or a secret key infrastructure.

In a public key infrastructure, each platform owns a public key and its corresponding secret key. A platform can encrypt data with another platform's public key, allowing only the owner of the public key to decrypt the ciphertext with his secret key. A platform can also encrypt data with his secret key, creating a signature on the data. Any platform, with knowledge of the signer's public key, can verify the signer.

In a secret key infrastructure, each platform shares a separate secret key with the originator. This secret key can be exchanged using a key exchange protocol, such as the Diffie-

Hellman protocol [2]. Since this key is used for both encryption and decryption, it must remain secret.

Other than encrypting and signing, data can be hashed. However, unlike an encryption or signature, the hash values are used to check the integrity of the data and cannot be reversed. The hash value is sent together with the data and the recipient can hash the data and compare it with the one provided. If they match, then the recipient can be confident that the data has not been tampered with.

## 2.4. Security Requirements

The users of a mobile agent system have four main security requirements: confidentiality, integrity, availability, and accountability [4].

### 2.4.1. Confidentiality

Sensitive information or data that are stored on the agent platform or carried by the mobile agent need to be protected for confidentiality. Eavesdroppers can also learn about a mobile agent or platform's activities by observing the message flow. For example, a mobile agent sends messages to an airline agency, followed by a message to its originator, and finally a message to the airline agency. Then, the airline agency sends a message to the bank. Although the eavesdropper does not have the details of this transaction, i.e. where the flight is going or credit card information, she can easily infer that the mobile agent just purchased an airline ticket.

Platforms may maintain an audit log that needs to be protected and kept confidential. These logs are used to solve any disputes between the mobile agent originator and the platform involved in the transaction and therefore must be tamper-proof.

### 2.4.2. Integrity

In order for a platform to execute a mobile agent, it needs to have access to all of the mobile agent's code and state information. Being completely susceptible to the platform on which it is computing, a mobile agent cannot prevent a malicious platform from modifying its

code, state, and data. A malicious platform could also interfere in mobile agent transactions and tamper with the audit logs, resulting in a dispute between the involved parties. However, detection mechanisms can be instilled to detect these types of tampering.

The integrity of the agent platform must also be protected from unauthorized users. Mobile agents can be intentionally malicious, like Trojan horses, or an agent's code may contain unintentional errors that can harm the integrity of the platform it is computing on.

The importance of maintaining the integrity of mobile agent code, state, and data needs to be weighed against the advantages of agent mobility. An agent's mobility may be restricted if it is carrying sensitive information.

### 2.4.3. Availability

Agent platforms must make sure that its data and services are available to mobile agents. The agent platform must provide services, such as concurrency support, deadlock management, and fault-tolerance and recovery, to mobile agents. Hundreds or perhaps thousands of mobile agents may be requesting services from a single platform at the same time, so agent platforms need to have the capability of handling large volumes of requests and providing proper rejection of service.

### 2.4.4. Accountability

Both mobile agents and agent platforms need to be held accountable for their actions. Mobile agents need to be held accountable for services and data it accessed, and platforms need to be held accountable for the services and data it provided. These actions need to be uniquely identified, authenticated, and logged.

Logging is required to hold mobile agents and platforms accountable for their actions. This log, maintained by either or both the mobile agent and agent platform, must be tamper-proof and non-repudiable. Measures need to be in place to handle situations when the log becomes full.

# CHAPTER 3

# RELATED WORK

The two main works related to this thesis are the publicly verifiable chained digital signature protocol and the execution tracing protocol. This chapter details each of these protocols and describes their pitfalls.

## 3.1. Publicly Verifiable Chained Digital Signature Protocol

The publicly verifiable chained digital signature protocol (PVCDSP) [5] uses the data encapsulation technique where an offer is encrypted with some other information to create the encapsulated offer. So, the offer is "encapsulated" within some other information. This protocol relies on a public key infrastructure. Each platform in this system has a unique identifier and a public signature verification key that is issued by the certification authority. There also exists a directory service from which to retrieve certificates.

In this protocol, each platform signs with its secret key for non-repudiability and unforgability. They also encrypt the offer with their public key for data confidentiality. PVCDSP uses the hash chaining mechanism to link the offer from the current platform with the identity of the next platform. The hash chain prevents a server from modifying its own offer later. In this protocol, anyone is allowed to authenticate the servers involved, thus publicly verifiable.

The protocol begins with the originator $P_0$ picking a random number $r_0$. He then hashes $r_0$ with the identity of the next platform $P_1$, producing the chaining value $H_0$. Then he creates the encapsulated offer $O_0$ by signing the encryption of his dummy offer $o_0$ and $r_0$ with the hash value $H_0$. The encapsulated offer $O_0$ is sent to platform $P_1$. $P_1$ uses the encapsulated offer he received to compute his chaining value, i.e. hashes the encapsulated offer with the identity of the next

platform. He then creates his encapsulated offer in the same way as before. The equations are given as follows:

Encapsulated offer: $O_i = SIG_{P_i}(ENC_{P_0}(\, o_i, r_i\,), H_i\,)$

Chaining relation: $H_0 = h(\, r_0, P_1\,)$

$H_i = h(\, O_{i\text{-}1}, P_{i+1}\,)$

An example of the PVCDSP is given in figure 3.1. For illustrative purposes, we have only showed four platforms making offers. In reality, it is most likely that there will be many more offerers.

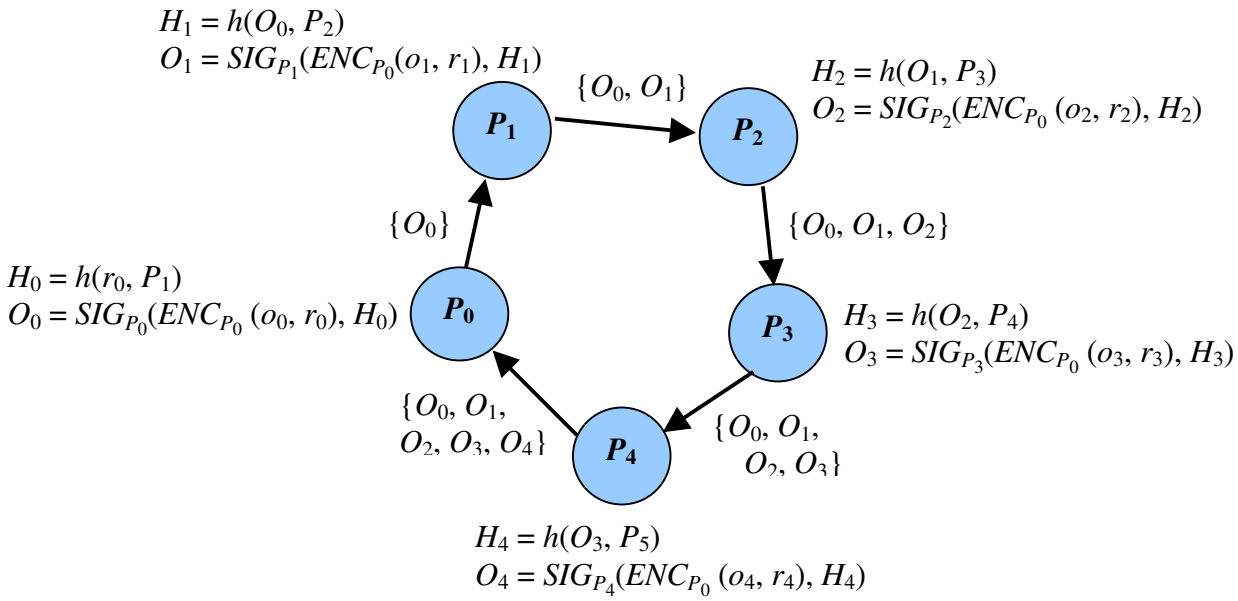


**Figure 3.1. Publicly Verifiable Digital Signature Protocol**

The encapsulated offers are probabilistically encrypted so only the originator can decrypt it. Consisting of the previous encapsulated offer concatenated with the identity of the next platform, the encapsulated offer links the previous offer with the current one. Also, $O_{i\text{-}1}$ cannot be modified without modifying $O_i$. The encapsulation also guarantees that only $P_{i+1}$ can append the next offer.

### 3.1.1. Pitfalls

There are several pitfalls in this protocol. One pitfall is that since the originator receives as a result a set of encapsulated offers, he needs to extract each individual encapsulated offer to find the offers. Since the encapsulated offers are chained, the originator must extract them in order, one after the other. If there are numerous offers, this extraction may become a costly process for the originator.

Another pitfall is that encrypting and then signing a piece of data is questionable. A principle of cryptographic protocols is that one should not infer that the signer knows the content of the encrypted data it is signing [Principle 5, 1]. So, it cannot be inferred that the signing platform is the one who encrypted the offer or even knows of the contents in that encryption (the offer). The platform can simply take someone else's encryption and sign it, claiming the offer to be its own.

In [8], it is explained that PVCDSP has a weakness. An adversary can truncate offers by removing them from the end of the set, build her own chain of offers, and submit them to the mobile agent. This weakness would cause the originator into believing that the mobile agent collected these offers while in fact it never even visited the platform.

Let the adversary be platform $P_{a>1}$. $P_a$ picks a platform $P_i$ who has already submitted an offer. $P_a$ also picks a new $P_{i+1}$. First, $P_a$ replaces the originator's mobile agent with her own and sends it to $P_i$ to collect an offer in order to maintain the validity of the chaining relation at $P_{i-1}$. $P_i$ subsequently sends the mobile agent to the new $P_{i+1}$. $P_{i+1}$ adds his offer to the set, and then sends it to $P_a$. $P_a$ then removes $P_{i+1}$'s encapsulated offer, increments $i$, picks a new $P_{i+1}$, and repeats the process.

For example, in figure 3.2, let $P_4$ be the malicious platform. The message is sent normally from $P_0$ to $P_4$ with $MA\_P_0$, the mobile agent of the originator platform $P_0$. After the malicious platform $P_4$ receives the mobile agent, she picks $P_1$ (where $i = 1$) to be the point from which to start collecting her new offers. And $P_4$ picks a new platform $P'_3$ as the next hop. Then $P_4$ replaces the mobile agent with her own mobile agent, truncates the offers from $P_1$ (it is easy to truncate offers [5]), and sends it to $P_1$. $P_1$ makes his offer and sends the mobile agent to $P'_3$. After $P'_3$ makes his offer, he sends the mobile agent to $P_4$.

$P_4$ now discards the offer from $P'_3$, increments $i$ to 2, picks a new platform $P'_4$, and sends the mobile agent to $P'_3$. $P'_3$ makes his offer and sends the mobile agent to the next new hop $P'_4$, who makes an offer and sends the mobile agent to $P_4$. This process repeats to collect an offer from $P'_5$.

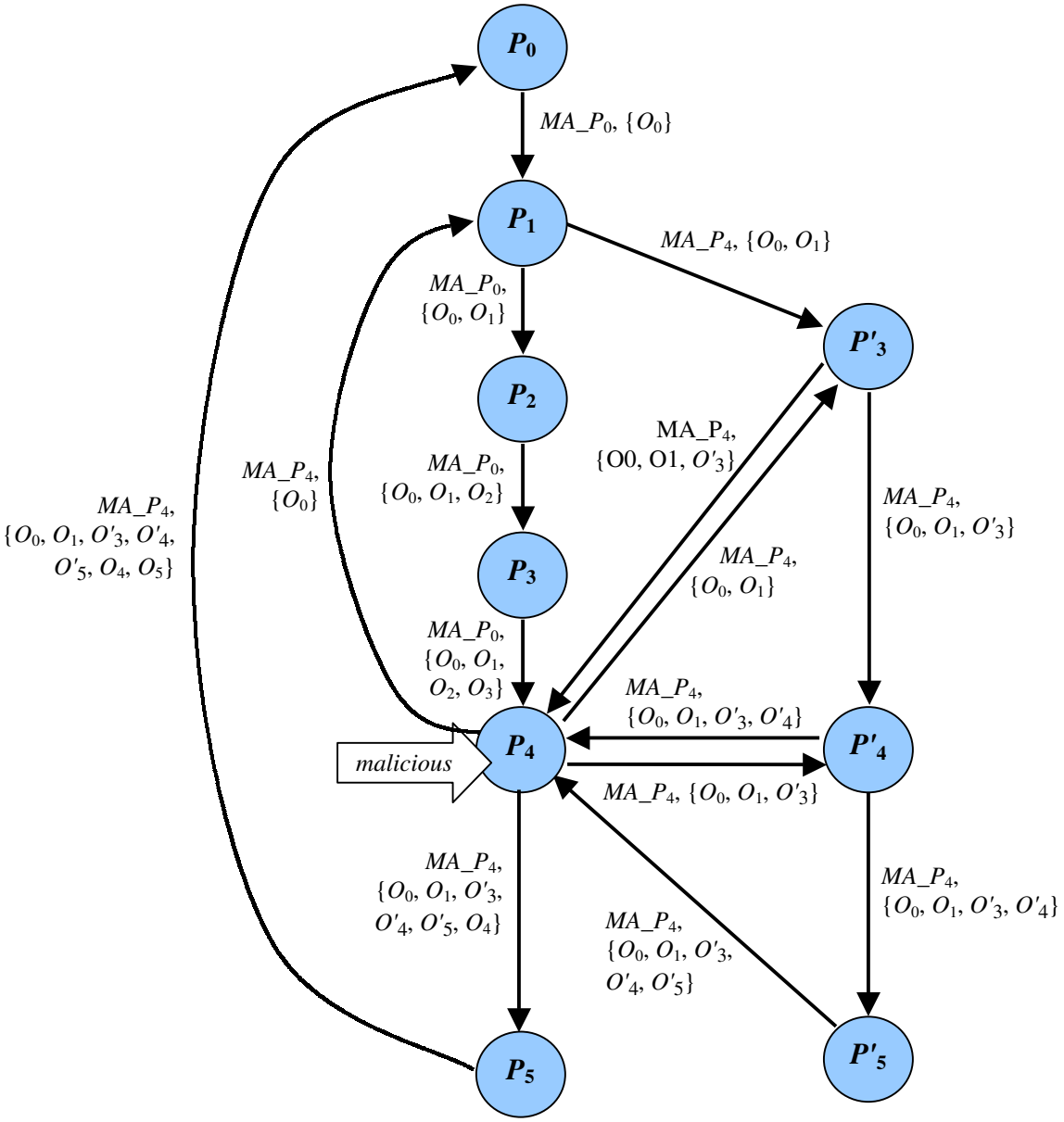


**Figure 3.2. Fault in PVCDSP**

And this collection process can continue until $P_4$ decides that she has collected enough misleading offers. However, for illustrative purposes, we have only collected three misleading offers in our example. Finally, $P_4$ simply pastes the set of misleading offers into $P_0$'s mobile agent, makes her own offer, and sends it to the next platform. $P_4$ can even send the mobile agent back to $P_0$ herself, without sending it to $P_5$. In our example, $P_4$ sends the mobile agent to $P_5$.

The authors of [8] claim that the weakness in PVCDSP is not truncation, as the authors of [5] claim, but the weakness is that a platform can act as an oracle and collect offers to the terms of the adversary instead of the originator. For example, the originator may be searching for properties for rent at the beach, with a preference for beachfront properties. The adversary can modify the mobile agent, so that it only searches for non-beachfront properties, making it seem like the adversary is the only one who offers a beachfront rental.

## 3.2. Execution Tracing

Vigna's execution tracing protocol [9] is based on the use of a mechanism to trace the mobile agent's execution in a way that cannot be forged. These execution traces can be used to verify the traces provided by a platform. They can also be used to prove that they never executed the mobile agent.

Vigna distinguishes between white statements and black statements. White statements are operations where the state is modified only by internal program variables. Black statements, on the other hand, are operations where the state is modified by external information. Depending on whether the statement is white or black, the execution trace $T_C$ of code $C$ contains different information. A trace is comprised of a sequence of pairs $<n, s>$, where $n$ is the unique identifier of the statement and $s$ is the signature. The signature for white statements is empty. The signature for black statements contains the resulting new value.

Vigna discusses three types of mobile agents: remote code execution agents, boomerang agents, and multi-hop agents. A remote code execution agent is one where a program, with some initialization data, is sent to a remote site, who executes the program and returns the results to the originator. A boomerang agent is exactly the same as a remote code execution agent, except that the entire mobile agent is returned to the originator. Finally, a multi-hop agent is one that visits

one or more platforms, whereas the previous two only visit one. Also, with multi-hop agents, only the results are returned to the originator. This thesis is concerned with multi-hop agents.

The execution tracing protocol for multi-hop agents depends on a public key infrastructure and a trusted third party (TTP). It is assumed that the code carried by the mobile agent is static and that there is a tamper-proof, non-repudiable execution trace file.

The protocol (figure 3.3) begins with the originator $P_0$ dispatching the mobile agent to the first platform $P_1$:

$$P_0 \rightarrow P_1: \; P_0, ENC_{P_1}(C, S_0), SIG_{P_0}( h(C), h(S_0), P_1, TTP, t_{P_0}, I )$$

The first field indicates the sender of the message. The second field contains the encryption of the code $C$ and the initial state $S_0$ with the recipient's public key, so only the recipient can decrypt the code and state. The final field is signed with the private key of $P_0$. With the final field, $P_1$ can verify that the code and state were not tampered with and that it was actually sent by $P_0$. Also, this field contains the identity of the recipient, so $P_1$ can verify that he is the intended recipient. From this field, $P_1$ can also know who the trusted third party is. The trusted third party *TTP* and the originator $P_0$ may coincide, or may be different if $P_0$ will be disconnected after dispatching the mobile agent. The timestamp $t_{P_0}$ is used to guarantee freshness and the unique identifier $I$ is used to prevent replay attacks.

Immediately upon receipt of the agent, the first platform $P_1$ sends a receipt message to the trusted third party *TTP*, indicating that he received a mobile agent from the sender and to verify to the TTP that the code and state were received correctly:

$$P_1 \rightarrow TTP: \; P_1, SIG_{P_1}( P_0, SIG_{P_0}( h(C), h(S_0), P_1, TTP, t_{P_0}, I ) )$$

The receipt message contains the identity of the sender in the first field. Signed by $P_1$, the second field contains the identity of the platform from which he received the mobile agent and the entire third field of the message received. With this message, *TTP* can verify that $P_1$ received a mobile agent execution request at time $t_{P_0}$ and that *TTP* is the intended receipt recipient.

Then the platform $P_1$ executes the mobile agent until it decides to migrate to the next platform. Consequently, he sends the trusted third party another receipt:

$$P_1 \rightarrow TTP: \; P_1, SIG_{P_1}( P_2, h(S_1), \mathrm{h}(T_C^1 ), I ) )$$

Again, the first field contains the identity of the sender. The second field is signed by $P_1$, consisting of the identity of the next hop $P_2$, a hash of the current state $S_1$, hash of the execution trace $T_C^1$, and the unique identifier $I$.

And then the mobile agent is sent to the next platform:

$P_1 \rightarrow P_2$: $P_1, P_0, ENC_{P_2}(C, S_1), SIG_{P_1}(h(S_1), P_2), SIG_{P_0}(h(C), h(S_0), P_1, TTP, t_{P_0}, I)$

This message is similar to the first message in this protocol except for the addition of the second and fourth fields. The second field indicates the identity of the mobile agent originator $P_0$ so that $P_2$ knows whose public key to retrieve for decrypting the last field. The fourth field is used to verify the intended recipient $P_2$, the message sender $P_1$, and the state $S_1$. If $P_2$ is able to decrypt the fourth field with the public key of $P_1$, then he has verified that $P_1$ is the sender. $P_2$ can verify the state by hashing the state given in the third field and comparing to that in the fourth field.

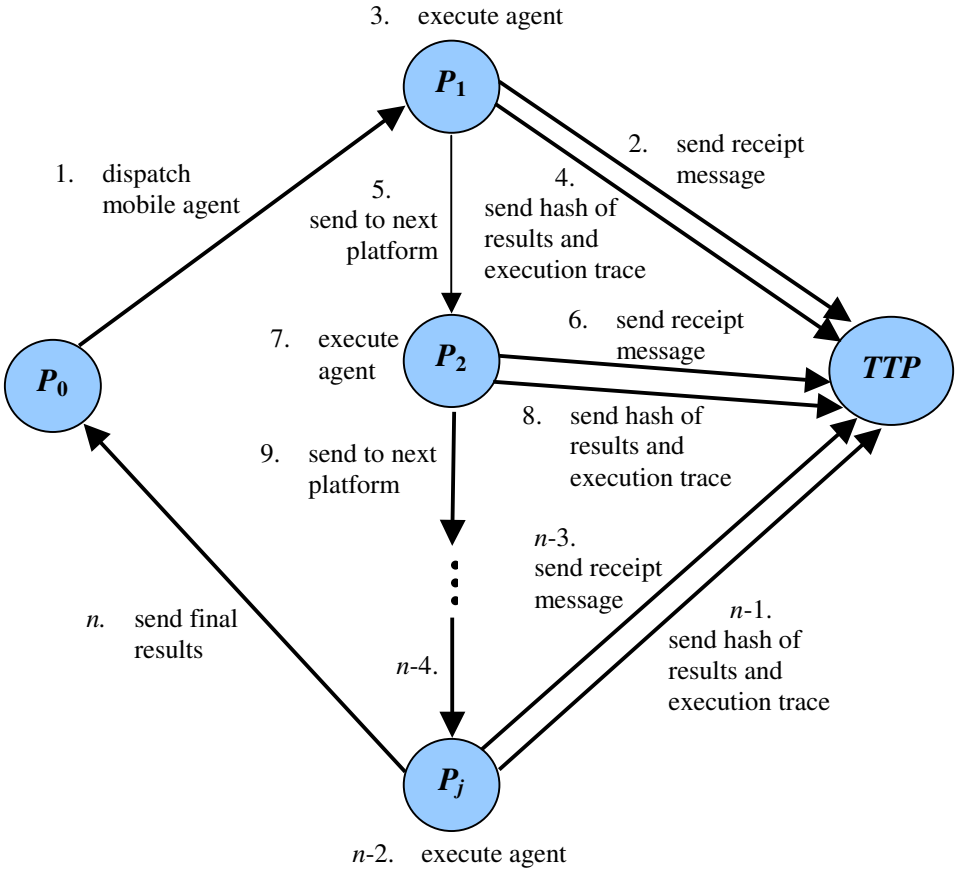


**Figure 3.3. Execution Tracing Protocol**

After receiving of the above message, $P_2$ sends a receipt to the trusted third party:

$P_2 \rightarrow TTP$: $P_2$, $SIG_{P_2}( SIG_{P_1}(h(S_1), P_2), SIG_{P_0}( h(C), h(S_0), P_1, TTP, t_{P_0}, I ) )$

This message is similar to the receipt message that $P_1$ sent to *TTP*, except that $P_2$ also needs to sign the fourth part of the message he received to verify that the message was sent by $P_1$.

This protocol continues until the last platform *Pj* is reached. The last platform sends the trusted third party a receipt of the results. This receipt, unlike the previous ones does not include the identity of the next hop since $P_j$ is the last one. $P_j$ then sends the originator the final state, which contains the resulting data encrypted with the public key of $P_0$:

$P_j \rightarrow TTP$: $Pj$, $SIG_{P_j}( h(S_j), h(T_C^{\,j}) ), I )$

$P_j \rightarrow P_0$: $P_j$, $SIG_{P_j}( ENC_{P_0}(S_n), I )$

The originator examines the results and if he suspects any platform cheated, he can ask the trusted third party for all the receipt messages and ask each platform for their execution traces. With this information, the originator then proceeds to simulate the agent execution by following the order of the execution traces. Each of the resulting states should match the hash result in the receipt message.

### 3.2.1. Pitfalls

Many pitfalls exist in this protocol. First of all, there is a dependency on the trusted third party. The integrity of the messages depends entirely on the trusted third party, so if the TTP were to fail, there would be no way to check the integrity of the messages. Therefore, the trusted third party is a single point of failure where the entire process fails if it shuts down.

Another pitfall is that the messages sent are not uniform. The initial message sent from the originating platform to the first platform is different from the subsequent messages sent from one platform to the next. Also, the receipt message that the first platform sends is different from the receipt message subsequent platforms send. So, a platform must know if he is the first platform to receive the mobile agent. The only way a platform can tell is via the message sent from the previous platform. The final two messages sent from the last platform are also different from previous messages of the same function.

An inefficiency in this protocol is due to the last field of the initial message that is repeated in almost all the subsequent messages. While it is important to include this information in subsequent messages, it is not clear that all the components of this field are needed.

In addition, the fact that the timestamp stays the same throughout the life of the mobile agent illustrates another weak point. The timestamp provided by the originator and only indicates freshness of the initial message. There is no way to tell if subsequent platforms are receiving messages in a timely manner.

Finally, the verification process is expensive. If there is suspicion of tampering with the data, then the originator has to simulate the entire mobile agent execution. Plus, the execution must be followed in the correct order. This is a taxing process because if the suspicion rate is high, then the originator will always have to simulate the mobile agent execution. In that case, the originator may as well have executed the mobile agent himself.

# CHAPTER 4

# THE ENCAPSULATED OFFER PROTOCOL

The encapsulated offer protocol (EOP) is a mobile agent protocol with mechanisms implemented to protect the mobile agent and its data. EOP applies to mobile agents for personal digital assistants, where the ideal paradigm is comparison-shopping. For example, a user dispatches a mobile agent, and the mobile agent hops from platform to platform collecting offers.

In this chapter, we describe our encapsulated offer protocol, which uses data encapsulation and execution tracing. EOP relies on a public key infrastructure and does not allow updating of previously submitted offers. For the class of mobile agents we address, the code and state are static, as we are dealing with a comparison-shopping model. Thus, this makes the data state separate from the execution state.

Each platform has a unique identifier. With the identifier a platform can retrieve the public key for the platform identified. This key can either be stored at each platform, or can be distributed by a certification authority. The identity of the signer must be explicitly stated in a message, otherwise the examiner of the signature cannot determine who the signer is.

Each platform also maintains a tamper-proof, non-repudiable execution trace file on their system. This file is used to solve disputes between the platform who provided the service and the originator. Provided that the platform executes a mobile agent's code non-maliciously, it always provides the correct answer.

## 4.1. Overview of the Encapsulated Offer Protocol

The EOP is modeled after Vigna's multi-hop agent protocol [9]. Each message in EOP is identical, so platforms do not necessarily need to recognize that he is the first, or the last, platform to receive the mobile agent. In addition, the message is optimized so that each field and

its components serve a purpose for the platform who receives the message. The message and the meaning of each part are explained in the next section.

In EOP, the trusted third party is not necessary, thus eliminating the keystone dependency and the overhead of sending receipt messages each time a message is received and after code execution. However, without the trusted third party, the integrity check mechanism is removed. Data encapsulation is introduced into the protocol to control data (and sometimes code and state) integrity. With data encapsulation, it is easiest to separate the state and results (offers) into different entities. This means that the state information will not change, so its hash value is included with the hash of the code in the fifth field.

## 4.2. The Parts of a Message in EOP

A message in EOP contains five fields: 1) the sender identity, 2) the originator identity, 3) the encryption of the code, state, and offers, 4) the signature of the receiving platform identity and timestamp, and 5) the signature of the hash of the code and state and the unique session identifier. A message from the sender $P_i$ to the receiver $P_{i+1}$ is formatted as in figure 4.1.

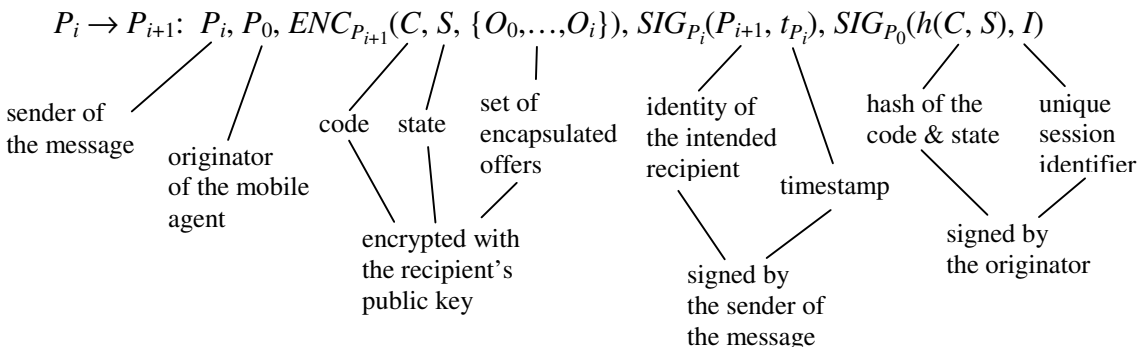


**Figure 4.1. Description of the fields of a message**

The first two fields $P_i$ and $P_0$ indicate the identities of the message sender and mobile agent originator, respectively. Since the identity of the signer cannot be deduced by examining

the signature, these two fields are needed to determine whose public keys need to be retrieved to decrypt the fourth and fifth fields.

The third field, containing the code $C$, state $S$, and set of encapsulated offers $\{O_0,\ldots,O_i\}$, is encrypted with the recipient platform's public key. With this encryption, only the recipient can decrypt this field with his secret key and access the code, state, and offers. This encryption prevents unauthorized third parties from attempting to execute the mobile agent. The offers are also hidden from an adversary, but even a valid platform still cannot extract another platform's offer due to the signature on the encapsulated offers.

The fourth field is signed by the sender of the message to verify the sender. This field contains the identity of the recipient $P_{i+1}$ and a timestamp $t_{P_i}$. The recipient examines this field and checks if the message was intended for him. Also, the recipient uses the timestamp to verify the freshness of the message. The timestamp should contain the time the message is sent and the duration of how long the message is valid.

The fifth and last field contains the hash of the code and state and the unique session identifier $I$. This field is signed by the originator to prevent tampering with this information and for platforms to verify the integrity of the code and state. The unique session identifier links all the messages in the same session together.

### 4.2.1. The Encapsulated Offers and Chaining

We model our encapsulated offers after that used in the publicly verifiable chained digital signature protocol [5], which also relies on a public key infrastructure. A hash function is used for creating the encapsulated offers. The hash function must be one-way and collision-free.

Since one should not infer that the signer knows the contents of the encrypted data it is signing, we eliminate the signature on the offer:

$$O_0 = SIG_{P_0}(o_0, h(r_0, P_1))$$

$$O_i = SIG_{P_i}(o_i, h(O_{i\text{-}1}, P_{i+1}))$$

It may seem that now the set of offers can be verified by any platform who receives the mobile agent. However, since the identities of the signers are not provided, platforms do not have the ability to see each other's offers. Only the first and last offers in the set can be verified since their identities are provided in the message, and those are the only offers that need to be checked.

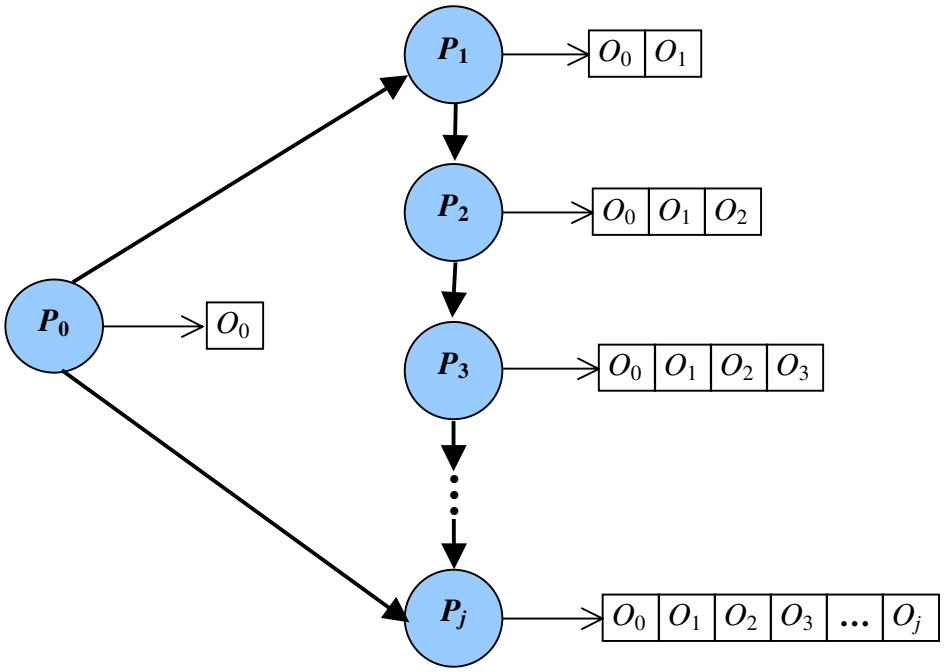


**Figure 4.2. Each platform appends his offer to the set of encapsulated offers**

The initial encapsulated offer $O_0$ is slightly different from that of the others. The originator creates an empty initial offer $o_0$ to include in his encapsulated offer $O_0$. It is empty because he is not offering anything, as he is the originator. Also, the originator generates a random number to use in the hash function because there does not exist a previous encapsulated offer. The originator must keep the random number $r_0$ and the identity of the next platform $P_1$ secret. If these values become known to an intermediary platform, they can attempt at tampering with the chain. Therefore, the strength of the chain depends on the knowledge of $r_0$ and $P_1$.

The set of encapsulated offers utilizes the chaining mechanism (see figure 4.3). This chaining mechanism prevents platforms from maliciously removing or replacing any of the encapsulated offers. Each encapsulated offer $O_i$ computed by $P_i$ contains a chaining relation, which is the encapsulated offer $O_{i-1}$ received from the previous platform hashed with the identity of the next platform $P_{i+1}$. The previous encapsulated offer $O_{i-1}$ is used in computing the new encapsulated offer $O_i$ to link the new encapsulated offer to the previous one. Also, the inclusion of the identity of the next platform $P_{i+1}$ in the chaining relation ensures that only the intended recipient can make the next offer.

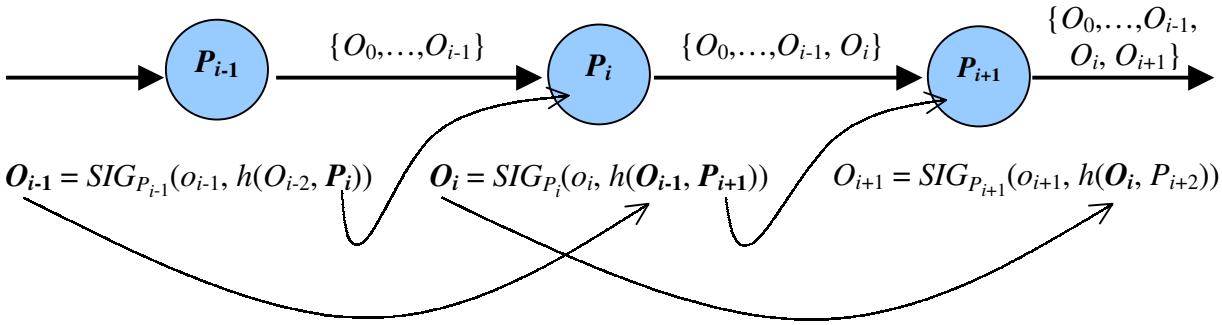


**Figure 4.3. Chaining in the encapsulated offers**

## 4.3. Protocol Description

EOP allows mobile agents to collect data (or offers) from various platforms. There are three distinct actions taken in EOP. First, the originator creates the initial message and dispatches it. Next, the message is received by intermediate platforms, who verify the integrity of the message, execute the code, and submit an offer to the mobile agent. Finally, the originator receives the results and processes them.

### 4.3.1. Creating and Dispatching the Initial Mobile Agent

To begin the protocol, the originator $P_0$ prepares the first message by creating an initial encapsulated offer $O_0$. To create $O_0$, the originator generates a secret random number $r_0$, decides whom the first recipient $P_1$ will be, and creates an initial offer $o_0$ (which is empty). After hashing $r_0$ and $P_1$, the originator then signs the initial offer and the hash value:

$$O_0 = SIG_{P_0}(o_0, h(r_0, P_1))$$

$P_0$ then fills in the rest of the message, as defined in section 4.2, and sends it to the first recipient.

### 4.3.2. Receiving a Mobile Agent

Upon receipt of a mobile agent message, the platform first performs a series of checks before proceeding with the execution (figure 4.4). The first check is to compare the identity in

the second field with its own. If they are the same, then it means that the platform is the originator of the mobile agent. In this case, the platform proceeds with processing the results. If

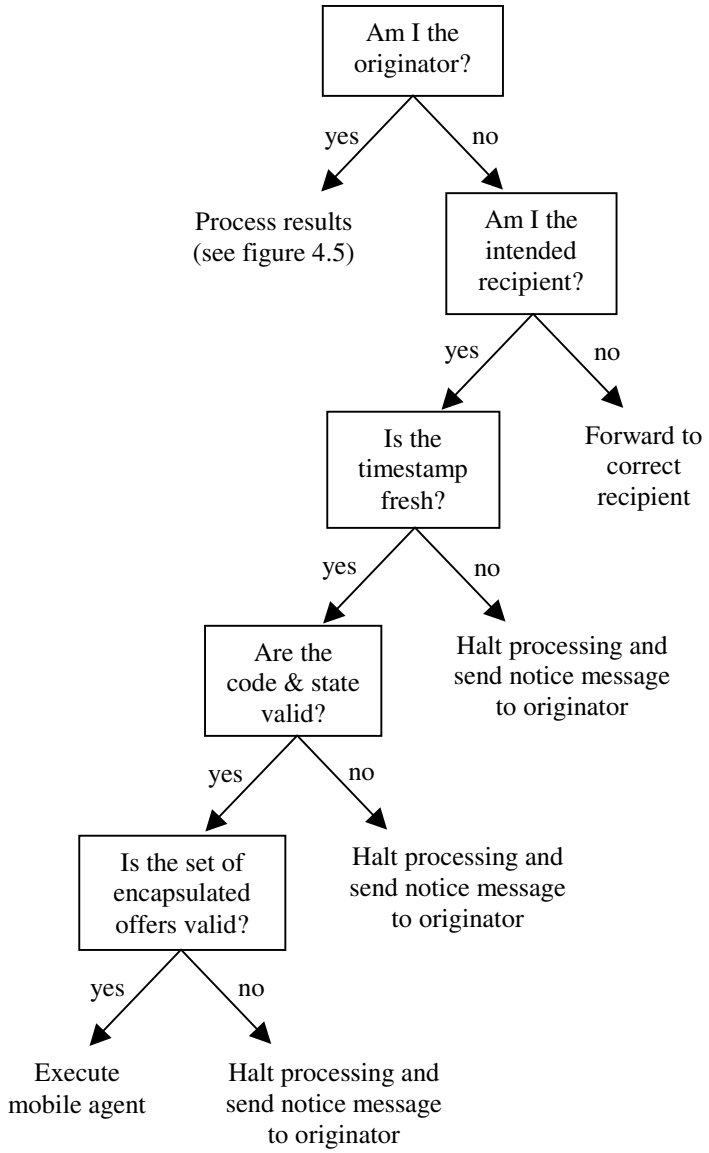


**Figure 4.4. Checking process completed by an intermediate platform**

the platform is not the originator, then it continues with the next check of verifying that he is the intended recipient of the mobile agent. If not, then he simply forwards the agent to the correct recipient or some other platform who can forward it to the correct platform. Next, the platform

checks the timestamp for freshness. If the message has expired, then the platform halts any further processing and sends a notice message to the originator (see section 4.3.5 for details about the notice message). Then the platform checks the integrity of the code and state by computing the hash value and comparing it against the hash value given in the fifth field of the message. If the hash values do not match, the platform halts the processing and sends a notice message to the originator. Finally, the platform checks the validity of the encapsulated offers. He should be suspicious of the set of encapsulated offers if one or both of the following occurs: 1) the originator's public key does not correctly verify the first encapsulated offer, or 2) the public key of the message sender does not correctly verify the last encapsulated offer in the set. If either one or both of these encapsulated offers do not verify correctly, then the platform halts the execution and sends a notice message to the originator.

If all of the above conditions are verified, then the platform proceeds to execute the mobile agent's code. This execution produces an offer, though if no offer can be made, an empty offer indicating that the requirements cannot be met is created. The platform must submit an offer to the mobile agent in order to maintain the chaining in the set of encapsulated offers. Upon completion of code execution the next recipient of the mobile agent is determined. Next, the platform creates the encapsulated offer by first taking the hash of the previous encapsulated offer and the identity of the next platform, and signs this hash value together with its offer:

$$O_i = SIG_{P_i}(o_i, h(O_{i-1}, P_{i+1}))$$

This encapsulated offer is then appended to the current set. The platform logs the execution trace, makes the appropriate signature and encryption updates in the message, and sends the mobile agent to the next platform.

### 4.3.3. Receiving and Processing the Results

When the originator $P_0$ receives his mobile agent, he performs a series of checks (figure 4.5). Like the previous platforms, $P_0$ checks the validity of the timestamp, integrity of the code and state, and validity of the first and last encapsulated offers. If any of these do not verify, then the any one or more of the offers may have been tampered with. If everything verifies, $P_0$ can unravel the chain of encapsulated offers with its secret random number and the identity of the first platform.

Should $P_0$ suspect tampering with any of the offers, he can ask the respective platform to provide the execution trace. The execution trace contains a copy of the encapsulated offer that it submitted to the mobile agent. $P_0$ compares the encapsulated offer that it received against that in the execution trace. If the offers match, then it has not been tampered with. If they do not match, then discard the offer, and the originator may report the incident to an authority.

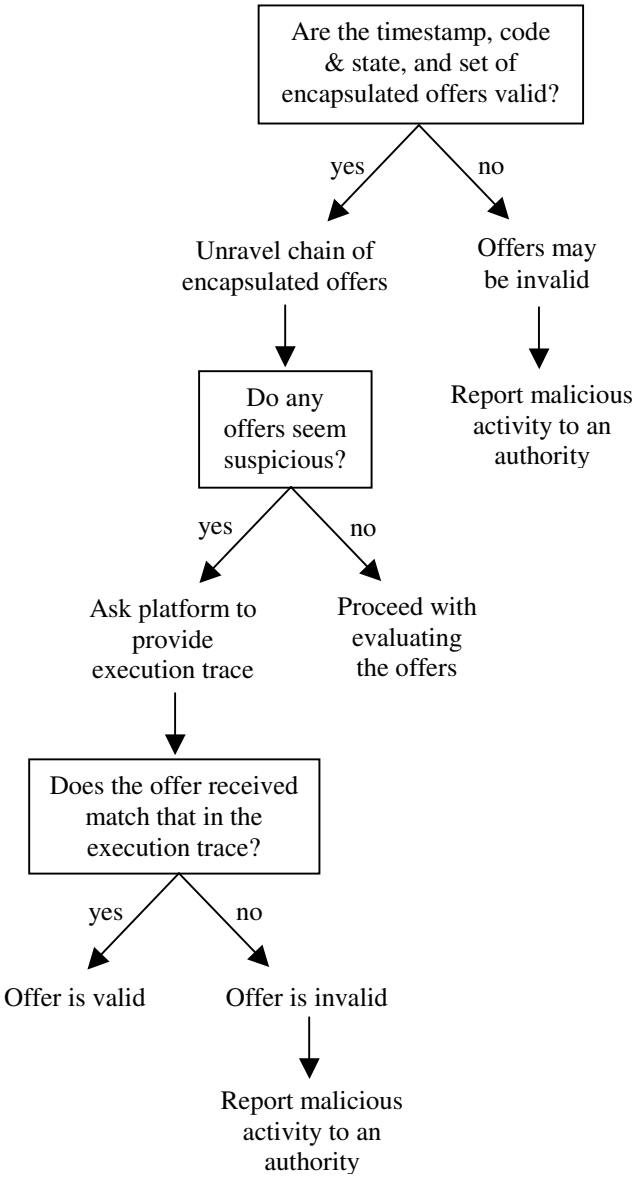


**Figure 4.5. Checking process completed by originator upon receipt of results**

### 4.3.4. Execution Trace File

The execution trace file is maintained by each platform. This file must be tamper-proof and non-repudiable, so that the originator can verify that a platform made an offer. In EOP, an execution trace contains a triplet $< I, O, t >$, where $I$ is the unique session identifier, $O$ is the encapsulated offer submitted by the platform, and $t$ is the timestamp.

### 4.3.5. Notice Message

A notice message is a message sent to the originator indicating that there has been tampering with his mobile agent. There are three types of notices: expired timestamp, invalid code and state, and invalid or broken chain of encapsulated offers. The message is flagged to indicate that it is a notice message, concatenated with the type of notice, and followed with the message the platform received, signed.

| | | |
|---|---|---|
| notice message | type of notice | $SIG_{P_i}( P_{i-1}, P_0, ENC_{P_i}(C, S, \{O_0,\ldots,O_{i-1}\}), SIG_{P_{i-1}}(P_i, t_{P_{i-1}}), SIG_{P_0}(h(C, S), I) )$ |

**Figure 4.6. Notice Message**

## 4.4. Security Evaluation

In this section we will evaluate the security of our protocol. First, we evaluate the security of the code and state. Then, we evaluate the security of the encapsulated offers. Next, the security properties of EOP are defined. Finally, other security concerns are covered.

### 4.4.1. Security Evaluation of the Code and State

Modification of the code and state can be detected in EOP. If a malicious platform intends to modify or replace the code and state, he will also need to update the hash value in the

fifth field. Having modified the fifth field, the malicious platform needs to sign it with his own private key, as he cannot sign with the originator's private key. Now having changed the signature, the malicious platform also needs to replace the identity of the originator (second field) with his own identity.

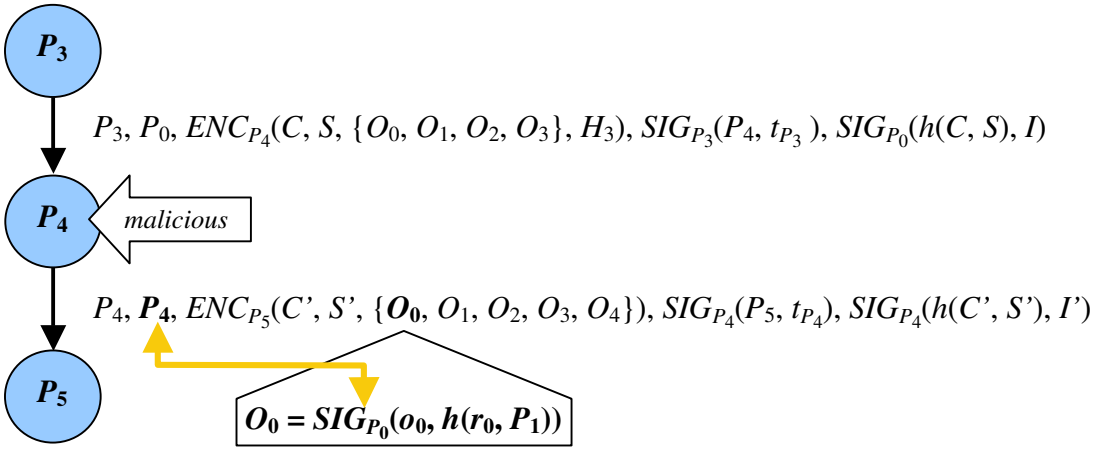


**Figure 4.7. Cannot modify code and state**

However, the receiving platform can detect this tampering because the encapsulated offer $O_0$ remains signed by the originator, as illustrated in figure 4.7. Upon verifying the first encapsulated offer $O_0$ with the originator's public key, the receiving platform can deduce that the sender was malicious. To avoid this problem, the malicious platform can simply strip off the originator's signature on $O_0$ and sign it himself. This would result in the mobile agent never returning to the originator $P_0$. This is a denial of service attack, which is a recurring issue in information security and not addressed in this thesis.

### 4.4.2. Security Evaluation of the Encapsulated Offers

We explained in section 3.1.1 that the PVCDSP has a weakness. A malicious platform can truncate the set of encapsulated offers, modify or replace the code and state, build a new chain of encapsulated offers with the new code and state, and append the new encapsulated offers to the truncated set of encapsulated offers. This attack would lead the originator into believing that his mobile agent collected these offers while in fact it never even visited the

platforms. An important step in this attack is that the malicious platform needs to pick a platform $P_i$ who has already made an offer from which to truncate. In EOP, however, the identity of the signer cannot be revealed by examining the encapsulated offer, so this attack cannot occur.

A malicious platform $P_a$ can attempt an attack similar to that described above. Rather than truncating the set of encapsulated offers, $P_a$ can simply create a new mobile agent with a modified state. For creating the initial encapsulated offer, instead of generating a random number, $P_a$ uses the value of the last encapsulated offer in the set she received. Using the last encapsulated offer allows $P_a$ to maintain the chaining mechanism after appending the new set to the existing one later. With her mobile agent, $P_a$ collects new offers. Upon return of the mobile agent, $P_a$ can append the set of offers she just collected to the set in the originator's mobile agent. $P_a$ also needs to submit her offer again to maintain the chaining mechanism. Then $P_a$ can either forward the mobile agent to other platforms or send it back to the originator. $P_a$ is successful in executing this attack; however, there is a subtlety that can be detected. When the originator unravels the chain, he can see that $P_a$ has offered twice and should suspect $P_a$'s two offers and all offers between them.

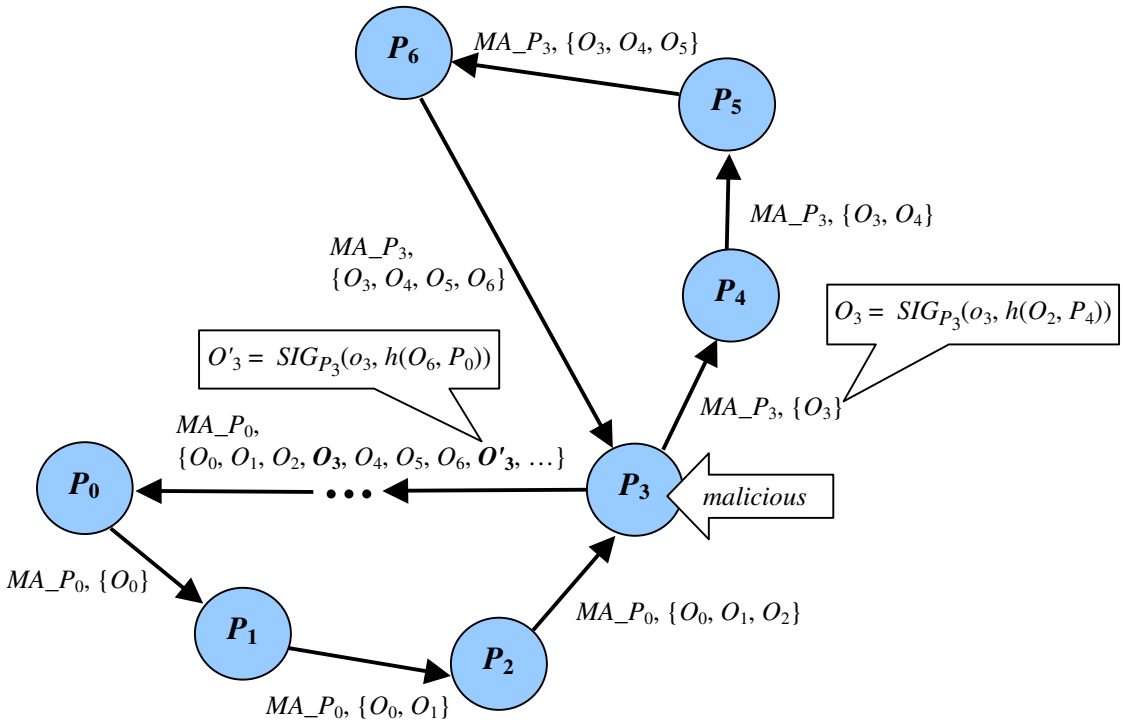


**Figure 4.8. Cannot collect and append offers**

For example (figure 4.8), Andy ($P_0$) sends out a mobile agent $MA\_P_0$ seeking the best deal on a new car. He specifies in his mobile agent that he is looking for a red or silver sports car. $P_1$ and $P_2$ submit their offers normally. When the malicious dealership $P_3$ receives Andy's mobile agent, she holds it and sends out her own mobile agent $MA\_P_3$ to collect offers only for red sports cars. To create her initial encapsulated offer, she uses the last encapsulated offer $O_2$ she received in $MA\_P_0$ as her random number. After her mobile agent returns, she can easily append the newly collected offers to the set in Andy's mobile agent $MA\_P_0$. Note that this appending maintains the chaining mechanism due to the use of $O_2$ in her initial encapsulated offer. Finally, $P_3$ can either forward Andy's mobile agent (with offers collected by $P_3$) to another platform or back to Andy. After Andy receives the results and unravels the chain of encapsulated offers, he can see that $P_3$ made two offers. Therefore, Andy should be suspicious of $P_3$'s two offers and any offers between them.

In addition, truncation of the set of encapsulated offers can also be detected in EOP. The last encapsulated offer in the set must be verifiable with the sender's public key, as illustrated in figure 4.9. If it does not verify, the recipient of the message can deduce that the sender had truncated the set of encapsulated offers. Moreover, a malicious platform cannot add his own encapsulated offer to the truncated set because doing so violates the chain (figure 4.10). The message recipient cannot detect a broken chain, but when the originator tries to unravel the chain he will discover the broken link.

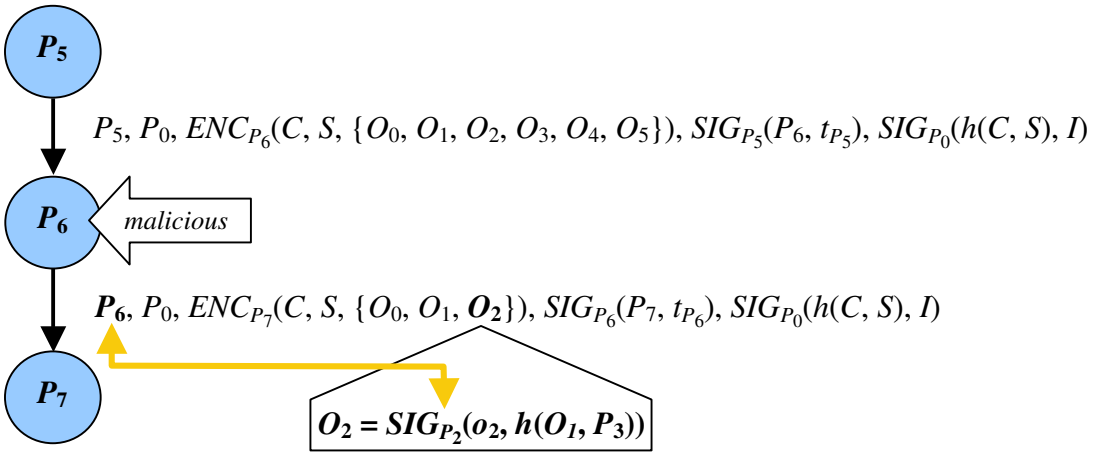


**Figure 4.9. Cannot truncate the set of encapsulated offers**

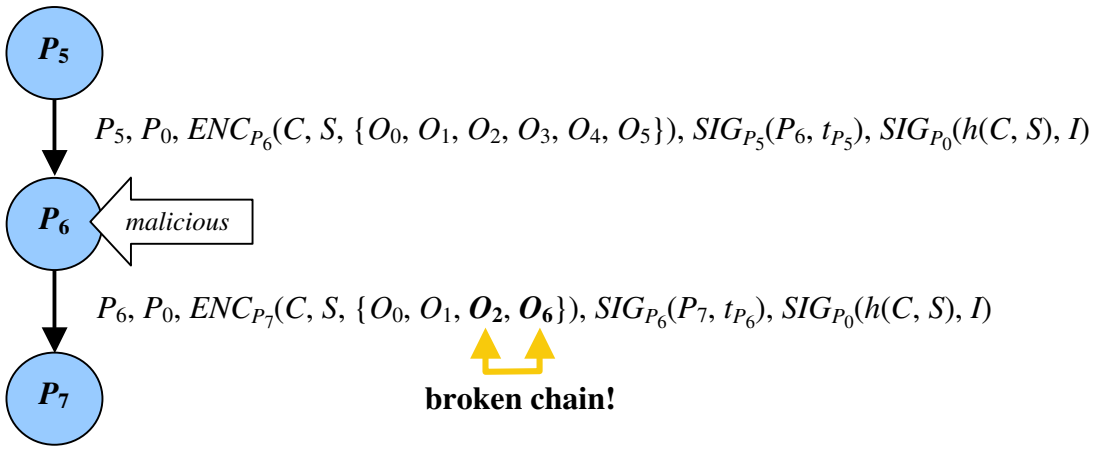


**Figure 4.10. Cannot insert an offer after truncation**

Similar to truncation, a malicious replacement of the entire set of encapsulated offers can be detected. If there is only one encapsulated offer in the set and it does not verify with the originator's public key, then the recipient of the message can conclude that the sender has replaced the entire set of encapsulated offers with the set only containing his offer.

### 4.4.3. Security Properties

A number of security properties in regards to an attacker who captures a mobile agent with a set of encapsulated offers $\{O_0, O_1, \ldots, O_j\}$ are defined below. Suppose that the agent has visited $j$ platforms, and let $i$ be in the range $1, \ldots, j$.

**Data Confidentiality.** An offer $o_i$ cannot be extracted by platforms other than the originator $P_0$ and the platform $P_i$ who made the offer because the identities of the signers are unknown. However, offers $o_0$ and $o_j$ can be extracted because their signatures are provided in the message.

**Data Integrity.** An offer $o_i$ cannot be modified by any platform due to the chaining mechanism in the encapsulated offers.

**Truncation Resilience.** The encapsulated offers cannot be truncated due to the signatures of the sender and the final encapsulated offer.

**Insertion Resilience.** A malicious platform cannot insert an offer into the set of encapsulated offers (either at the beginning, middle, or end) due to the chaining mechanism.

**Modification Resilience.** A malicious platform cannot modify one or more of the encapsulated offers due to the chaining mechanism. Also, if $O_0$ is modified, then the originator can detect this change due to his secret random number and knowledge of the first platform.

**Non-repudiability.** A platform $P_i$ cannot claim that he did not provide an offer $o_i$ due to the signature on the encapsulated offer and the execution trace.

**Partial Forward Privacy**. None of the identities $P_i$ can be extracted from the encapsulated offers, except the originator $P_0$ (first encapsulated offer) and message sender $P_j$ (last encapsulated offer).

### 4.4.4. Other Security Concerns

Assuming that the identity of the signer cannot be extracted from the signature unless explicitly stated is not a cryptographically strong assumption. If there are only few platforms in the system, say fifty platforms, an attacker can easily test each identity to find the signer of an encapsulated offer. However, this assumption is practically strong because in reality it is more likely much more than fifty platforms will coexist on the same network. With the large amount of platforms, it becomes infeasible for the attacker to test each identity.

# CHAPTER 5

# CONCLUSION

In this thesis we have proposed a protocol, the encapsulated offer protocol, for protecting mobile agents and their data. The encapsulated offer protocol is based on two existing protocols, the publicly verifiable chained digital signature protocol and the execution tracing protocol. Many pitfalls were discovered in the publicly verifiable chained digital signature protocol and the execution tracing protocol. By combining and modifying these two protocols to create the encapsulated offer protocol, we have eliminated most of the aforementioned pitfalls.

EOP ideally applies to a comparison-shopping paradigm, where the mobile agent migrates from platform to platform and collects offers. The encryption, signature, and data encapsulation mechanisms protect the mobile agent code, state, and data from being tampered with.

## 5.1. Future Work

In this thesis we have presented a mobile agent protocol that includes features to control the integrity of the mobile agent and its components, in particular the code, state, and data.

Future work can include implementation and testing for the encapsulated offer protocol. The efficiency of EOP can be compared against the efficiency of existing protocols. It may result that EOP is not as efficient as some other protocols. If that is the case, the advantages of the security mechanisms in EOP need to be weighed against its inefficiency.

Furthermore, EOP can be modified to adapt to a non-public key infrastructure environment. The platforms involved in the mobile agent execution need to exchange secret keys prior to dispatching the mobile agent. Or, a platform can exchange keys with the originator upon receipt of a mobile agent.

Currently, EOP only applies to mobile agents for personal digital assistants, or specifically for collecting offers from various platforms. Future work can extend EOP to enable operation for various types of paradigms.

In this thesis we have only covered attacks from one malicious platform. However, it is also possible that two or more platforms may collude to perform an attack. One such case is the "shortcut attack" [6]. For example, the mobile agent had already visited platforms $P_1$, $P_2$, $P_3$, $P_4$, $P_5$, $P_6$, $P_7$, and $P_8$ in that order and just arrived platform $P_9$. If platform $P_9$, who is malicious, suspects that any of the previous offers are better than his, he can collude with his conspirator $P_3$ and remove offers $P_4$ through $P_8$, inclusive. To do this $P_9$ sends the mobile agent to $P_3$. $P_3$ in turn truncates the set of encapsulated offers from his offer $O_3$, leaving the set $\{O_0, O_1, O_2\}$. Then $P_3$ creates a new encapsulated offer, this time with $P_9$'s identity, and sends the mobile agent to $P_9$. $P_9$ can then submit his offer and forward the mobile agent to another platform or back to the originator. Doing this a "shortcut" is taken from $P_3$ directly to $P_9$, bypassing all the platforms in between. Collusion, such as the one described above, should be considered in future work.

# REFERENCES


[1] Martín Abadi and Roger Needham. "Prudent Engineering Practice for Cryptographic Protocols." *IEEE Transactions on Software Engineering*, 22(1): 6-15. January 1996.

[2] Whitfield Diffie and Martin Hellman. "New Directions In Cryptography." *IEEE Transaction on Information Theory*. IT-22/6: 644-654. November 1976.

[3] Fritz Hohl. "Mobile Agents and Active Networks." *Proceedings of the IFIP Fifth International Conference on Intelligence in Networks (SMARTNET '99)*. Pathumthani, Thailand. November 1999.

[4] Wayne Jansen and Tom Karygiannis. "NIST Special Publication 800-19 – Mobile Agent Security." 2000.

[5] G. Karjoth, N. Asokan, C. Gülcü. "Protecting the computation results of free-roaming agents." *Proceedings of the Second International Workshop, Mobile Agents 98*. Springer-Verlag Lecture Notes in Computer Science, vol. 1477, pages 195-207. 1998.

[6] P. Kotzanikolaou, M. Burmester, V. Chrissikopoulos. "Secure transactions with mobile agents in hostil environments." Information Security and Privacy, ACISP 2000. Springer-Verlag Lecture Notes in Computer Science, vol. 1841: 289-297. Berlin. 2000.

[7] Sergio Loureiro, Refik Molva, Alain Pannetrat. "Secure Data Collection with Updates." *Electronic Commerce Research Journal*. 1/2: 119-130. February/March 2001.

[8] Volker Roth. "On the Robustness of some Cryptographic Protocols for Mobile Agent Protection." Proceedings of Mobile Agents 2001. Springer-Verlag Lecture Notes in Computer Science, vol. 2240. December 2001.

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

# BIOGRAPHICAL SKETCH

Anna Suen was born in Atlanta, Georgia on May 26, 1979. Anna graduated with a bachelor's degree in Computer Science from Florida State University in 2001. Upon graduation, she began working on her master's degree in Computer Science, specializing in Information Security. Anna served as Secretary of FSU's chapter of the Association of Computing Machinery during her senior year of her undergraduate career and Chair during her second year of graduate school. Anna is also a member of the Phi Eta Sigma National Honor Society, Golden Key National Honor Society, and Upsilon Pi Epsilon Computer Science Honor Society. During her spare time, Anna enjoys working on her personal website, spending time with family and friends, and relaxing on the beach.