Securing Mobile Agent in Hostile Environment

BY

MO CHUN MAN

A THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF PHILOSOPHY

IN INFORMATION ENGINEERING

JUNE 2001


香港中文大學
大學
圖書館系統
22 APR 2002
UNIVERSITY
THE CHINESE UNIVERSITY OF

Acknowledgement

First of all, I have to express my hearty thanks to my supervisor Prof. Keh-wei Wei for his guidance on my research. To be his student, I have benefited from his deep and broad knowledge, not only on my research field, but also on other facets. He has always inspired me with new ideas that are profound. I am also very grateful for his patience in improving my writing and presentation skills.

I am very grateful to Mr. Ivan Ng who suggested ideas to me and showed constant concern on my thesis. I cannot forget his solid help on my paper writing by spending his precious time on reviewing my paper and giving advises.

I also want to give special thanks to Sandy and Joseph who gives me support in my research work. Thanks to their patience in listening to me and attempt in helping me. Thanks also to Joseph who gives valuable advises on my thesis writing.

I cannot forget all the people who shared the master experience with me.

I would like to express my deepest gratitude to my family that gives me unconditional encouragement and support in the last two years.

Abstract

Mobile agent paradigm is an evolving distributed computing paradigm. Different from traditional paradigms, mobile agent paradigm introduces autonomy, mobility and customization into the paradigm. Many applications can be benefited from these properties. For example, mobile computing is limited by its intermittent network connection and limited bandwidth. It can be benefited from the mobility of mobile agent to finish tasks asynchronously, and thus saves bandwidth and reduces network latency.

Security issues are the most important part in implementing the mobile agent system. It decides whether mobile agent system can be broadly deployed or not. Both agent host and mobile agent can suffer from attacks. Comparing the two, the latter is more difficult to prevent. Although there are many protection schemes that have been proposed in the literature, there is still no satisfactory general solution.

This thesis focuses on investigating the protection schemes of mobile agent. Firstly, we give a survey on existing protection schemes with classification and analysis. Then, we propose taxonomy of various attacks that may be suffered by

mobile agent from its hostile environment. This taxonomy is useful for our further analysis on the possibility and efficiency of various protection schemes. Lastly, we propose a reactive mobile agent model. It has the property to communicate to hosts interactively. That means host can also get plaintext result from the agent. We will illustrate in detail how this kind of agent can be protected from various kinds of attacks.

摘要

移動代理模式是一正在演進中的分佈式計算模式。它與傳統計算模式的不同之處在於其加入了自主性，移動性以及客戶化。許多應用都能從這些特性中獲益。例如移動計算通常都受制於有限的頻寬和間歇性網絡連接。移動代理的移動性能使任務非同步地完成，從而節省頻寬，也減少了等待時間。

安全問題是建立移動代理系統最重要的一環。它對移動代理系統能否得到廣泛應用起了決定性的作用。支持移動代理的主機和移動代理兩者都有受到攻擊的可能性。兩者相比，後者更難得到防範。雖然研究人員已提出不少方案去保護移動代理，但仍未有一滿意且具統一性的方案。

這篇論文會集中探討移動代理的保護方案。首先，我們會俯瞰現存的保護方案，把它們加以分類及剖析。然後，我們提出了來自惡意環境的各種攻擊的分類法。這個分類法有助於我們進一步分析各種保護方案的可行性和有效性。最後，我們也提出了一具反應性的移動代理計算模型。它具有與主機交互式溝通的特性，也就是主機也可以從移動代理得到部分結果。我們會詳細解釋怎樣去保護具有此特性的移動代理及防範各種可能的攻擊。

Contents

Chapter 1

Introduction

The exponential growth of Internet has led us to a new Internet Century. Many services and activities such as information dissemination, entertainment or even electronic commerce have been moved to Internet. At the same time, fast evolving network and computer technology will lead people to be able to access the Internet at anytime and anywhere through their desktops, notebooks, mobile phones, etc. The traditional approaches to software design seem not sufficient for designing applications in such a large-scaled distributed and sometimes mobile environment. Then what is the next-generation computing paradigm? One of the promising answers is the *mobile agent paradigm*. Mobile agents paradigm have been the focus of much speculation and are hype in recent years. In this chapter, we give an introduction of mobile agents, mobile agent paradigm including strengths, weaknesses and requirements of this paradigm.

1.1 The Mobile Agents

The concept of "Agents" is mainly used in the subject matter of two basic contexts. One is in physical world that the term "agent" or "intelligent agent" is a cognitive attempt to the explanation and simulation of human mental functions. The other is in software world that an agent is a software entity living only in a software world distributed among different types of computers. The software agents perform tasks of humans according to human-like roles/functions designed by software specialists. They are at some extent autonomous and intelligent. The movie "The Matrix" had given us a personification of "agents" both in physical world and software world. They are ideal agents with human-intelligence as they are autonomous, mobile, reactive and able to learn.

In this thesis, the "mobile agent" is referred to as mobile software agent. A mobile agent is a software agent that can migrate from host to host, under own control, in a heterogeneous network in order to perform tasks on behalf of human users using the resources of these hosts. Agents are differentiated from programs by the properties associated with the agent. [FG96]. Although a concise definition for a software agent has yet to be reached, we can summarize some fundamental properties of an agent [Tod98, FG96] as follows:

- Autonomous -- Capable of existing independently; has control of it's own actions
- Adaptive/learning – Changes its behavior based on its previous experience.

- Mobile – able to transport itself from one machine to another
- Persistent -- A continuously running process; temporal
- Goal oriented -- Not merely responds to their environment, takes the initiative to achieve its goal; proactive
- Communicative/collaborative -- Interacts with other agents (and / or users) using an *agent communication language*
- Flexible – Action are not scripted
- Reactive -- Senses changes in its environment and responds in a timely manner

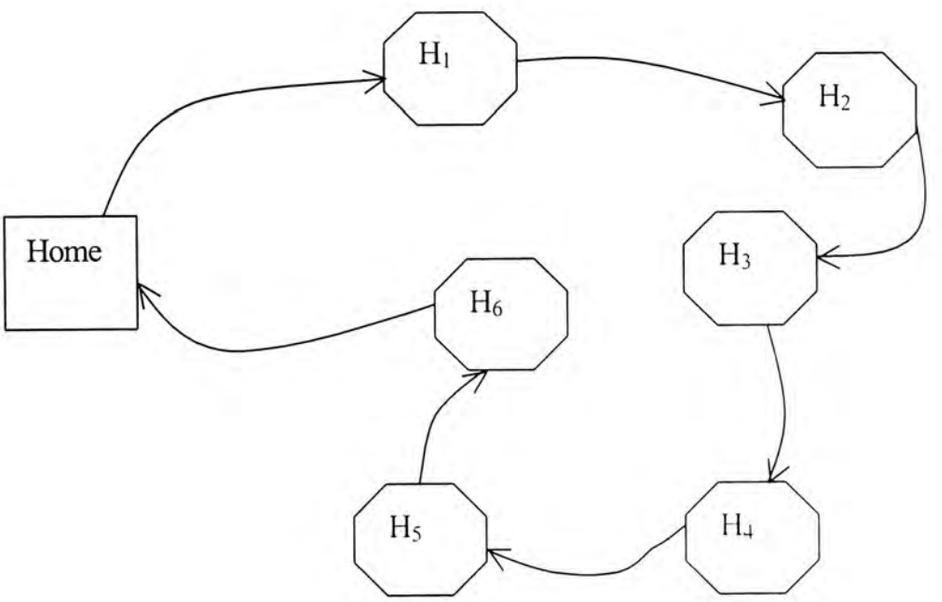

Figure 1.1 Working path of mobile agent

Clearly, a mobile agent is an agent having a subset of above properties with "mobile" as the necessary property. Figure 1.1 shows a typical working path of mobile agent.

1.2 The Mobile Agent Paradigm

Mobile agent paradigm is a kind of mobile code paradigms. Different from the traditional client-server paradigm, mobile code paradigms introduce code mobility into systems that is more suitable in designing large-scale distributed applications. We can identify three kinds of mobile code paradigms: Remote Evaluation (REV), Code on Demand (COD) and Mobile Agent (MA). We give a brief description and comparison between the traditional client-server paradigm and the three mobile code paradigms below.

Client-Server

In client-server paradigm, servers are both the resource and code owners, which may be physically distant to their clients. The client-server communication is initiated by a client request to server for the execution of a service. In response, the server performs services and delivers the results back to the client.

Remote Evaluation

In REV paradigm, one party has code but don't have resources while the other has

the resources but without code. It is the former party who takes initiative to send the code to the receiver who is the resources owner. The receiver executes the code received with the available resource and returns the result back to the sender.

Code On Demand

In COD paradigm, in contrary to the REV paradigm, it is the resource owner who requests the code owner to send code to it. The code owner, after receiving the request sends requested code in response.

Mobile Agent

In MA paradigm, the communication between parties no longer focuses only on the code transfer, but also on the transfer of partial computation results or state as well as some other resources that are also required for the computation. In other words, MA paradigm involves transfer of a whole computational component. Different from other mobile code paradigms, which are two-hop, mobile agent paradigm are multi-hop. That means mobile agent should be able to move from one host to another in order to finish its task without interaction with agent owner, only the final result is transferred back to agent owner.

1.2.1 Initiatives

The initiative to use mobile agent paradigm in distributed computing environment

has been discussed intensively in literatures [HCK95, LO99]. Compared to the traditional client-server paradigm, mobile agent paradigm has several advantages.

a. Reduction of Network Traffic and Latency

Consider the case when the code (the mobile agent) is small, the resources are far away, and a lot of interactions between code and resources are needed for finishing the whole computation. It induces much network traffic and latency for communication. If instead the mobile agent is moved to resources, the interactions become local and real time.

b. Asynchronous Computation

For a mobile user, using rather resource limited computing device, asynchronous computation is quite beneficial. By sending out the mobile agent, the user is free to work on other tasks without affected by the workload of agent' s task.

c. Autonomy

Once the mobile agent starts their journey, they are autonomous to decide where to go and how to react to different situations. This adds "intelligence" into the paradigm and largely reduces the intervention of human users.

d. Customization

In traditional client-server paradigm, functions and services are defined and

established on the server. Clients are confined to use these functions or services and without the ability to customize them. However, in mobile agent paradigm, users can customize or extend mobile agent' s functionality to meet their own needs. Users may even create new services and implants them onto server that previously had to be installed by the service provider.

1.2.2 Applications

Although there is no killer application for mobile agent, there are many applications can be benefited from mobile agent paradigm. Several applications are clearly benefited from the mobile agent paradigm.

Electronic commerce is one of the most interested and concerned areas in mobile agent application. Electronic commerce can utilize advantages of mobile agent paradigm in many facets. For example, purchasing require mobile agent to visit many different services providers and scan large amount of data. Commercial transaction needs real-time access to remote resources such as stock quotes.

Mobile agent paradigm also benefits largely on mobile computing in which mobile devices users often has low bandwidth and intermittent connection to network.

Other applications for mobile agent may include distributed network management [BPW98], workflow management [SWME00], runtime change of software [OMT98], making use of agent autonomy and mobility.

1.3 The Mobile Agent System

A Mobile Agent System (MAS) is a framework that implements the mobile agent paradigm. An mobile agent system consists of the mobile agents, the agent host or agent platform that provides execution environment to mobile agents and some other parties that coordinate and provide services to the whole system such as directory services, certification authority, etc. Thus, in order to host mobile agents, the involved distributed applications must provide a basic sets of services and characteristics as follows:

Transportability

A mobile agent has to be able to suspend its execution in a host, move itself to another node and start its execution from where it stopped. The transportation of the agent from on host to the other must be helped by external entities such as message services, middleware or e-mail servers. The system can either support weak mobility or strong mobility. [FPV98]

Autonomy

To cope with agent autonomy characteristic, the agent has to use resources and mechanisms that allow to making decision on where to migrate.

Security

Executing mobile agents in an open network with different administrative domain can cause serious security problems. An mobile agent system has to provide security mechanisms to protect both mobile agent and agent hosts.

Fault-Tolerance

The MAS must provide resources to the agent programmers for detecting any hardware and software errors. Procedures to overcome error may be to use alternative resource, waiting the resource to become active again [Sch97], etc.

Performance

The MAS must justify performance of the whole system when compared to other paradigms such as the client-server paradigm.

1.4 Security in Mobile Agent System

Security requirement of mobile agent system is the most important for mobile agent paradigm to be deployed into practical applications especially electronic commerce. Issues and security requirements in mobile agent systems have been reviewed and discussed in [FGS96b, GBH98]. Basically, the security issue in mobile agent systems include two large areas:

- Securing executing environment from malicious mobile agents

- Securing mobile agents from malicious environments.

The first area is the traditional security issue. Most traditional protection techniques, such as authentication, access monitoring, audit logging, etc. can be used to provide analogous protection mechanisms for the executing host.

On the other hand, a hostile environment can attack the mobile agent by running the agent code incorrectly, refusing to transfer the agent, tampering with agent code and data or listening into inter-agent communication. Security requirements [JK99, FGS96b] of mobile agents include the following:

Privacy and Integrity: The state and data carried by the mobile agent can have sensitive information such as credit card number, private keys, etc. Thus, both the agent code and agent data should be kept secret.

Anonymity: The identities of the mobile agent should be kept anonymous to public. For a masked mobile agent, not only the identity of the mobile agent should be hidden, but also the hosts that it has visited.

Accountability: If in case one or several of the hosts violate some of the security requirements, we must have mechanisms to account for such violations.

Availability: There should have system mechanisms to ensure that mobile agents are allocated fairly and faithfully with host's resources.

To protect mobile agent from hostile environment is quite challenging. Although research has been put on the issue, there are still no satisfactory solutions yet. Some researchers even think it is impossible to prevent such kind of attacks because mobile agent is under total control by its executing environment. This thesis will pay effort

on the security of mobile agents in hostile environment by analyzing possible attacks on mobile agent and propose schemes to protect them.

1.5 Thesis Organization

This thesis is organized as follows.

In chapter 2, the background knowledge on cryptography and secure function evaluation are reviewed. In Chapter 3, a survey on existing protection schemes on mobile agent is given. Different protection approaches are briefly explained and analyzed. The strength and weaknesses of each protection scheme is presented.

In chapter 4, taxonomy of attacks on mobile agents is proposed. The taxonomy is general and complete and will be useful in our discussion on protection scheme of this thesis. The content of this chapter will be published in the paper "A taxonomy for attacks on mobile agent" [MW01].

In chapter 5, a protection scheme on reactive mobile agents is proposed. The reactive mobile agent has the property that it has the ability to interactively communicate with the host in many rounds, and that implies host will also be able to get some plaintext result from the mobile agent. We achieve this by dividing the intermediate output of mobile agent into two: one is for agent state updating and the other is the output for host. An approach to prevent black box testing is also proposed.

A conclusion of the thesis is given in last chapter.

Chapter 2

Background and Foundations

Many of the protection schemes involved in the protection of mobile agent are achieved by cryptographic techniques. In this chapter, we explain some of the cryptographic primitives [Sch94] that are related and the foundation knowledge mentioned in this thesis.

2.1 Encryption/Decryption

In an encryption process, an original intelligent message (M), or *plaintext*, is converted into a random string (C), or *ciphertext*, by an algorithm and an encryption key. Decryption is an inverse process of encryption. It transforms cipertext to plaintext with a decryption key. The encryption and decryption process are represented by:

$$C = Enc_{k1}(M)$$

$$M = Dec_{k2}(C)$$

In symmetric encryption scheme, the encryption key and decryption key are the same, that is k1 = k2. We call it *symmetric key* or *secret key*.

In asymmetric encryption scheme, the encryption key is different from decryption key but they form a pair. Only the decryption key can decrypt the ciphertext that is encrypted by a corresponding encryption key. It is similar to lock and key. We call the encryption key the *private key* and decryption key the *public key*.

2.2 One-way Hash Function

A one-way function is a function that is relatively easy to compute but significant hard to reverse. That is given x, it is easy to compute $f(x)$, but is much harder to get back x from $f(x)$.

A hash function is a function that takes an input string of variable size and converts it to a fixed-size output string.

A one-way hash function is a hash function that is one-way.

2.3 Message Authentication Code (MAC)

Message authentication code (MAC) is a key-dependent one-way hash function. MACs have same property as the one-way hash function, but they also need a key. Therefore, only the person who knows the identical key can verify the hash.

One simple MAC is to concatenate M and K first, and then computes the

one-way hash of the concatenation. So if Alice wants to send the MAC for the message M to Bob, Bob must have the key K in order to verify the MAC.

2.4 Homomorphic Encryption Scheme

A *homomorphic encryption* is a public-key encryption function E such that given $E(x)$ and $E(y)$, we can easily compute $E(x+y)$ and $E(xy)$.

We call an encryption function E *additively homomorphic* if there is an efficient algorithm PLUS to compute $E(x+y)$ from $E(x)$ and $E(y)$.

We call an encryption function E *mixed multiplicatively homomorphic* if there is an efficient algorithm to MIXED_MULT to compute $E(xy)$ from $E(x)$ and $E(y)$.

2.5 One-Round Oblivious Transfer

Oblivious Transfer protocol fulfils the following scenario:

There are two parties A and B. A has two messages M1 and M2. B is allowed to get one of the messages, but is not allowed to get the other one. A should not know which message B gets.

One-round oblivious transfer protocol fulfils the above scenario in one round trip.

2.6 Polynomial-time Algorithms

The computational complexity of an algorithm is measured by two variables, the

time complexity T and space complexity S. Both T and S are commonly expressed as functions of n, with a "big *O*" notation, where n is the size of the input.

An algorithm is *constant* if its complexity is independent of n.

An algorithm is *linear* if the complexity grows linearly with n.

An algorithm is *polynomial* if its complexity is $O(n^t)$, for a constant t.

Algorithms that have a polynomial time complexity class are called *polynomial time algorithms.*

2.7 Circuit

As defined in [Rog91], a circuit C is a computing device specialized for computing a function from a fixed number of bits to a fixed number of bits. It is a (finite) labeled directed acyclic graph. Each node is labeled by a symmetric Boolean operator drawn from some fixed set of Boolean operators, such as AND, OR, XOR, and their negations. Input nodes are labeled $x_1,\ldots,x_i$ and output nodes are labeled $y_1,\ldots,y_o$.

Circuits provide convenient encoding for finite functions. In a natural way, a circuit C on i inputs and o outputs computes a function f with i-bit input and o-bit output.

The size of a circuit is the number of gates in C plus the number of wires in C. The depth of a circuit is the length of a longest path from an input node to an output node.

Chapter 3

Survey of Protection Schemes on Mobile Agents

3.1 Introduction

Different from the protection of hosts from malicious agents, which is a direct evolution of traditional mechanisms employed by trusted hosts, protection of mobile agents from malicious environments is more radically depart from traditional lines. This is because traditional protection mechanisms were not devised to address threats stemming from attacks on the application by the execution environment. It is exactly the situation faced by an agent executing at an agent platform that is not completely trusted to. The main problem is stemmed from the losing of initiative on agent when agent has moved to other hosts and has to be totally accessible by the agent platforms in order to be executed.

With the realization of attacks that mobile agents may suffer from hostile environment, a considerable body of work has been put on the protection of mobile agents. These approaches can be broadly divided into two main categories [Vig98]: **Detection** and **Prevention**

In this chapter, we will give a survey on the protection approaches already proposed. We are not intended to make an exhaustive list of detailed schemes but to summarize main approaches that were used.

3.2 Detection Approaches

This class of protection approaches aims to detect any unauthorized modification on agent code, state, or execution flow. This detects agent tempering after the attack, tracing the identity of the illegitimate attacker and proving its behavior.

3.2.1 Execution Traces

In [Vig98], an agent host is required to create a trace of an agent' s execution. This trace includes the lines of codes that were executed by the mobile agent and the external inputs that were read by the mobile agent. Before the mobile agent moves to the next agent host, both the trace and the mobile agent' s current state will be hashed and signed by the current agent host. This signed "commitment" will be attached to the mobile agent and sent to next agent host. Every host is needed to store a copy of the trace it creates and the commitment from previous host in a limited amount of

time for later use in case there is any dispute.

After the mobile agent returns home, if the agent owner suspects the execution result, a complete agent execution trace can be recovered. The agent owner can ask the first host for its trace and then use this trace to simulate the agent execution. The correctness of the execution of the first host can be verified by the commitment stored in second agent host. (The identity of second agent host can be identified from the mobile agent's intermediate state). The whole process will be repeated until the last agent host. Any discrepancy in the intermediate will disclose the cheating agent host.

This approach can detect all possible manipulation of agent's code, state or executing flow. However, the detection of manipulation needs a human suspicion and thus not automatic. Also, this approach can only detect discrepancies after the mobile agent has returned home.

The drawback of this approach is that it places heavy burden on agent hosts to create and store the trace, consuming large resources.

3.2.2 Partial Result Encapsulation

In [Yee97], the basic idea of forward integrity was introduced. According to Yee's definition, forward integrity means if a mobile agent visits a sequence of hosts $H = h_1, h_2, \ldots, h_n$ and the first malicious server is h_c , then none of partial results generated at hosts h_i, where $i < c$, can be forged. In Yee's approach, an offer by a

service provider is protected with a partial result authentication code (PRAC), which consists of computing a message authentication code (MAC) [MvOV97]. This technique requires the agent and its generator to maintain or incrementally generate a list of secret keys. The key used in computing the message authentication code at service provider i is k_i and k_{i+1} is related to k_i by a one-way function f such that $k_{i+1} = f(k_i)$. Once a key is used, it will be destroyed before the mobile agent moves to next agent host, thus only the agent owner can reproduce the whole sequence of keys and verify all offers as well as the completeness of the sequence with the knowledge of k_1. A trivial problem to this approach is that agent host i can calculate the keys of k_j for all $j > i$. If later the mobile agent revisits the agent host, offers after host i can be changed without detection. This problem can be solved by the scheme of Karjoth and his associates [KAG97].

The approach is to construct a chain of encapsulated results that binds each result entry to all previous entries and to the identity of the subsequent host to be visited. Each host digitally signs its entry and uses a hash function to link results and identities within an entry. It provides both confidentiality and integrity of the mobile agent's data.

This scheme can detect manipulation automatically for those parts that are protected but the detection can only be made after the agent goes back to home. Also, it protects agent data but not code. Thus, this approach can be used in the application where agent data is much more important than agent code. One typical example is the data-collecting agent.

3.2.3 State Appraisal

The goal of State Appraisal [FGS96a] is to ensure agent has not been subverted by alteration of its state information. This is achieved by the state appraisal functions that checks whether the agent states meets some important invariants. The state appraisal functions are part of agent code. When an agent arrives at a host, the host will decide what privileges to grant the agent depends on the state appraisal functions. If an agent whose state violates the invariants, no privilege will be given. A restricted set of privileges will be given if the state fails to meet some of the conditional factors.

However, it is not clear how well this approach can detect the modifications as the state space of an agent could be quite large, some subtle attacks to agent state may not be detected.

3.3 Prevention Approaches

This class of protection approaches aims at preventing possible attacks by making it impossible or at least difficult to attack an agent' s code or state in a meaningful way. This approach can be either hardware based or software based.

3.3.1 Sliding Encryption

A simple solution to provide the confidentiality of the data collected by mobile

agents is to encrypt the data. However, information gathered by a mobile agent is often small when compared to the encryption keys involved and the resulting cipertext. Young and Yung [YY97] proposed a technique known as Sliding Encryption that allows small amounts of data to be encrypted and yield efficient sized results. The method is aimed at conserving space rather than time. By using sliding encryption, mobile agents encrypt the information collected at each platform visited using public key, and decrypt it using private key when they return to home. However, this solution only provides confidentiality, measures to provide integrity should also be applied before encryption.

3.3.2 Tamper-resistant Hardware

Approaches that rely on tamper-resistant hardware have been proposed by Yee in [Yee97] and by Wilhelm et al. in [WSB98, WSB99].

In [Yee97], the basic idea is to encapsulate the whole agent execution environments inside the tamper-resistant hardware, which is sealed and can withstand potent physical attacks.

A more limited approach [Fun99] is to assume only less powerful hardware can be installed in the system such as smart card. Therefore, not the whole mobile agent but the part that are security important will be protected by the tamper-resistant hardware. This makes a trade-off between the security and the cost. However, the security of this situation will become complex because the malicious agent host can

always control the communication between the protected and unprotected part of the mobile agent unless all the operations of the unprotected part can be verified by the protected part.

Hardware protection approaches can protect agent from most of attacks, but with high cost and is un-scalable.

3.3.3 Multi-agent Cooperation

Roth in [Rot98, Rot99] proposed a scheme with two co-operating agents. It is assumed that there are two independent subgroups of agent hosts, which will not collaborate with each other to attack a mobile agent. The two co-operating agents will migrate to different subgroups each time. These two mobile agents can then use secret sharing [Sha79], remote authorization, and remote storage of commitments to protect the applications. For example, mobile agent A can send the commitment from its host to another mobile agent B. B then can verify whether this commitment satisfies the agent owner's requirement or not. If it is satisfied, B returns its shares of a sufficient amount of electronic-cash to A. However, the drawback of this approach is the cost of setting up the authenticated channel.

The scheme can be generalized to more than two cooperating agents. In [NC99], a scheme called "intention shrinking" is used to shrink the spectrum of an agent by splitting the agent into many collaborating agents and letting them residing in different hosts. By doing so, we can reduce the amount of information disclosed to

any particular host and thus increase largely the difficulty to make meaningful attack to mobile agent

3.3.4 Code Obfuscation

Hohl in [Hoh98a] proposed a protection scheme that protects mobile agents by making agents' code difficult to understand. This includes a transformation from original code to obfuscated one. Hohl assumes the existence of a certain minimal time interval during which it is impossible for an attacker to understand the agent's code and make meaningful manipulation on it. After the specified time interval, all the sensitive data inside the mobile agent should be invalid.

This is a comparatively practical solution with much less complexity than other approaches such as encrypted function evaluation approach. The main problem of this scheme is how to specify a reasonable agent protection interval. Also, there is no theoretical foundation on evaluating the level of security of this approach.

3.3.5 Encrypted Function Evaluation

Sander and Tschudin [ST97, ST98] introduced a rather promising approach that provides hardware-like protection on mobile agent by using software only. This contradicts some researcher's belief that mobile agent cannot be totally protected by software solution.

It has long been believed that programs can only be executed in its plaintext

form. However, Sander and Tschudin questioned that how about if we can find an encryption scheme such that the cipertext is also executable. They proposed that the encryption scheme exists if a segment of program code can be modeled as polynomials and rational functions.

Consider the situation: Suppose now Alice has a function f and Bob has an input x, the function has to be evaluated in Bob's environment. Only Alice should learn the output of the function and Bob should not learn anything of the function.

They should follow the computing protocol as shown in Figure 3.1:

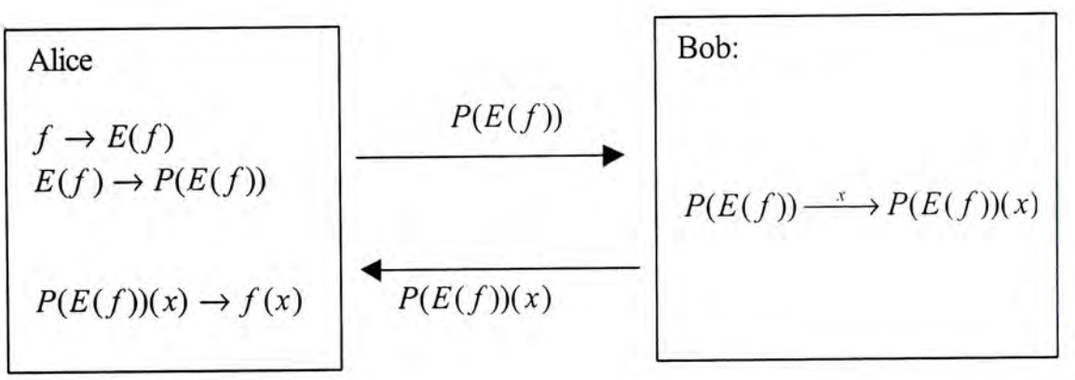

Figure 3.1 Non-interactive Computing with Encrypted Functions

In this protocol, Alice first encrypts the function f to get $E(f)$. Here the encryption scheme is additively and mixed multiplicatively homomorphic. This transformation E consists of the encryption of the coefficients of the polynomial. Thus, the skeleton of the polynomial cannot be hidden. Later, a non-interactive solution for secure evaluation of log-depth circuits has been presented in [SYY99]. The evaluation is done in a gate-by-gate basis using a new privacy homomorphism for processing NOT an OR gates in a sure way. The restriction on the depth of the

circuit comes from the increase of the output size by a constant factor when computing an OR gate.

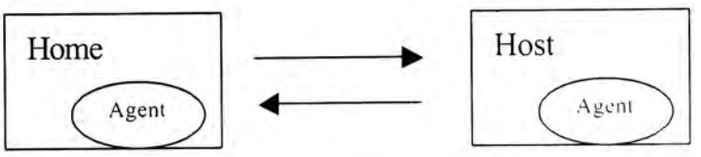

Figure 3.2 Two-hop agent Model

Another similar scheme of encrypted function evaluation is the one proposed by Loureio and Molva in [LM99]. This scheme based on intractability assumption of coding theory and supports those functions that can be represented by matrix. The overhead on the communication and on the remote host computational complexity is linear. The construction is quite cumbersome.

All these approaches only support the scenario when the mobile agent goes to one host for computation and then back to home immediately. (Figure 3.2)

Recently, a more powerful technique called one-round secure computation [CCKM00] that combines the technique suggested in [Yao86] and one-round oblivious transfer [BM89] has been presented. This work extended non-interactive secure computations to polynomial-size circuits. Besides, this approach supports multi-hop computations so that a mobile agent may visit several hosts before back to home. Each visited host may use outputs computed on previously visited hosts without learning anything about the previously used inputs and obtained outputs. The

major drawback of the secure function evaluation is its computation complexity.

3.3.6 Intention Spreading and Shrinking

In [Ng00], a new notion of agent entropy is introduced to measure the spectrum of intention of an agent. This approach aims at spreading intention among collaborating agents so as to prevent any single malicious host from harming the autonomous agents. By intention spreading, such as adding noisy code to the agent, a host becomes more difficult to determine the agent actual intention. Similarly, by intention shrinking, such as splitting the agent into many pieces, the host can only get to know small portion of intentions of the agent. Compared to the code obfuscation technique that only re-configures the mobile agent without changing the global functioning, this technique changes the intention (or entropy) of the mobile agent also. By intention spreading and shrinking, even if the program code is in its plaintext form, the host still cannot get the exact and full information about the program intentions. In other words, instead of code obfuscation, this technique provides obfuscation of intentions of mobile agent.

Although this scheme may introduce redundancy into the system, it can be an alternative or add-on to other schemes such as code obfuscation and encrypted function evaluation to further protect the privacy of mobile agent. This approach can overcome the inadequacy in code obfuscation or even encryption protection of the agent code in case the encryption is cracked.

3.3.7 Black Box Test Prevention

In [HR98], a protocol to prevent black box test of mobile agent is proposed. The main idea is to rely on a third trusted party to register every input and detect any re-input later, so a globally uniquely input identifier is needed. This identifier consists of the following:

AgentID: Identity of the mobile agent

HostID: Identity of the host as well as the number of visits at this host

InputID: A unique number identifying each input of an agent

Note, to include number of visits at the host in HostID allows a mobile agent to visit same host more than once. It is a required feature when a mobile agent needs to go back to the host that gives the best offer among others.

As shown in Figure 3.3, before the mobile agent accepts inputs from host and starts to compute, the mobile agent has to put an input record (registry) on the trusted party. The mobile agent first sends a request containing the unique input identifier {AgentID, HostID, InputID}, an input data hash and an expiring time (Exp_time). The expiring time acts as a record keeping time, after the expiring time, the record will be removed from the trusted party. That means the mobile agent is no longer need to be protected from black box testing because any such attacks cannot harm the mobile agent any more.

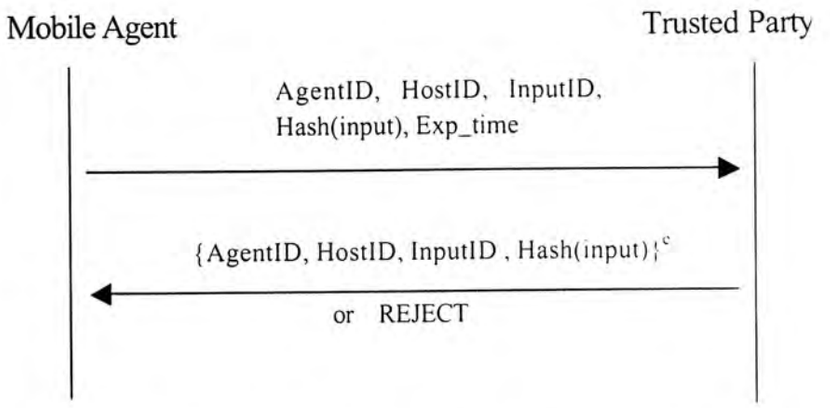

Figure 3.3 Input Recording Protocol

In response to the mobile agent's request, the trusted party either return an "accept" or a "reject" message. If accept, the trusted party will sign the received message with its private key e and return to the mobile agent. Otherwise, the trusted party returns a special message "REJECT". The "accept" message will only be returned when there is no other record having exactly same identifier and data hash in the trusted party's database.

After receiving the reply from trusted party, the mobile agent will check whether host accepts the record by decrypting the "accept" message with trusted party's public key d. Also, the integrity of the message can be checked at the same time. If the decrypted identifier and the data hash are correct, the mobile agent will proceed to execute, otherwise, execution will stop.

This protocol is established under the assumption that mobile agent can be black-boxed in an efficient way. The cost of input registry can only be covered when data transportation is large.

3.4 Chapter Summary

In this chapter, a survey of state-of-the-art protection approaches has been given followed by a thorough analysis of these protection approaches and their relationship and potential application area. Although much effort has been put on the issue of mobile agent protection, the existing solutions are still immature; many of them only give a partial protection or can only be applied to a particular situation.

Chapter 4

Taxonomy of Attacks

4.1 Introduction

As introduced in previous chapters, mobile agent paradigm raises new security issues, especially the problem of malicious executing environment. Not only agent-executing hosts will have risk on inviting viruses, worms, Trojan horses or other kind of attacks that will damage their system, but also the mobile agents will have risk from hostile executing environment. Basically, a malicious host can attack mobile agents in many different ways:

- It can spy sensitive information stored in the agent, such as user's private key, credit card number, or data that have been collected from other hosts during the agent's travel.
- It can modify the data and code of an agent, such as changing the lowest price or the decision logic in the scenario of a price-comparing agent.
- It can even simply alter the agent code and data or even totally destroy the

whole agent.

There are still many more! Then how can these attacks be done? How can they be classified? What is the seriousness of each attack? Can they all be prevented? We should first answer these questions so that we can have a clear direction to provide satisfactory protection to mobile agents. Although attacks from hostile environment to mobile agents have been studied in some context [Hoh98b, GBH98], none of them gives a formal classification of attacks. Taxonomy of attacks is, however, useful for research developments.

In this chapter, we develop attack taxonomy for mobile agent under hostile environments. This new developed taxonomy will be used both in the evaluation of existing protection schemes and the development of new protection schemes.

4.2 What is attack?

When related to mobile agent protection, the term "attack" was rarely defined explicitly. However, before we can develop taxonomy for attacks on mobile agents, we should first define the term "attack". In [Hoh00], a definition of attack was given by using the concept of a reference host (i.e. an abstract host that acts as expected given the same input and resource). Here, we define the attack in a more intuitive way:

An attack is a violation of expectations of the agent programmer or owner caused by one or more than one intentional attacker(s).

Here the word "intentional" explicitly excludes cases when the violation is caused by unintentional exceptions such as errors, misinterpretation of specifications or technical faults.

4.3 How can attacks be done?

It is easily to understand that agents are vulnerable to attacks from its execution environment. However, exactly how these attacks can be done? Hohl in [Hoh98b] introduced a model of attacks that can be the basis for understanding attacks.

We here give a simple explanation. Just imagine that mobile agent will be loaded into an abstract machine when arriving the host. The machine contains the program code, memory cells, program counter, stack and stack pointer. However, the stack pointer is actually in the host memory, the host can always load an attacker program into another abstract machine in parallel when the agent is initialized in the host memory. The attacker has the power to access external environment such as with system information, code library and environmental variables. So actually, the attacker program can fetch and store the content in the memory and stack, or even the program counter and stack pointer in agent's abstract machine.

Therefore, the attacker program can actually copy, change and delete any agent's program statements. Or even worse, attacker program may monitor every execution statement of the agent program and thus can make the agent program to execute in the way it wants.

4.4 Taxonomy of Attacks

Knowing how attacks can be done, we now investigate how can the attacks be classified. As stated in the definition that any difference in the real execution of mobile agents to the expectation of the agent owner will become an attack. Therefore, attacks also include those that are random without any particular aim such as just delete or change something without knowing what is actually being deleted or changed. In extreme cases, the whole agent will be deleted or destroyed. We will discuss attacks according to following two categories:

- Purposeful Attack
- Frivolous Attack

4.4.1 Purposeful Attack

Purposeful attacks are those attacks that are carefully planned and designed so that attackers can take advantage from the attacks. In the mean time, the attackers are conscious on what they are doing, why they do it, and what the possible consequences are by doing so.

We will discuss the purposeful attacks according to two dimensions: the nature of attacks and the number of attackers.

Nature of Attack

There are two kinds of attacks that are rather different in nature. They are read

attacks and non-read attacks.

Read Attack.

It is the threat of attackers reading or copying the private data, code or even the flow control of a mobile agent. It leaves no trace that could be detected. That is the case when the attacker program just copy from memory or stack of the agent's machine and do nothing else. The actual effect may occur a long time after the visit of the mobile agent.

We now analyze this kind of attack by following three levels of read attack:

a. Data reading

The most simple and direct way is to read out the private data such as secret keys or electronic cash carried by mobile agents. This simple knowledge of data results in loss of privacy and money.

b. Code reading

Knowing the code leads to knowledge about the execution strategy of the mobile agent. Consider the scenario that a mobile agent is traveling around the network for finding out cheap but high quality CD, malicious CD shop may use the knowledge about the choosing criteria to win the competition or record down the buying habit of the customer.

c. Execution reading

Combining with the knowledge of code and data, attackers can deduce information about the state of the agent. In the scenario of finding a best offer, the

attacker can recognize whether an offer is better or worse than the best offer so far by simply watching the execution steps,

Non-Read Attack

Unlike the read attack that leaves no traces immediately after the execution, non-read attacks involve the attacker actively making some actions and thus leaving traces. This includes many different kinds of attacks. Such as modification of code, data or even execution flows. Other attacks such as denial-of-service, black box testing, incorrect execution, incorrect input, are all fall in this category.

Number of Attackers

According to the number of attackers, we classify the attacks into two kinds:

Solo Attack

This kind of attack is those attacks that are done by a single attacker. A solo attack can encompass read attack, non-read attack as well as a combination of these attacks as long as it does not include any cooperation with other attackers.

Collaborative Attack

Different from solo attack, collaborative attack includes several attackers to collaborate in order to achieve an attacking goal. This kind of attack may also include reading attack and non-reading attack. For example, several hosts may collaborate to

track a mobile agent's path in order to get information about it or its sender. Another example is that when one of the attackers modified the mobile agent's data or code, the other attacker ignores it. This kind of attack is sometimes strong and may defeat some of the protection approaches.

4.4.2 Frivolous Attack

In contrary to the purposeful attacks, frivolous attacks are those attacks that are not carefully planned and designed so that attackers may or may not take advantage from the attack. Attacker may change or delete some bits of the mobile agent without knowing what actually they are changed or deleted. Also, they do not know what is the effect for such an action. In extreme cases, they may totally destroy the agent or refused to execute the agent.

We can classify this type of attacks into random attack and total attack. Random attack means the random change or delete of parts of the agent or just don't execute some part of the agent or don't give enough resources for agent computation. One may ask why this kind of attack will exist as it seems that the attacker have no motive to do so. Well, actually this kind of attacks does exist. Sometimes, because of curiosity or just want to try out something to see the result. Besides, It may be the indication of further purposeful attacks. One "good" facet for attackers to do so is that even when their action has been disclosed later, they can always excuse that it is because of some random error, or technical faults.

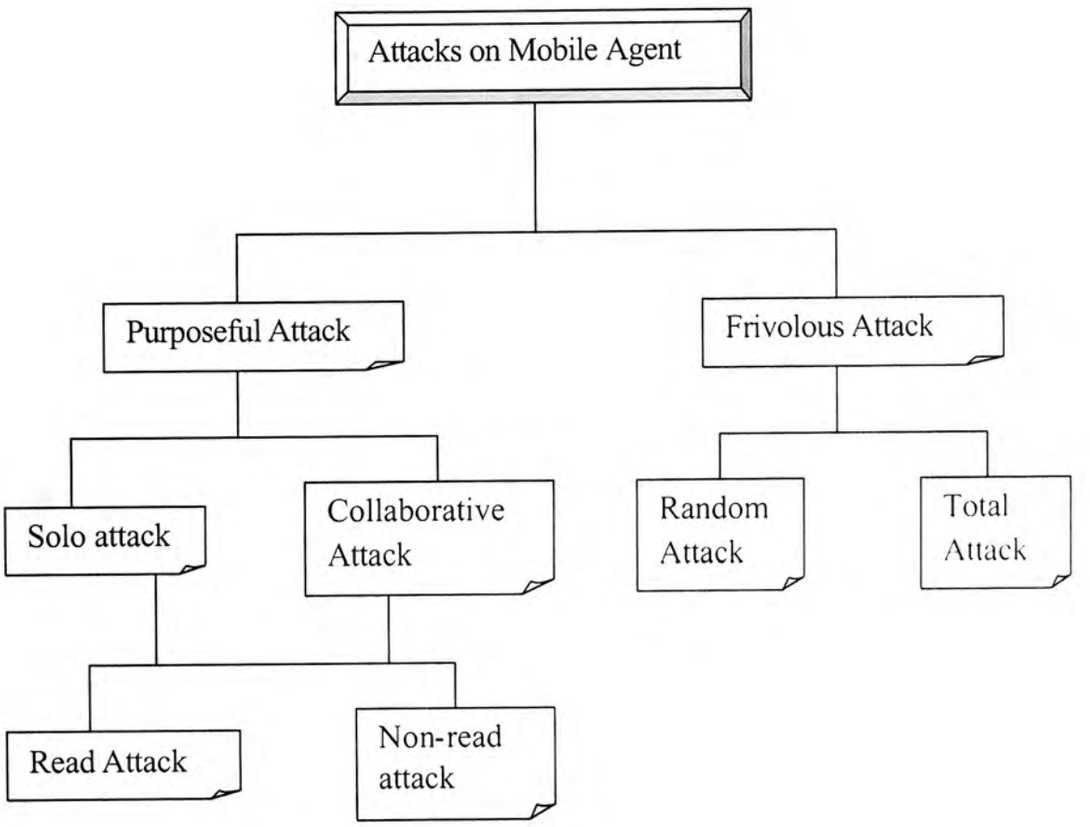

Figure 4.1 Taxonomy of attacks on Mobile Agent from Hostile Environment

Totally destroy the agent or refuse to execute the agent can be classified as total attack. In this case, mobile agents are either disappeared or cannot finish their tasks because of the refused execution.

4.4.3 The Full Taxonomy

Figure 4.1 shows the full taxonomy. It is important to understand that in the real life, an attack may not just fall in one of the classes; several classified attacks may combine together as one in order to achieve a particular attacking goal.

4.5 Using the Taxonomy

In this section, we will show how the proposed taxonomy could be matched to the existing protection schemes to evaluate the effectiveness of the protection schemes as well as to bring out insights in making some new and innovative protection schemes.

4.5.1 Match to Existing Protection Schemes

Many protection schemes have been proposed to combat attacks on mobile agents. However, most of these efforts only fight one or several classes of attacks but not comprehensive.

We now use the just developed taxonomy of attacks to give some evaluation to

existing protection schemes. In following paragraphs, we will briefly explain each class of protection schemes and illustrate what kind of attacks they can fight by using examples. It should be noted that not all of the existing protection schemes are listed here, but the taxonomy can still be applied to those schemes that are not list here

Detection Approach

Clearly, these approaches are effective for non-read attacks that leave traces for detection but not for read attacks. In particular the protocol suggested in [Hoh99] relies on the next host to check the status of the mobile agent coming from the previous host. This is also vulnerable under the collaborative attack when two consecutive hosts collaborate.

Hardware Based Prevention Approach

In this approach, secure hardware devices are used to protect agents. By using special trusted hardware as presented in [Wil97, Wil99, Fun99], even the host who is running the mobile agent cannot access the agent code and data. This prevents most of the solo attack but may not be effective for collaborative attacks. For example, one malicious host routes the agent to another collaborating malicious host without detection by the owner. In this case, other approaches must be added to fight these kinds of attacks.

Software Based Prevention Approach

This class involves many different protection schemes each gives different levels of protection. Some of the protection schemes try to encrypt the agent data such that only the corresponding receiver can decrypt by using public key encryption scheme. A typical example is [YY97] in which a mobile agent uses sliding encryption technique to encrypt acquired data by using a public key so that potentially malicious hosts cannot steal the data. However, these approaches can only fight against data read attack but not others like code read attacks and modification attacks. They are suited to protect the mobile agent with data privacy much important than code privacy.

Obscuring a mobile agent's code to make it hard for reverse engineering is another approach. The one proposed in [Hoh98a] is a quite promising method called the time-limited black box security. This protocol can protect the mobile agent from most of the attacks but not long time read attack. Since it only protects within a limited time period, it is not suitable to be used in some applications in which mobile agents have to carry precious data that will be valid in a long time.

In [ST98], Tomas Sander and Christian F. Tschudin proposed a scheme by using encrypted function such that program could be executed without decryption. It means the host that mobile agents visit runs the agent in ciphertext form instead of plaintext form, so it is difficult to give purposeful attack to mobile agents. However, this approach only supports special kind of functions, namely polynomials and rational functions and does not prevent denial of service, replay and other forms of attacks.

4.5.2 Insight to Potential Protection Schemes

From the discussion in previous section, we have some insight for developing new protection schemes.

Firstly, not all applications require the same set of countermeasures. Instead, countermeasures are applied commensurate with the anticipated threat profile and intended security objectives for the application.

Secondly, few protection schemes are suggested for frivolous attacks that are difficult to be totally protected from. Fortunately, they are quite similar to random errors, so we can use some fault-tolerant approaches such as error detection and correction to increase the survivability of agents.

Thirdly, to totally protect mobile agent from read attacks is the most challenging part. The mobile agent needs to be executed by some hosts but at the same time does not want the host to be able to know what they have executed. Hardware dependent approach can be one direction but it results in high cost and is not scalable. Software based approach to fight read attack still have much room for further research.

Fourthly, besides solo attack, collaborative attacks are also worth noting, as this kind of attacks can be varied largely and specific to certain kind of protection schemes, so in developing new schemes, we should always bare in mind collaborative attacks.

4.6 Chapter Summary

In this chapter, we have presented taxonomy of attacks on mobile agent in hostile environment. The nature of various attacks is discussed. This taxonomy is a base for our further studies on protection of mobile agents and can be applied to match with the existing protection approaches to see how they are effective to various attacks.

Chapter 5

Protection For Reactive Mobile Agents

5.1 Introduction

As introduced in Chapter 3, software protection approaches that can protect mobile agents from most of attacks are code obfuscation and encrypted function evaluation. The former protects mobile agents from a spying host for a limited time while the latter protects mobile agents with no time limitation. Encrypted function protection scheme was firstly proposed in [ST98]. Sander and Tschudin recognized that, at least in principle, it is possible to completely protect mobile agents by software only solution applying cryptographical tools such as homomorphic encryption approaches. However, only functions representable as polynomials can be computed securely in this manner. Sander et al. [SYY99] later extended this to all functions that are

computable by circuits of logarithmic depth. Recently, Cachin et al. [CCKM00] further generalized this to arbitrary functions as long as they can be represented by polynomial-size circuit. This is achieved by combining Yao's "encrypted circuit" method [Yao86] for secure computation with a one-round oblivious transfer protocol. [BM89].

However, all these approaches protect non-reactive mobile agents only. That is mobile agent cannot have interaction to the host. In this chapter, we will propose the reactive mobile agent computation model and a scheme to protect reactive mobile agent. In our protection scheme, host can also get computation results from mobile agent and mobile agent can have several rounds of interaction to the host while at the same time without leaking any information of agent state to host.

This chapter is organized as follows: Section 5.2 explains what we mean the reactive mobile agent and the computation model of reactive mobile agent. Section 5.3 reviews the abstract version of Yao's encrypted circuit construction and the technique to cascade the circuits without revealing every circuit's output that is presented in [CCKM00]. These two serve as the theoretical basis for our protection scheme. Section 5.4 gives the details of our protection scheme. Section 5.5 analyzes the security of the proposed protection schemes by looking at how well the scheme can combat various kinds of attacks. A special attack to the reactive mobile agent – the black box attack is addressed. Section 5.6 improves our basic protocol to combat black box testing as well as frivolous attacks. Section 5.7 discusses some further considerations on our protocol.

5.2 The Model

We first give a macroscopic view on the non-reactive and reactive mobile agents. Then, a microscopic computation model will be given.

5.2.1 The Non-reactive and Reactive Mobile Agent Model

In the following, the non-reactive model and reactive model are referred to mobile agent computation in one host only but not mobile agent' s whole lifetime. Therefore, we here do not set any limitation on the number of host that the mobile agent can visit, so both active and non-active model in this section could be two-hop or multi-hop.

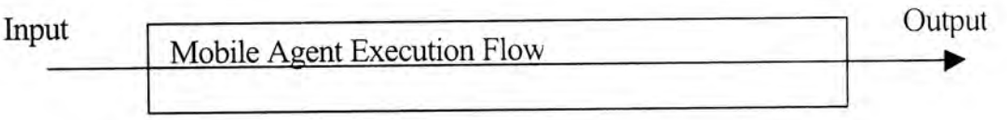

Figure 5.1 The Non-reactive Model

We say a mobile agent execution is non-reactive if the mobile agent accepts input and calculates output only once in one host. That is there is no interaction between the mobile agent and the host during computation process as shown in Figure 5.1. This is the model generally used in encrypted function protection approaches. It is because once the function is encrypted, it only accepts inputs and

calculates the corresponding outputs accordingly.

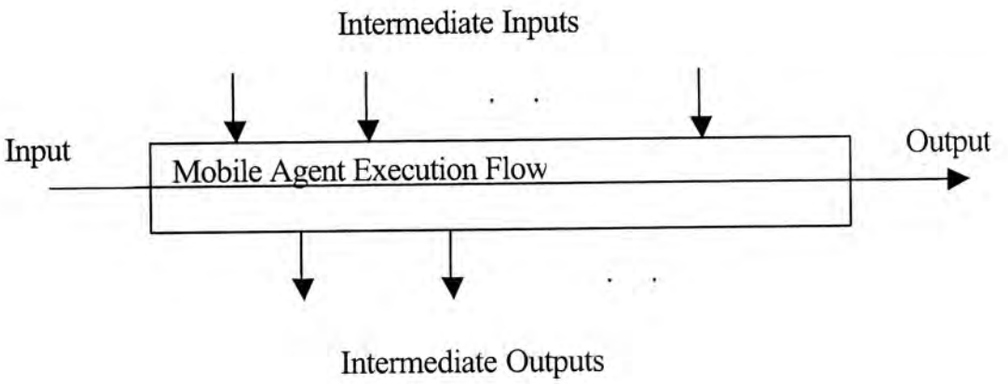

Figure 5.2 The Reactive Model

The difference of the reactive model and non-reactive model is that the mobile agent will have several inputs and outputs on a single agent host for reactive model. As shown in Figure 5.2, there are intermediate inputs and outputs during execution process. The next intermediate input depends on the previous intermediate output. Thus, mobile agents can have interactive communication with hosts. This improves the original model in the sense that it supports more complex and reactive mobile agents.

Consider the scenario when a mobile agent roaming in network to book birthday cake on behalf of its owner. The mobile agent has the ability to bargain the price of the cake according to some strategies that are specified by the agent owner. The mobile agent should also be able to book the "best offer" cake. That means it should

be able to sign a booking order immediately after it has found the best offer. At the same time, the mobile agent will request the commitment from the host as well as challenge the host for authentication. All these requirements need a reactive mobile agent.

Intuitively, we can regard the whole flow of computation of mobile agent in one particular host as composed of several consecutive computation sections. Let the computation between first input and first output be section 1, and second input to second output be section 2 and so on. We want each section of computation be encrypted and computed securely. Before introducing the protocol, let's study the computation flow inside the mobile agent more carefully.

5.2.2 The Computation Flow

Imagine the internal logic of a reactive mobile agent: to be reactive, it must be able to distinguish between different input of host and then react to that input of a host. That means the mobile agent must be able to decide the next action according to host input in a real time. Therefore, each section of mobile agent computation can be viewed as a mapping from many input values to a small set of output values (decisions). If there are several output values, then there will be several parallel sections each corresponds to one output. One example of internal flow of mobile agent is shown in figure 5.3.

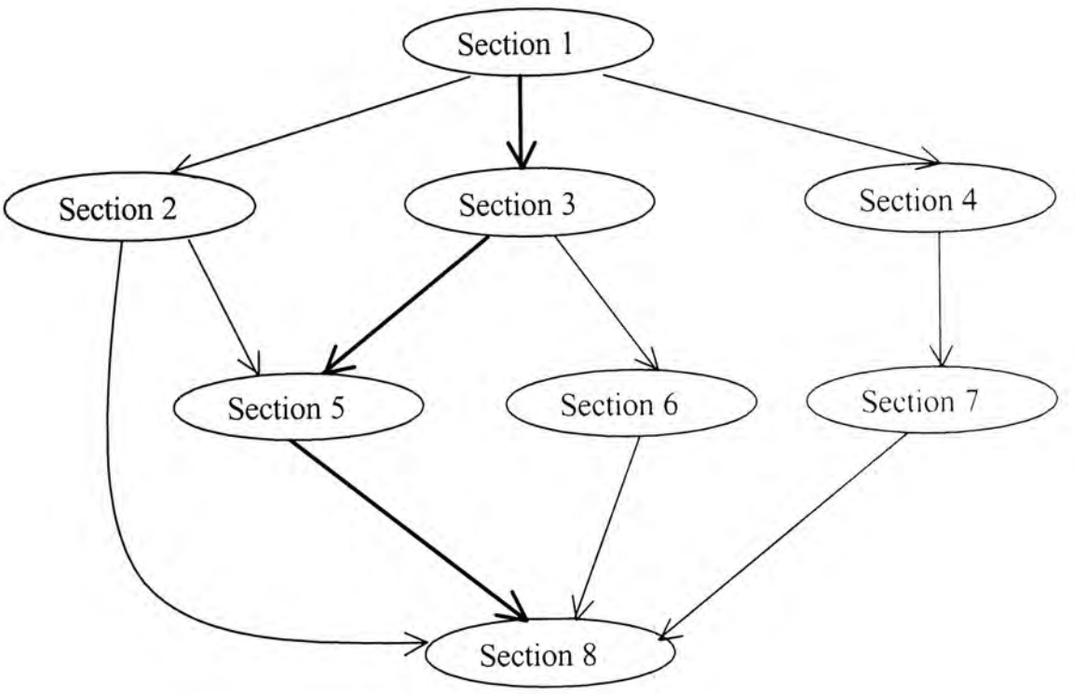

Figure 5.3 Computation flow of Reactive Mobile Agent

Each node in the graph represents a computation section. The first node is the initial section. Every section may have several predecessors and successors. The former means different sequence of inputs from the host may result in same final action. The latter means different actions will respond to different input of host. This can be easily explained in the scenario of a bargaining mobile agent. Different bargaining path can result in final booking. The dark line shown in the figure represents one of the possible computation paths in a host. Note that all the sections will eventually converge to one section in the graph. It is because the mobile agent

will eventually need to conclude the computation in the host and calculate the next host to migrate to.

5.2.3 An Example

We now use the bargaining mobile agent aimed at booking a birthday cake as an example to illustrate what we have introduced in previous sections.

Figure 5.4 shows a simple bargaining agent with only 4 sections. The agent' s bargaining strategy is as follows: The maximum price that will be accepted is $150. If host input price is lower than $150, then the agent will book the cake immediately; If host input price is between $150 to $200, then the agent will try to bargain the price using a bargaining base $100. That is to take a average between the host input value with $100 and then output to host to see whether it accepts the value; If the host input is greater than $200, then the agent will not try to bargain and immediately request to go to next host. If the host accepts the bargaining value or input another price that is lower than $150 in second round, the agent will also accept, otherwise go to next host. Anytime when the agent accepts an offer, it will go home immediately.

In Figure 5.4, some detailed variable settings such as minor state changes from section to section for flow controls are omitted for simplicity. This example is presented for an intuitive understanding on reactive mobile agent computation flow and will be used in our following discussion on the protection of such mobile agents.

```
Section 1:
Maximun_accepted_price = 150;  Bargaining_base = 100;
Accept(input_price);
If (input_price > 200) then OUTPUT REJECT and GO section 4
If (input_price > 150 and input_price < 200) then OUTPUT (input_price
   +bargaining base)/2 and GO to section 2.
If (input_price <150) then OUTPUT ACCEPT and Go to section 3.
```

```
Section 2:
Accept(input_price);
If (input_price <= 150) then OUTPUT ACCEPT and GO to section3.
Otherwise then OUTPUT REJECT and GO to section 4
```

```
Section 3:
Accept(booking order);
Signing the booking order and set agent current_state = finish;
OUTPUT signed booking order and go to section 4
```

```
Section 4:
If (current_state = finish), then go HOME
Otherwise, calculate the next host and OUTPUT host ip.
```

Figure 5.4 A simple bargaining agent

5.3 Tools

In this section, we review the basic tool and technique we will use in our protection scheme. One is Yao's "encrypted circuit construction" or "garbled circuit construction" and the other is a technique to privately cascade the circuit without revealing the intermediate information.

5.3.1 Encrypted Circuit Construction

As we will use the encrypted circuit construction of Yao [Yao86] as our basics, we first give an abstract description of Yao's construction.

Here only those properties that are necessary to our protocol are shown. Details and in-depth analysis please refer to [Rog91].

Assume there is a binary function g with two inputs x and y and output z, ie. $z = g(x, y)$. Now, let $(x_1,...,x_{n_x}),(y_1,...,y_{n_y})$, *and* $(z_1,...,z_{n_z})$ denote the binary representation of x, y and z respectively, and let C denote a polynomial-size binary circuit computing function g. Yao's construction includes the following two procedures:

1) An algorithm CONSTRUCT

This is a probabilistic algorithm that takes the circuit C as input and outputs the tuple **(C, X, Y, Z)** where **C** is a representation (or a encrypted version) of circuit $C(.,.)$,

and

$$\mathbf{X} = (X_{1,0}, X_{1,1}), \ldots, (X_{n_x,0}, X_{n_x,1})$$
$$\mathbf{Y} = (Y_{1,0}, Y_{1,1}), \ldots, (Y_{n_y,0}, Y_{n_y,1})$$
$$\mathbf{Z} = (Z_{1,0}, Z_{1,1}), \ldots, (Z_{n_z,0}, Z_{n_z,1})$$

In order to compute $C(x, y)$, one needs a k-bit key for each input bit x_i and y_i. The key $X_{i,b}$ corresponds to the key used for the input x_i = b and the key $Y_{i,b}$ corresponds to the key used for the input y_i = b. The pairs $(Z_{i,0}, Z_{i,1})$ represent the output bits. Thus, **X**, **Y**, **Z** denote lists of "key pairs" corresponding to x, y, z respectively. We call these "keys" the tags.

Note that if we hardwire one input, say y, into the circuit C, then the resulting circuit is single-input and is equivalent to the original one by fixing value of y.

2) An algorithm EVALUATE

In order to compute the circuit $C(x, y)$ in its encrypted form **C**, we should choose the input tags first.

If the input bit x_i = 0, we will choose the tag $X_{i,0}$, otherwise choose $X_{i,1}$. Similarly,

If the input bit y_i = 0, we will choose the tag $Y_{i,0}$, otherwise choose $Y_{i,1}$.

After choosing the tags for every bit of input x and y, we can input these tags to the encrypted circuit **C** to get the output.

Now let

$$X_i' = X_{i,x_i} \quad for \quad i = 1,\ldots,n_x \quad and \quad \mathbf{X}' = X_{1,x_1} X_{2,x_2} \ldots X_{n_x,x_{n_x}}$$
$$Y_i' = Y_{i,y_i} \quad for \quad i = 1,\ldots,n_y \quad and \quad \mathbf{Y}' = Y_{1,y_1} Y_{2,y_2} \ldots Y_{n_y,y_{n_y}}$$

The algorithm EVALUATE take the encrypted circuit and two sets of selected tags (which are representation of x and y respectively) as input and outputs either a special symbol REJECT or a representation of z.

$$\mathbf{Z}' = \text{EVALUATE}(\mathbf{C}, \mathbf{X}', \mathbf{Y}')$$

Therefore, from the output tags, we can easily get the output value z. If we get $Z_i' = Z_{i,0}$, we set output bit $z_i = 0$, if we get $Z_i' = Z_{i,1}$, we set $z_i = 1$.

To explain in simple words, the encrypted circuit **C** ensures that given two sets of tags representing its inputs, the set of tags representing the resulting output bits can be readily calculated. The existence of CONSTRUCT, EVALUATE is based on the existence of pseudo-random functions [GGM86] and efficient implementations of pseudo-random functions can be based on the Decisional Diffie-Hellman assumption. [NR97].

5.3.2 Circuit Cascading

In [CCKM00], a scheme is presented to secretly cascade the circuit without revealing the intermediately results of circuit computation. This technique is briefed below:

Suppose there are two encrypted circuits, **C1** and **C2**, the output of **C1** will be the input of **C2**. We want to calculate the two encrypted circuits in sequence; however,

we don't want to reveal the value of the output of **C1** before inputting to **C2**, we can use the following technique:

The circuit constructor should encrypt each input tag of circuit **C2** by using corresponding output tag of **C1** as the symmetric key. After encryption, there should be n_x pairs of encrypted tags. We then randomize the position of the tags inside a pair, and called the randomized list **U**. The party who computes **C2** only gets the encrypted randomized list **U** for circuit **C2**. In order to calculate **C2**, those output tags calculated from **C1** can decrypt half of the encrypted tags in **U** and thus recover the set of input tags representing current computation state for **C2**. However, the party is not aware of what has exactly been input to the circuit **C2** because **U** is randomized in position beforehand, so it doesn't know whether the decrypted tags representing 0 or 1.

Here note that the symmetric encryption scheme has the property that given a key K, one can efficiently check if a ciphertext represents an encryption under key K, which is in the same way as the encryptions for single gates in Yao's construction.

5.4 Proposed Protection Scheme

From the model we have introduced in section 5.2, we know that for a mobile agent to be reactive, we have to support the property that the mobile agent can also output something to the host so that the host can re-input to mobile agent according to its output. We say this can be achieved by dividing the output of the circuit into two

parts, $\mathbf{Z_x}$ and $\mathbf{Z_y}$. One part represents agent states that should be kept secret throughout the whole path of the computation. The other part is the output to host. (Figure 5.5). The part of agent states can be kept secret by using the circuit cascading technique. For the remaining part of circuit output, we can send corresponding tags pairs to hosts so they can recover plaintext output. The detailed protocol is presented in the following sections. We first illustrate the protocol for reactive mobile agent protection in two-hop scenario and then extend the two-hop protocol to multi-hop protocol.

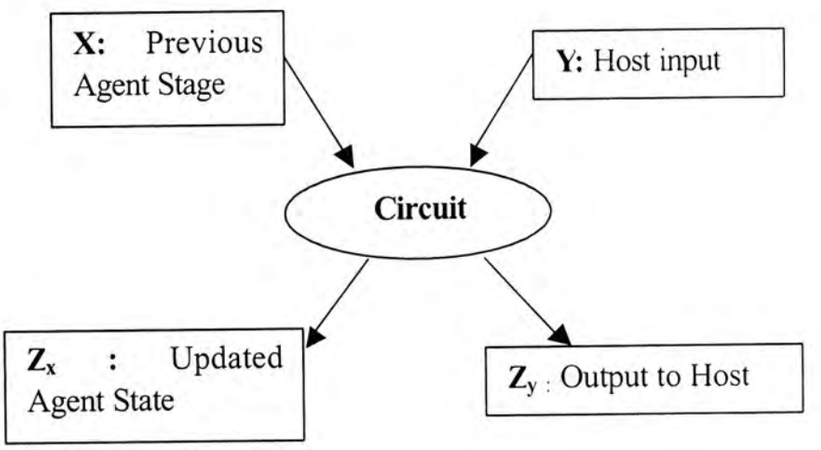

Figure 5.5 Relation of tags and encrypted circuit

5.4.1 Two-hop Protocol

In two-hop scenario, mobile agent will go only to one host for computation and then back to agent owner. We present our protocol in four stages below:

1. Agent Preparation

The preparation work of agent includes encrypting all the sections of computation (ecliptics in Figure 5.3) as well as the computation links (arrows in Figure 5.3). Here, we use Yao' s encrypted circuit construction as described in section 5.3.1 for encrypting the sections. To encrypt the computation links (or agent states), we will use the circuit cascading technique as described in section 5.3.2. That is use predecessor' s output tags as the symmetric keys to encrypt the input tags of the successor. If a section has more than one successor, we will encrypt each successor's input tags by the predecessor' s output tags. Details are as follows:

Encrypting computation sections:

Agent owner O run the algorithm CONSTRUCT(C^i) for i = 1, ... , s where s is the number of sections in the agent and gets the corresponding tuples:

$$(\mathbf{C}^i, \mathbf{X}^i, \mathbf{Y}^i, \mathbf{Z}^i) \quad \text{for i} = 1, \ldots, \text{s}$$

The output tags $\mathbf{Z}^i$ are divided into two parts, one part is the tags that are used to represent updated agent state by section i, and the other part is the tags that are used to represent the output to the host. Let $\mathbf{Z}_x^i$ represents the set of tags that updates agent state by section i and $\mathbf{Z}_y^i$ represents the set of tags that gives output to host by section i. Then $\mathbf{Z}^i$ is a cascade of $\mathbf{Z}_x^i$ and $\mathbf{Z}_y^i$.

Note that the length of list $\mathbf{X}$ is the same for every i. That means the agent state should be represented by fixed number of bits.

Encrypting computation links:

For each circuit $\mathbf{C}^i$, agent owner finds out the circuit's successor and do the following:

For each successor $\mathbf{C}^j$, use tags of $\mathbf{Z}_x^i$ as the symmetric encryption keys to encrypt input tags of $\mathbf{X}^j$ of $\mathbf{C}^j$. That is:

For each k = 1, … , n_x, encrypts tag $X_{k,b}^j$ under $Z_{k,b}^i$ (for b = 0, 1), and set

$$U_{k,b\oplus c_k}^j = Enc_{Z_{k,b}^i}(X_{k,b}^j)$$

where C_k is a random bit chosen for each k..

Now, what we have got? To explain in simple words, the corresponding tags pair at same bit position of predecessor' s output tags encrypts the successor's input tags pair for each bit representation. However, inside a tags pair, we don' t know the relative position of encrypted tags **U** to the original tags **X** because we have introduced a random bit C_k to randomly permutate the position of **U** inside a pair.

2. Agent Transfer

After the whole agent (including its computation flow) is encrypted. Agent owner will selectively send the agent to host such that only those parts that are necessary for computation are sent. Figure 5.6 shows a list of stuff that the agent owner should send to the agent host.

1. $\mathbf{C}^1, \mathbf{C}^2, ..., \mathbf{C}^s$: Encrypted circuits for all the computation sections.
2. $X'^1_1, X'^1_2, ..., X'^1_{x_n}$: The selected tags for agent initial state.
3. $(U^i_{1,0}, U^i_{1,1}), (U^i_{2,0}, U^i_{2,1}), ..., (U^i_{n_x,0}, U^i_{n_x,1})$ (i = 2, ... , s) : Encrypted and randomized input tags for agent state for the encrypted circuits. (Note that a circuit having more than one predecessor should have more than one set of tags.)
4. $(Y^i_{1,0}, Y^i_{1,1}), (Y^i_{2,0}, Y^i_{2,1}), ..., (Y^i_{n_y,0}, Y^i_{n_y,1})$ (i = 1, 2, ... , s) :

   Host input tags for all of the circuits. (Note value of n_y may be different for different circuit.)
5. $(Z^i_{n_x+1,0}, Z^i_{n_x+1,1}), (Z^i_{n_x+2,0}, Z^i_{n_x+2,1}), ..., (Z^i_{n_x+n_z,0}, Z^i_{n_x+n_z,1})$

   (i = 1, 2, ... , s) : Output tags representing the output to host for all the encrypted circuits. (Note : value of n_z may be different for different circuit)

Figure 5.6 Agent Transfer List

Note that the agent owner sends all the prepared stuff to H_i except those output tags representing the part of updated agent state. Thus, host will not get the information of agent state.

3 Agent Computation

After receiving the message from agent owner, the host should compute the agent according to following procedure:

a. Selects the input tags of $\mathbf{C}^1$ according to its first input to get $\mathbf{Y}'^1$

b. Runs EVALUATE $\left(\mathbf{C}^1, \mathbf{X}'^1, \mathbf{Y}'^1\right)$, and gets $\mathbf{Z}'^1$.

c. Recovers output of $\mathbf{C}^1$ from $\mathbf{Z}'_{y'}{}^1$ and $\mathbf{Z}_{y'}{}^1$

d. Determines the next encrypted circuit to be computed and the next input to that circuit according to the output from current circuit.

e. Assume the next circuit to be computed is $\mathbf{C}^k$, tag $\mathbf{Z}'^1_i$ can decrypt one of the encrypted input tag $U^k_{i,0}$ or $U^k_{i,1}$ for i = 1,...,n_x. After decryption, a set of input tags $\mathbf{X}'^k$ representing agent current state will be got.

f. Runs EVALUATE $\left(\mathbf{C}^k, \mathbf{X}'^k, \mathbf{Y}'^k\right)$ and repeat the steps c – e until the last circuit $\mathbf{C}^s$ is reached.

g. Send the computation result back to agent owner.

4. Final State Retrieval

The agent owner O will receive a set of tags $\mathbf{Z}'^s_x$ representing agent's final state from the host. The agent owner can then retrieve agent's final state by comparing with $\mathbf{Z}_x{}^s$.

5.4.2 Multi-hop Protocol

It is not difficult to extend two-hop protocol to multi-hop protocol. Assume that the mobile agent travels in a closed loop, from an agent owner O to host H_1 and then from host H_j to host H_{j+1} for j = 1,...,m-1 and then from H_m back to O. To be autonomous, the agent path is not fixed and can be decided by the mobile agent itself. We also do not confine the number of hosts that a mobile agent can visit.

To let the next host can continue the computation using the result of previous host, we use same technique same as circuit cascading, but now the predecessor is $\mathbf{C}^s$ and the successor is $\mathbf{C}^1$. It is the next host that runs $\mathbf{C}^1$ after $\mathbf{C}^s$ computed in previous host. Therefore, in agent transfer stage, agent owner O should also transfer encrypted randomized list **U** of section $\mathbf{C}^1$.

Data that should be sent from H_j to H_{j+1}, for j = 1, ... , m-1, is nearly the same as that O sent to H_1 except item 2 in the list will be replaced by output tags of $\mathbf{C}^s$, which represents the state of agent just before agent transfer.

Finally, H_m should send last agent state (last output tags) to O. From what received, O can recover the final computation result of the agent.

5.5 Security Analysis

In this section, we give an analysis of security on our protection scheme by looking at how well it can combat various kinds of attacks.

5.5.1 Security under Purposeful Attacks

Our scheme is secure under read attack. The host has no means to get information of agent code and data as it is encrypted or garbled. The host also cannot get any information of agent state by read as because agent state is represented by set of randomized tags from beginning to the end.

Our scheme is secure under most of non-read attacks such as modification attacks. By encrypted circuit construction, host cannot make any meaningful modification to the mobile agent. If hosts try to execute the mobile agent incorrectly, or input incorrect data, they can only get a REJECT from the circuit output. For example, after section 1 is computed, the host is told to compute section 2. If the host does not follow and instead, it tries to execute section 3 that is also a successor of section 1 so it can decrypt the input tags for section 3, the host cannot be successful because the circuit is not allowed to do so by outputting a REJECT symbol. This kind of violation can easily be checked by circuit itself just looking at the agent state information.

However, the above scheme is vulnerable to black box attack. Black box testing to mobile agent is a special kind of attack directing towards those mobile agents with black box property. In our reactive mobile agent model, outputs are also given to agent hosts; this special security issue must be considered. As now the agent host can try mobile agents as many times as it wants, until it gets enough information to guess the logic of the mobile agent.

Consider our simple bargaining agent example, the host can always run the section 2 many times with different values of input. If the agent rejects its offer, then next time lowers the price, otherwise raises its price, until the agent just reject the offer. Then the host can easily test out the maximum acceptable price set by the agent owner. That is $150! Certainly, agent owner doesn't want this to happen. We will improve our protocol in section 5.6.

5.5.2 Security under Frivolous Attacks

In our basic protection scheme, we haven' t taken frivolous attacks into account by implicitly assume host will not undergo frivolous attack. However, if one of the sections is randomly erased, then the whole agent cannot be executed as expected. If the host destroys the whole agent, the agent cannot be recovered automatically.

Therefore, if we also consider frivolous attack, as discussed in chapter 4, we can use many different kinds of fault-tolerance approaches. We will propose possible solution in next section also.

5.6 Improvements

In this section, we suggest possible solutions to combat the black box testing as well as frivolous attacks for our reactive mobile agent protection scheme. We first give an intuitive explanation on combating the attacks and then give the actual implementation possibility.

5.6.1 Basic Idea

As black box testing is to test the agent by running it with different input values, a direct and intuitive method is to ensure hosts can run or only able to run the mobile agent once. There is no way to prevent such attacks by agents themselves. It is because hosts can always duplicate many copies of agent, and run them in parallel. Therefore, we have to introduce a third party to supervise the operation and disallow a host to run the mobile agent more than once by inputting different values. Every time when host wants to input data into the agent, the host has to request the trusted party for approval. Computation of agent is allowed only when the agent receives the approval from the trusted party.

One of the solutions is to apply a general protocol as proposed in [HR98]. To integrate the protocol into our reactive mobile agent model, each section that needs the protection from black box testing has to add the functionality of input recording. Host should input to each section not only its input data, but also the signed approval from the trusted party. However, this is rather impractical, as the complexity of circuit construction will become prohibitively large for such a complex function. Instead, we will propose another scheme that is an easy add on to our basic protection scheme.

5.6.2 Input Retrieval Protocol

Instead of using general protocol, we propose a new protocol for preventing black

box attack that is specific to our reactive mobile agent protocol.

As host needs the input tags for calculating the encrypted circuit, we can control the host input by not sending the input tags to host, but instead sending them to a trusted party. Therefore, when the host wants to input to the mobile agent, it has to request the trusted party to give the corresponding keys. The detail could be as follows:

The agent owner O will send all the stuff listed in Figure 5.6 except the item 4 to the first host. The item 4 together with AgentID, expiring time and corresponding SectionIDs are sent to the trusted party.

When the host wants to give a protected input to agent, it should send its input together with its ID, AgentID and sectionID to trusted party. The trusted party will then send the corresponding tags back to the host and keep down the unique input identifier (introduced in section 3.3.7) until time is expired.

However, this approach is still vulnerable to collaborative attack. Imagine that when two hosts collaborate, one host sends an input, say Y, to the trusted party, and gets the corresponding tags. Then the other sends the complement of Y to the host, it gets the complement tags. By combining two, they get all the input tags and may then undergo black box testing attack to mobile agent!

In order to prevent collaborative attack, we further improve our protocol by introducing dummy input tags. That is when we construct the encrypted circuit, we will embed some dummy input bits in the circuit *C*. Position of these bits are kept secret from host but revealed to trusted party. Each time the host requests for input,

the trusted party will send back the input tags together with the dummy tags embedded in. So even if two hosts collaborate, as before, they will be uncertain where are the exact positions of useful input tags. Although if many hosts collaborate in this case may still reveal all the input tags, we can at least tolerate collaborative attacks to certain extent depending on the number of dummy bits added.

To elaborate more, we give a simple example. Assume the number of input bits is n, and we add also n dummy bits to it, then we have 2n pairs of input tags after circuit construction. A host H1 sends n-bit input to trusted party and gets back 2n tags. Another collaborating host H2 sends the complement of the n-bit input, and then it certainly gets at least n tags that are different from H1. However, as the trusted party will randomly assign the tags at dummy bit position, on average, half of another n tags are same to that got by H1 and half are different. The two hosts have to guess the positions of n useful tags by choosing n out of (n + n/2).

One drawback of this scheme is the increase of complexity of the circuit, so we need to make tradeoff between the efficiency and the privacy.

5.6.3 Combating Frivolous Attacks

From Chapter 4, we have learnt that frivolous attacks are generally difficult to totally prevent from. However, we could use some fault-tolerant techniques to at least tolerate frivolous attacks to some extent.

We may add error control codes at the end of each computation section, such as

Reed-Solomon code or check sums. In case some of the bits in the section are randomly changed, the error can be detected and recovered by error control codes. We may also add dummy codes to whole agent, such that the mobile agent can be recovered even if the whole section of computation is deleted.

To tolerate total attack, cooperating agents approaches could be used. That is the agent under computation will send to another agent residing on trusted host with its most recent result. In case the mobile agent is destroyed, the cooperating agent could send a copy of the mobile agent with most updated state to host for continuing computation. For example, in our bargaining agent case, if a host H_j stops agent execution at the middle, the cooperating agent can then send a copy of agent to next host H_{j+1} with agent state the output state of H_{j-1}. Thus, the mobile agent computation can be continued through skipping the malicious host H_j.

5.7 Further Considerations

A drawback of using Yao' s secure circuit construction is that it is only efficient for small functions, its complexity increase significantly for more complex functions. Therefore, this scheme may be restricted to only those small applications. One example is the bargain agent with simple bargaining strategy.

For larger applications, we may use encrypted circuit only for security sensitive part with other program statements in plain code. Or to be more secure, the plain code can be obfuscated first. For example, if we want to integrate the input recording

protocol into circuit of each section, the overhead is too large to be really deployed. But if we use obfuscated code to implement input recording part and let the result directly input to the circuit, the complexity is much lower. All in all, which method should be used depends on what level of security we want.

5.8 Chapter Summary

A protection scheme on reactive mobile agent is presented. Our scheme is inspired by the one-round secure function evaluation presented in [CCKM00]. We explained the internal computation logic of reactive mobile agent by using the example of bargaining agent. We realized the reactiveness of mobile agent by separating the agent output into two parts, one is an update of agent state and the other is the output to host and thus host can also get the plaintext result from mobile agent form which it can decide what is the next input. Agent states are kept secret by using circuit cascading technique.

However, this scheme still suffers from black box testing attack. We then suggest the possible solutions to combat the black box testing. We concluded that a general protection protocol is not suitable in our scheme and proposed a protocol that is new and specific to our protocol. It is comparatively practical and can be applied to applications that need several rounds of interaction between mobile agent and hosts. These applications include bargaining and negotiation

Chapter 6

Conclusions

This thesis is focused on the problem of securing mobile agent from possibly hostile execution environment. This is a quite challenging issue that has given rise to much research interest. However, there is still no satisfactory solution. We have proposed our own protection schemes based on the analysis of possible attacks from hostile environment

To start with, we presented various facets of mobile agent paradigm and compare it with more traditional implementation techniques. We pointed out that the mobile agent paradigm is most probably the next generation computing paradigm for large-scale distributed and mobile systems. It is supported by the beneficial aspects of mobile agent paradigm: reduction of bandwidth and network latency, support of asynchronous computation, autonomy and client customization. We also suggested possible applications that can be benefited from mobile agent paradigm.

However, there are many requirements for implementing a mobile agent system.

The most important one is the security requirement, which is the prerequisite for the mobile agent system to be largely deployed into real life especially to electronic commerce applications. We discussed the security issues in the mobile agent system and paid emphasis on the security of mobile agents which is more difficult to fulfill because we have to keep secrecy and integrity of mobile agents while at the same time give full access to the executing environment in order to execute the mobile agent.

We then surveyed different kinds of protection approaches that have been proposed in the literature and analyzed them with their strengths and weaknesses. Cryptographical background was listed beforehand to facilitate the understanding on the protection schemes. Based on the nature of the protection approaches, we classified them into two categories: detection approaches and prevention approaches.

In chapter 4, we have paid effort on the analysis of various kinds of attacks that hostile environment can impose on the mobile agents. We developed taxonomy of these attacks. This taxonomy can be useful both in evaluating existing protection schemes and in inspiring the development of new protection schemes.

We have proposed a protection scheme on reactive mobile agent in chapter 5. Our approach is one kind of encrypted function evaluation that aims at read attack and modification attack and can tolerate collaborative attack. We use Yao' s encrypted circuit construction as our basis. By cascading the encrypted circuits with a predefined logic, and outputting plaintext results also to host, we can support the mobile agents to be reactive. The output of a circuit is used as a symmetric key to

encrypt the input of next circuit so that two circuits can be securely connected without revealing intermediate state of a mobile agent. However, the black box property of encrypted circuit can be vulnerable to a special kind of non-read attack, that is black box testing. We also proposed a method to integrate black box testing prevention protocol into our reactive mobile agent model. Further protection on frivolous attacks has been considered at the end.

To summarize, although there are a wide variety of protection approaches to mobile agent, they are not compatible with each other due to redundancy in purpose and overlap in functionality. Prevention schemes often implicitly involve detection, so when all other things being equivalent, a prevention approach is preferable to one oriented towards detection as detection approach depends on legal or other social framework to penalize misbehavior. Also in order to effectively apply the available approaches, many of the approaches should be integrate into an agent system, while the others can be applied independently according to particular application.

All in all, the area of mobile agent security is still in its immaturity, but rapidly improving. Mobile agent paradigm offers a new paradigm for application development and it grows with significant potentials.

Appendix

Paper Derived From This Thesis

[MW01] Mo Chun Man, Victor K. Wei, "A Taxonomy for attacks on Mobile Agent". Accepted for publication in EUROCON'2001, July.

Bibliography

[BM89] M. Bellare and S. Micali, "Non-interactive oblivious transfer and applications," in Proc. CRYPTO'89 (G. Brassard, ed.), LNCS 435, pp. 547-557, 1990

[BPW98] A. Bieszczad; B. Pagurek; T. White. Mobile agents for network management. IEEE Communications Surveys, September 1998

[CCKM00] Cachin, Christian; Camenisch, Jan; Kilian, Joe; Müler, Joy: One-round secure computation and secure autonomous mobile agents. In: Ugo Montanari, Jos?P. Rolim, and Emo Welzl (Eds.): Proc. 27th International Colloquium on Automata, Languages and Programming (ICALP), Geneva, volume 1853 of Lecture Notes in Computer Science, pages 512-523. Springer-Verlag, 2000.

[FG96] S. Franklin; A. Graesser. Is it an Agent, or just a Program? A Taxonomy

for Autonomous Agents. In proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages. Springer-Verlag. 1996.
http://www.msci.memphis.edu/~franklin/AgentProg.html

[FGS96a] Farmer, William; Guttman, Joshua; Swarup, Vipin: Security for Mobile Agents: Authentification and State Appraisal. Fourth European Symposium on Research in Computer Security (ESORICS 96), (Pages 118-130).

[FGS96b] Farmer, William; Guttman, Joshua; Swarup, Vipin: Security for mobile agents: Issues and requirements. In Proceedings of the 19th National Information Systems Security Conference, pages 591-597, Baltimore, Md., October 1996

[FPV98] A. Fuggetta, G. P. Picco, and G. Vigna. Understanding Code Mobility. IEEE Trans. on Software Engineering, May 1998.

[Fun99] Fünfrocken, Stefan: Protecting Mobile Web-Commerce Agents with Smartcards. In: Proceedings of the First International Symposium on Agent Systems and Applications / Third International Symposium on Mobile Agents (ASA/MA'99), IEEE Computer Society, pp. 90-102,

1999

[GBH98] Greenberg, Michael S.; Byington, Jennifer C.; Harper, David G., "Mobile Agents and Security", *IEEE Commun. Mag.*, July 1998, vol. 36, no. 7, pp. 76-85, 1998

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali, How to construct random functions, Journal of the ACM 33 (1986), no. 4, 792-807

[HCK95] C.G. Harrison; D.M. Chess; A. Kershenbaumn. Mobile Agents: Are they a good idea?
http://www.research.ibm.com/iagents/paps/mobile_idea.pdf

[Hoh98a] Hohl, Fritz: Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts, in: Giovanni Vigna (Ed.): Mobile Agents and Security. pp 92-113. Springer-Verlag, 1998.

[Hoh98b] F. Hohl, "A Model of Attacks of Malicious Hosts Against Mobile Agents", *Proc. of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations, INRIA, France*, (1998) 105-120

[Hoh99] F. Hohl, "A New Protocol Protecting Mobile Agents From Some Modification Attacks", *Technical Report Nr. 09/99, Faculty of Informatics, University of Stuttgart, Germany, 1999.*

[Hoh00] F. Hohl, "A Framework to Protect Mobile Agents by Using Reference States", *In: Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS 2000)*

[HR98] Hohl, Fritz; Rothermel, Kurt: A Protocol Preventing Blackbox Tests of Mobile Agents. Accepted paper for the 11. Fachtagung "Kommunikation in Verteilten Systemen" (KiVS'99).

[JK99] Jansen, Wayne; Karygiannis, Tom: Mobile Agent Security. NIST Technical Report, National Institute of Standards and Technology, 1999

[KAG97] G. Karjoth, N. Asokan, and C. Gulcu. Protecting the computation results of free-roaming agents. In Rothermel and Popescu-Zeletin [RPZ97], pages 1-14

[KBC00] Kotzanikolaou, P.; Burmester, M.; Chrissikopoulos, V., "Secure Transactions with Mobile Agents in Hostile Environments", *In: Dawson, E.; Clark, A.; Boyd, C. (Eds.): Information Security and Privacy. Proc.*

of the 5th Australasian Conference, ACISP 2000. LNCS Vol. 1841, Springer-Verlag, pp. 289-297, 2000

[LM99] S. Loureiro and R. Molva, "*Privacy for Mobile Code*", Proceedings of the workshop on Distributed Object Security, OOPSLA '99, p. 37—42

[LO99] Danny B. Lange; Mistsuru Oshima. Seven good reasons for mobile agents. Dispatch your agents; Shut off your machine. Communications of the ACM, 42(3):88-89, March 1999

[MvOV97] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. Handbook of applied cryptography. CRC Press, Inc., 1997

[NC99] S. K. Ng and K. W. Cheung, "Intention Spreading: an extensible theme to protect mobile agents from read attack hoisted by malicious hosts," First Asia-Pacific Conference on Intelligent Agent Technology, 1999, Hong Kong

[Ng00] Ng, Sau-Koon: Protecting Mobile Agents against Malicious Hosts. Master Thesis. Division of Information Engineering, The Chinese University of Hong Kong, June 2000.

[NR97] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," in Proc.38th FOCS, 1997

[OMT98] P. Oreizy, N. medvidovic, and R. N. Taylor. Architecture-based runtime software evolution. In Proceedings of the International Conference on Software Engineering 1998 (ICSE' 98), Kyoto Japan, April 1998

[Rog91] P. Rogaway, The round complexity of secure protocols, Ph.D. thesis, Laboratory for Computer Science, MIT, April 1991

[Rot98] V. Roth: Secure Recording of Itineraries through Cooperating Agents, in: Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations, pp 147 - 154, INRIA, France, 1998.

[Rot99] V. Roth: Mutual Protection of Cooperating Agents, in: Jan Vitek; Christian Jensen (Eds.): Secure Internet Programming, LNCS 1603, Springer-Verlag, pp. 275-288, 1999.

[Sch94] Bruce Schheier. "Applied Cryptography", second edition, John Wiley & Sons, Inc.

[Sch97] Schneider, Fred: Towards Fault-tolerant and Secure Agentry. Invited Paper to the 11th International Workshop on Distributed Algorithms, Saarbrücken, Germany, Sept. 1997.Also available as TR94-1568 , Computer Science Department, Cornell University, Ithaca, New York. http://cs-tr.cs.cornell.edu:80/Dienst/Repository/2.0/Body/ncstrl.cornell%2fTR97-1636/postscript

[Sha79] A. Shamir. How to share a secret. Communication of the ACM, 22(11):612-613, November 1979

[ST97] Sander,Tomas; Tschudin,Christian: Towards Mobile Cryptography. Technical Report 97-049, International Computer Science Institute, Berkeley. 1997. http://www.icsi.berkeley.edu/~sander/publications/tr-97-049.ps

[ST98] Sander,T.; Tschudin,C.: Protecting Mobile Agents Against Malicious Hosts. In G. Vigna, editor, Mobile Agents and Security. Springer-Verlag Berlin Heidelberg, 1998. http://www.icsi.berkeley.edu/~sander/publications/MA-protect.ps

[SWME00] R. S. Silva Filho; J. Wainer; E. R. M. Madeira; C. Ellis: CORBA Based Architecture for Large Scale Workflow. Special Issue on Autonomous

Decentralized Systems of the IEICE Transactions on Communications, Tokyo, Japan, Vol. E83-B, No. 5. May 2000, pp.988-998

[SYY99] Tomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for NC^1. In Proceedings of the IEEE FOCS, October 1999

[Tod98] Todd Sundsted. An Introduction to Agents. http://www.javaworld.com/javaworld/jw-06-1998/jw-06-howto.html

[Vig98] Vigna, Giovanni: Cryptographic Traces for Mobile Agents, in: Giovanni Vigna (Ed.): Mobile Agents and Security. pp 137-153. Springer-Verlag, 1998.

[Wil97] Willhelm, U.G., "Cryptographically Protected Objects", *Technical Report, Ecole Polytechnique Federale de Lausanne, Switzerland (1997)*

[Wil99] Wilhelm, Uwe: A Technical Approach to Privacy based on Mobile Agents Protected by Tamper-resistant Hardware. PhD Theses Nr. 1961. Departement D'Informatique, Ecole Polytechnique Federale de Lausanne, 1999.

[WSB98] U. Wilhelm, S. Staamann, and L. Buttyán, On the Problem of Trust in Mobile Agent Systems, IEEE Network and Distributed Systems Security Symposium (NDSS'98), pages 114-124. Internet Society, March 1998.

[WSB99] Wilhelm, Uwe G.; Staamann, Sebastian; Buttyàn, Levente: Introducing Trusted Third Parties to the Mobile Agent Paradigm, in: Jan Vitek; Christian Jensen (Eds.): Secure Internet Programming, LNCS 1603, Springer-Verlag, pp. 469-492, 1999.

[Yao86] A. C. Yao. How to generate and exchange secrets. In IEEE symposium on Foundations of Computer Science 86, pages 162-167, Toronto, 1986

[Yee97] Bennet S. Yee. *A sanctuary for mobile agents.* In Proceedings of the DARPA Workshop on Foundations for Secure Mobile Code, March 1997.

[YY97] Young, A., Yung, M.: *Sliding Encryption: A Cryptographic Tool for Mobile Agents.* (ed) Eli Biham, Proceedings of the 4th International Workshop on Fast Software Encryption, FSE'97, January 1997, LNCS 1267, Springer-Verlag, 1997