Secret Sharing for Mobile Agent Cryptography

A Thesis
Presented to
The Academic Faculty

by

Chung Miao

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

Department of Computer Science
Lakehead University
May 2003

0-612-84956-2

DEDICATION

This thesis is dedicated to my grandmother Wang who passed away September, 2001 at the age of 102. I also want to dedicate this thesis to my grandpa Gu who passed away February, 2002 at the age of 78. I miss you and I will always remember your love.

ACKNOWLEDGEMENTS

First, I would like to thank my thesis supervisor Dr. Ruizhong Wei for his patience, support and advice. Dr. Wei was the originator of the idea of using secret sharing as a tool for mobile agent security. I would also like to thank my external examiner Dr. Xining Li for his willingness to contribute his time to examination of this thesis. Also, I want thank the every at Lakehead University's Computer Science department for giving me the opportunity learn and grow over the last two years.

I wish to express my gratitude to my wonderful and lovely wife Yentle for her patience and constant encouragement over the years and especially during the time I spent writing this thesis. Last but not least, I want to thank my parents, brother, sister and anyone who has shown me support over the last two years.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

A mobile agent is a multi-threaded autonomous program that can be distributed over a heterogeneous network to perform some predefined task for its human creator (user). The independence, flexibility and autonomy offered by the mobile agent paradigm hold many promises for the future of distributed computing [18]. However, for an agent to be autonomous, it must carry its code, states and results from host to host. Since an agent executes on remote potentially hostile environments, hiding information from hosts has proven to be a tough challenge.

This thesis introduces two novel ideas that can be used in the mobile agent paradigm. First, is the use of Shamir's [33] (t, n)-threshold secret sharing scheme for the distribution of the private key of a public/private key pair amongst n agents. Thus giving an agent the ability to use cryptographic primitives for protection of intermediate results obtained from previously visited hosts. An agent can use the public key for encryption while the private key is safely shared between the agent and its siblings. Second, a *"cookie"* will be introduced as an tool for avoiding agent collisions. Analogous to cookies used by web browsers for tracking of visitors to a site, it will be used as a tool for marking previously visited hosts. The goal of marking previously visited hosts is to resolve the problem of *agent collisions.* Agent collisions occurs when a group of agents from the same originator with the same purpose visits a particular host more than once.

ABBREVIATIONS

CA	-	Certificate Authority
CEF	-	Computing with Encrypted Functions
COD	-	Code On Demand
DES	-	Data Encryption Standard
DC	-	Digital Certificate
DS	-	Digital Signature
EC	-	Execution Code
ES	-	Execution State
EU	-	Execution Unit
IP	-	Internet Protocol
MA	-	Mobile Agent(s)
MAC	-	Message Authentication Code
PKC	-	Public Key Cryptography
PKE	-	Public Key Encryption
PKI	-	Public Key Infrastructure
PRAC	-	Partial Result Authentication Code
REV	-	Remote Evaluation
RPC	-	Remote Procedure Call
RSA	-	Rivset, Shamir and Adleman PKE
SE	-	Slide Encryption
SEE	-	Secure Execution Environment
SH	-	Secure Hardware

SSS	-	Secret Sharing Scheme
TTP	-	Trusted Third Party
TH	-	Trusted Host
VM	-	Virtual Machine

CHAPTER I

INTRODUCTION

Few would argue the importance of the internet as a medium for the distribution and retrieval of information. The vast amount of information available is overwhelming. What began as a hand full of hosts in the late 1960's has since become a world wide network of over 160 million hosts [41]. This growth has not been strictly confined to the internet. With the availability of cheap and reliable hardware, smaller *intra-nets* have also experienced similar growth. A computer's connectivity to a network is no longer considered an optional add-on, but an integral part of its functionality. Until just recently, users were confined to well defined static sets of locations to gain network access. However, the recent advancements in wireless and mobile technologies means that today, the same users can literally *bring the network with them.* The effect of this growth is the ever increasing demand that the entire network infrastructure be more flexible and scalable.

Traditional models of network computing rely primarily on well established technology based on the client-server paradigm. In the typical client-server scenario, contact is initiated by the client as some form of request for execution of service. The server which hosts computational resources, data and a static set of services performs the request on behalf of the client then responds with the result. Depending on the types of services requested, multiple rounds of interactions between the client and server might be required to complete the request. This model of network computing has its usefulness and is well suited for most applications. However, because of the ever increasing size and dynamics of the internet, this model is beginning to show its limitations. For instance. a client requests a service from a server which the server has the computational resources and necessary data to fulfil the request but lack the knowledge (e.g. program). To complete the request, the client either has to download the data from the sever to process locally on the client's machine or find another server that can complete the request. In either case, the model lacks flexibility and exerts unnecessary strain on the network infrastructure.

The benefit of the mobile agent paradigm over the above paradigm is its inherent ability to process information at its source. A mobile agent is autonomous; each agent is encapsulated with the knowledge required to perform its function. A host is obligated to provide only the input and computational resources to an agent. This flexibility has the potential of resolving many of the shortcomings of the client-server network computing model in use today. Chess *et al.* [6] investigation into the benefits of using mobile agents concluded that mobile agents offer a better general solution in terms of *prototyping*, *transactions* and *scalability* when compared to client-server model of distributed computing. Its ability to reduce network traffic, operate over intermittent unreliable network connections, and execute over heterogeneous networks and environments makes the paradigm an attractive option for future implementations of e-commerce, information research and mobile device applications [18].

1.1 The Security Problem

Agent code written by one party typically executes on remote host under control of another party. Because neither party has to know or trust the other party, without adequate security that fulfills the requirements of both, participation of either party in this potentially beneficial relationship is unlikely.

The level of security required by an agent depends primarily on the function of the agent. Potential gains by implementing security on an agent must always be weighted against the cost of providing such security. It is arguable that in certain instances an agent requires no security. For example, an agent is created to search for information on the topic, *"mobile agent security"*. If the purpose of the agent is to collect a list of links and abstracts on documents pertaining to the subject. Then the risk of such agent to attacks should be considered relatively low. Providing such agent with cryptographic security might be far too expensive to both the agent owner and remote hosts in terms of hardware resources such as CPU cycles and storage requirements. On the other hand, if an agent is carrying private and sensitive information such as, credit card numbers and private keys on behalf of its owner. Such agent must not only be protected, cryptographic primitive should be given to the agent so that information collected can be protected as well. For example, in the often used scenario where an agent is searching for a plane ticket from one destination to another. The importance of protecting secret information the agent carries on behalf of its creator is obvious. However, in this case, results collected by the agent from visited competing hosts must also be protected. To keep all participating hosts honest, none of the hosts should know what offers any of the other hosts has rendered.

Often, a mobile agent is thought of as an *"autonomous"* entity capable of making decisions, executing code fragments and storing results before migrating to another host. Although justified, after all, autonomy is one of the underlining primitives of a mobile agent. However, to truly grasp the complexity of the many security issues associated with a mobile agent, we must think of them at a lower level of structural components. Mobile agents are nothing more than encapsulated code fragments comprising of execution code, execution states and data segments. The agent migration process although conceptually similar traditional migration from location to location, agent migration is simply the replication of an agent on another host's machine. An agent can only migrate if the environment which *"hosts"* the agent establishes a connection with a target host, copies the agent on to the host before destroying the local copy. It is at this level of thinking the true complexities of securing an agent can be realized. How does an agent hide information from a host if the host has total control over the agent? What stops a host from deleting the agent or its data all together? What if the host refuses to transport the agent? As you can see the problem of a malicious host is quite challenging.

1.2 Thesis Position

Until Sanders and Tschudin [32], using only software approaches for the protection of a mobile agent was generally considered to be impossible. In [32], the proposed solution uses the notion of encrypted functions for the protection of an agent and its functionality (see Chapter 4). However, this method is complex and has limited applications. Therefore, for most applications, the problem of protecting an agent from a malicious host remains mostly unresolved. The position of this thesis is that, although the agent itself cannot be protected, using cryptographic primitives, its results/data can. The term *protection* in this case is the encrypting of results obtained by an agent on previously visited hosts. The use of cryptographic primitives for the protection of data has its limitations, a malicious host will still be able to delete encrypted results on an agent. However, this problem is wide ranging in the mobile agent paradigm as mentioned earlier. The premise behind using cryptographic primitives is *"out of site of mind"*. If a host cannot decipher any meaning from data contained within an agent, the likely hood of tampering by such host can be dramatically reduced. An example is the agent searching for the airline ticket, if a host cannot determine what offers other hosts has rendered, then the hosts will tend to be more honest with its bid. By cryptographically enciphering the agent's results, a host is left with two decisions, to delete the agent or just parts of the agent which the host believes contains bids from competing hosts. Because there's no financial gain by deleting the agent, the host must weight the possibility of being detected and *"black listed"* against any potential benefits of deleting only parts of the agent.

1.3 Thesis Layout

This thesis will show how Shamir's (t, n)-threshold secret sharing scheme [33] can be used to share the private key of a public/private key pair amongst n agents. The public-key can be used by an agent to encrypt results obtained from visited host. By distributing the private key as shares, only agents with a piece of the original share can participate in the successful regeneration of the private key. Thus provide additional security for the protection of the private key. Chapter 2 describes the mobile agent paradigm including its applications and security. In Chapter 3, cryptographic techniques for the prevention and detection of malicious host attacks will be discussed. Chapter 4 will first give a general overview of Shamir's [33] threshold scheme before outlining how it's used in the protocol to hid and distribute shares to agents. Later in Chapter 4, the protocol will be extended to show how an agent on a remote *"trusted host"* can adopt parts of the same protocol to distribute agents of its own. Chapter 5 will demonstrate how the protocol can be used with RSA [28] keys. Chapter 6 investigates the use of a *"cookie"* to track agents for avoiding *agent collision.* Security related issues of using cookies will also be discussed in Chapter 6. Chapter 7 concludes this thesis. (Note, all or parts of Chapters 2, 3, 4, 5, 6 will appear in the 1st annual proceedings of the *Communications Networks & Services Research* (CNSR) conference [23].)

1.4 Security Terms For Mobile Agents

The following security definitions all have relevance to security in the mobile agent paradigm. For every term, a general definition will first be given and whenever possible, the context to which it pertains to the mobile agent paradigm.

Authentication: In security, to *authenticate* is to verify the identity of a person or process. Authentication is also used as a term for describing integrity of information. For instance, whether results obtained by an agent has been tampered with. Agents often need to authenticate a host to avoid *masquerade* types of attacks. The authentication of a host by an agent and *visa-versa* is commonly performed through the *Public-Key Infrastructure* (PKI), consisting of *Digital Signatures* (DS) and *Digital Certificates* (DC).

Public-Key Infrastructure: A system of DC, *Certificate Authorities* (CA), and other registration authorities that verify and authenticate the validity of each party involved in an internet transaction. It is assumed in the rest of this thesis that DC belonging to an agent owner, or a host, are legitimate products of the PKI.

Public-Key Cryptography: *Public-Key Cryptography* (PKC) refers to encryption e and decryption d using a pair of asymmetric keys, public-key k_{pub} and private-key k_{priv}. Each k_{pub} is made public typically through the use of DC and k_{priv} is only known to its owner. Let the m be a message, $e_{k_{pub}}(m)$ is the encryption of m and $d_{k_{priv}}(e_{k_{pub}}(m)) = m$ is the decryption of $e_{k_{pub}}(m)$ using the receiver's k_{priv}. In the case of mobile agents, the receiver is assumed to be the owner of the agent unless otherwise specified.

Digital signatures: DS are produced using k_{priv} which is only know it is owner. Once singed, the signature is non-reputable. Anyone can verify a signature using the associated k_{pub}. Let *sig* be the signing function, $sig_{k_{priv}}(m)$ is a signature of m using k_{priv}. Let *ver* be the verification function of *sig*, if $ver_{k_{pub}}(m, Sig_{k_{priv}}(m)) = true$, then $sig_{k_{priv}}(m)$ is valid signature on message m. Agents and hosts can use DS for the signing of transactions. Again using the airline agent as an example, once the agent has determines that a host has made the best offer, DS then can be used by the agent and the host to make the transaction non-reputable.

Message Authentication Code: *Message Authentication Code* (MAC) is the hashing of a message m combined with a key k. Each unique k will produce a unique MAC on m. Let h be the hash function, then $h(h(m, k))$ is the MAC of message m using k. The one way property of h ensures that each m using k, produces an unique MAC. For example, a host can produce a MAC of its agent before dispatching the agent. Upon return of the agent, the owner can again calculate the MAC to verify the integrity of the agent.

Obfuscation: The obscuring or hiding of the functionality of code. Typically, obfuscation is used to obscure proprietary agent code. CEF [32] and code scrambling [27] are examples of code *obfuscation.*

Masquerade: Is when an entity is disguised as another entity. Agents with limited permissions might *masquerade* as another agent with more permissions to gain access to resources on a host which the agent otherwise wouldn't have. A host might masquerade as another host to lure an agent into revealing private information such as credit card numbers and private-keys.

CHAPTER II

MOBILE AGENTS

A mobile agent is a multi-threaded autonomous program that can be distributed over a heterogeneous network to perform some predefined task for its human creator (user). In this chapter, a more formal description of a mobile agents will be given with respect to its mobility, composure, benefits and applications. A more in-depth discussion on security issues of the paradigm will be covered in section 2.4. Since this thesis focuses on agent security rather than the environment which it executes (host), the terms *mobile agent* and *agent* will be used interchangeably.

2.1 *What is a Mobile Agent?*

To answer this question, a clear distinction must first be made between a *mobile agent* and *mobile code* because the two terms are often used synonymously. Fuggetta *et al.* [9] considers mobile code as any program which exploits code mobility. Therefore, *mobile agents*, *Remote Procedure Call* (RPC), *Remote Evaluation* (REV), *Code on Demand* (COD), Java ^{TM}applet and other code written for distributed applications fall within the definition of mobile code. In Figure 1, a model of a mobile agent is given. The mobile agent consists of Execution Code (EC), Execution State (ES) and Storage Space (SS) which together makeup an Execution Unit (EU). EC is the knowledge or *"know-how"* required for computation. SS encapsulates initialization data and computational results. ES includes information on the program counter, registers and stack information required by an agent to execute or resume execution after it has been suspended either for migration or hibernation. The ability to suspend state, and resume execution on another host is known as *strong migration*[9].

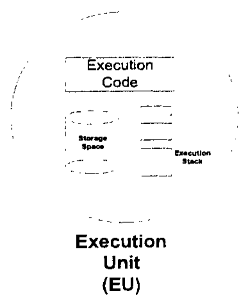


Figure 1: *A Model of mobile agent as an EU.*

One of the key distinguishing features between the above mentioned mobile codes is how each communicate between their respective *originators*, otherwise known as their level of migration. As mentioned earlier, strong migration is the ability for mobile code to migrate with state information so that execution can resume on another host. However, in each of the client server models mentioned above, code mobility is restricted to the passing of procedures and data, or *weak migration*[9]. For example download Java^{TM}applet contains code and data, but an applet lack the ability to suspend execution and migrate to another host to resume execution. In Figure 2, similarities and differences of the different mobile code paradigms are shown. Figure 2a-c show the more traditional client-server models of code migration. Figure 2a, is the communication model of RPC, parameters (e.g. data and threshold of computation) of the request is sent from the client to the server which hosts the know-how and computational resources. The server executes the request before returning results to the client. In the COD model of Figure 2b, the client which hosts the computational resources and data, requests the know-how from the server. Once the client receives the know-how from the server, execution is carried out on the client's machine using local computational resources. Figure 2c, the REV model, the client provides the know-how and data, execution is carried out by the server before results are returned to the client. In each of the these synchronous models, the client initiates contact with the server and blocks for a response from the server. Figure 2d, is the mobile agent model of communication, the User dispatches the agent which has the know-how, storage (storage encapsulates execution results and if necessary initialization data) and execution state information. In this model, each host provides the data and computational resources required for execution. The agent migrates from host 1 to n autonomously and communicates with the user only upon its return from host n.

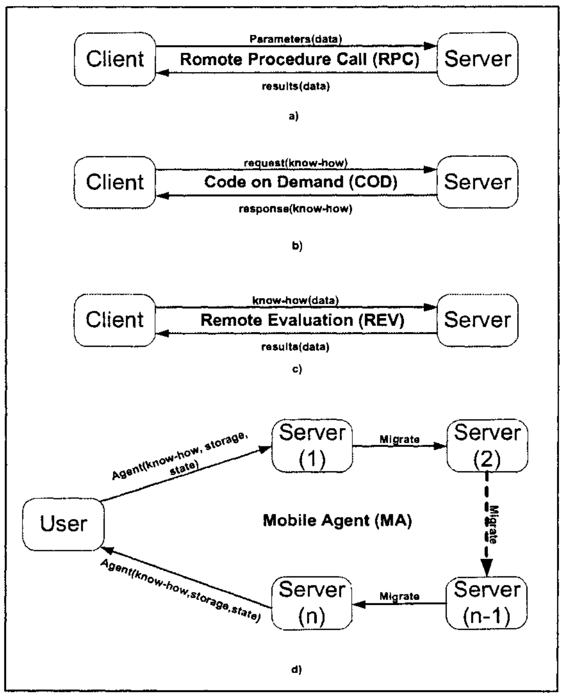


Figure 2: *Comparison of communication models between Remote Procedure Call (RPC), Code on Demand (COD), Remote Evaluation (REV) and Mobile Agent (MA).*

In the more traditional mobile code paradigm such as in distributed systems, code migration is designed for relatively small scale networks. Code is passed from one machine/processor to an other. To the programmer, these actions are seamless and transparent because they are handled at the operating system level. Location of execution is never a concern to users of such system, as far as they know, everything appears to be executing on their local machine. In the left diagram of Figure 3, each EU resides within the distributed execution environment (In distributed computing environments EU can be considered as individual processes or a thread of a multi-threaded process). In contrast, the mobile agent paradigm is designed with the vast scale of the internet in mind. Agents migrate independently from host to host and must always be aware of its location and the identity of its current host (Right diagram of Figure 3).

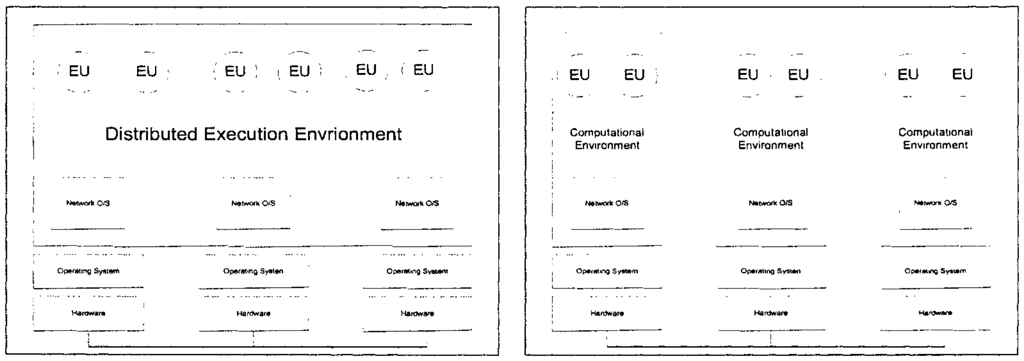


Figure 3: *Distributed execution environment (left) Vs. mobile agent computational environment (right).*

2.2 Why Use Mobile Agents?

Various authors have studied benefits of the mobile agent paradigm [6, 18, 9, 15]. The consensus between them seems to be that the mobile agent paradigm offers greater flexibility, scalability and transportability over the traditional client-server paradigm. The next few sections will summarize some of the most important features that make the paradigm so promising.

Overcomes network latency. Network latency refers to the time it takes a packet to get from the sender to the receiver. In networks with high noise to signal ratio or networks with low bandwidth connections, network latency can be substantial. Because a mobile agent reside on the target host, network latency is rarely a factor during its execution.

Loyalty: An agent is created to perform computations on behalf of its owner. The agent represents the owner on the network, therefore, performs only tasks which the user assigns.

Reduces network traffic: Often, results collected by an agent is much smaller than the data required for its computation. Moving an agent to the source of data reduces not only the amount of data to be sent over a network, but the number of rounds of communication between the client (agent user) and remote hosts.

Execute asynchronously: Agents once dispatched can operate independently and autonomously without further interactions with its creator. This unique ability is particularly well suited for networks with unreliable or expensive connections.

Dynamic adaptability and fault-tolerant: An agent can sense and adapt to their environment. If a host is unable to provide the requested service, an agent has the ability to migrate to another host. In the event a host environment becomes unfavorable, an agent can migrate to another host or hibernate until favorable conditions are again available.

Operate in heterogenous networks and environments: An agent typically operating in a Virtual Machine (VM) execution environment independent from host hardware. Agent migration protocols protect agents against network incompatibilities.

Flexible and scalable: The typical client-server model of computing limits a client to a static set of services offered by a server. The mobile agent paradigm overcomes these limitations because it allows the user of an agent to design the agent to suit their specific set of needs.

2.3 MA Applications

The goal of the mobile agent paradigm is the eventual adoption by the internet community for use in various research and commercial applications. Ever since the introduction of mobile agent paradigm [36], there has been ongoing discussions about its application domain [24]. Various *"killer applications"* has since been proposed, below are some of well acknowledged applications of the mobile agent paradigm. If possible, security requirements of each will also be included.

2.3.1 E-Commerce

Mobile agents for use in e-commerce is a particularly popular idea [35, 31, 17]. By dispatching agents to seek out products or services not only reduces network traffic but can also save the agent user considerable amount of time. Once created, an agent is independent from its owner, depending on the design of the agent, it will seek for the products or services and communicate with its owner only upon completion of the task or when some criteria set by the owner is meet. Contract negotiations, service brokering, auctions and stock trading are just some of the applications mobile agents are well suited for [9].

In most instances using a mobile agent for e-commerce applications requires the highest levels of security. For example, a *"shopping agent"* might have to carry electronic cash, credit card information and private signature keys to sign contracts. In this case, the owner would obviously want to protect his/her assets including results collected by his or her agent. Another type of shopping scenario agent is the *"window shopper"*. Although such agent might not be carrying any *"secret"* financial assets, but as mentioned in the airline ticket scenario, bids gathered by such agent still require protection from spying hosts.

2.3.2 Personal Assistants

An example of a mobile agent used as a personal assistant is the *meeting agent* [18] dispatched for setting up a meeting between a number of participants. In this scenario, several people need to arrange a meeting time that suits the schedules of all participants. Each participant can create an agent preloaded with acceptable times which he/she can meet. The agents can then meet to make the necessary arrangements. If an agreement can be made, each agent returns to its respective owner with the scheduled time for the meeting.

Security for such agent depends on sensitivity of the data (schedule of individual participant) and the level of security sought by their respective owners. In cases where participants wish to keep their data hidden, arrangements can be made so that negotiations takes place on a mutual trusted host often referred to as a *Trusted Third Party* (TTP).

2.3.3 Network Management

The use of mobile agents for remote network administration can be argued as the *"kill application"* for most network administrators. A mobile agent's ability to adaptively respond to network events in real-time make them a very powerful tool for system monitoring and hardware reconfiguration. An agent's mobility and independence can also be useful for distribution of software on the network wide bases.

An adaptive agent (*intelligent agent*) over time can learn the normal behavior of the resource it's monitoring. Such agents can be dispatched to monitor vital network resources for *"abnormal"* behavior so that preventative measure can be taken prior to the occurrence of catastrophic events. If an agent senses abnormal behavior, the administrator can be notified, and another *"maintenance agent"* could be dispatched with preloaded configuration and support software to perform required services. Thus eliminate the need for such software to be installed on every resource. The maintenance agents can also be used for applying software patches, reconfiguring of existing applications and installation of new applications as they become available.

An agent used for network management is vulnerable to both internal and external threats. The most likely internal treat is disgruntled employees assuming that all hosts within the network are trusted. A disgruntled employee with access to internal system resources can potentially modify or delete an agent. External threats to such agent usually depend on the resilience of the firewall protecting the network.

2.3.4 Information Retrieval

There are many benefits of dispatching MA for information processing and retrieval [18]. *Information agents* have the ability to process information without being confined by operational time of their owner. Distributing multiple agents to process or search for information increase the scope of the of coverage while reducing the time required to complete the task. Information retrieval and processing typically involve the processing of large amounts of data to obtain relatively small amount of results. Again, by sending agents to the source of information, only the results need to be transported by the agent, thus reducing load on the network infrastructure.

The level of security required for an information retrieval agent depends on the sensitivity of the information being collected. Information gathered on publicly available sources might not require protection, but encryption still can be used if the agent owner does not wish to disclose the type of information sought by the agent. For example, if a commercial enterprise is seeking data for a new product or service under development. Although the data gathered are from publicly available sources, there would still be the need to hide sources and composure of such data from competitors.

2.3.5 Mobile Devices

Perhaps the most natural application of a mobile agent is in mobile devices such as cell phones, personal organizers, notebook computers, etc... Most mobile devices rely on wireless technology which at best is sporadic in availability and have limited bandwidth. Another inherent weakness of mobile devices with the exception of notebook computers is their limited processing power and storage capacity. User of mobile devices can dispatch an agent when network connection is available and collect it at a later time when network connection is once again available. Thus eliminating the need to transfer large amount of data to the device to be processed locally.

As with all applications of mobile agent, security requirement of the agent depends on its functionality and the sensitivity data within the agent. However, mobile device users must always take into consideration the limitation of their devices, particularly the lack of processing power and storage capacity when making security related decisions.

2.4 MA Security Issues and Countermeasures

Many functional mobile agent systems have already been implemented [10, 7, 21, 12]. However, their use has yet to be widely adopted outside the research and academic arenas mainly because of the many unresolved security related issues. Mobile agent systems as with any network computing platforms have four core security requirements [15]. First, both agents and platforms must have *confidentiality* protection against potential eavesdroppers and thieves. Second, security policies must provide for varying levels of code and data *integrity*. Third is the *accountability* of both agents and hosts for any damages caused by malicious actions or poorly written code. Finally, agent platforms must ensure *availability* of resources for agents (e.g. CPU cycles, network bandwidth, temporary storage space etc...) if prior contractual agreements bound hosts to do so.

As discussed earlier, the introduction of mobile agents into a network posses many security related issues. An agent typically executes on remote potentially hostile environments. This leaves the agent open to various types attacks by malicious hosts and hosts to attacks by agents [2, 14]. The security domain of the mobile agents paradigm can be broken into two broad categories.

1. Securing remote hosts from malicious agents.
2. Securing agents from malicious hosts.

Many solutions to security issues related to the mobile agent paradigm have been proposed by various authors, unfortunately, they are *ad hoc* in nature because of the absence of well defined set of specifications. The rest of this chapter will discusses some these security issues for both hosts and agents along with various proposed methods for countering them.

2.4.1 Host Security

Any host which offers external agents access to its resources faces many serious threats from a malicious agent [14, 15, 2]. The attacks a mobile agent host might face are similar to attacks any traditional host connected to a network faces. These include: *masquerading*, *denial of service* and *unauthorized access.* Fortunately, many methods to protect against a malicious agent can be adapted from the traditional client-server model of security. Mechanisms for process isolation, resource access control are already in place on most hosts. Well known and proven cryptographic techniques for encryption/decryption, signatures and authentication are readily available for use by every host.

Bierman and Cloete [2] classified countermeasures to agent threats as suited either for *prevention* or *detection.* The same classification will be used here for countermeasures classification of host treats as well. For a host, the first and obvious goal is to detect malicious or poorly written agent code prior to its execution. However, this is not always possible or reliable. Therefore, the goal of prevention is to contain or minimize the effects of such code.

Detection: An example of detection is the static type checking of downloaded Java^{TM} byte code implemented in the Java^{TM}security architecture [16]. Before a downloaded applet is permitted to execute, the class loader invokes the verifier which checks for memory management violations such as stack overflow, underflow and illegal type casts within the applet code. Signed code can also be used as part of an overall detection strategy. Allowing only signed code from verifiable trusted entities to execute does not guarantee code integrity, however, it does allow a host to seek retribution in cases where the code has caused damages. Agents which record their itinerary[5] and can prove that they've already executed on other hosts trusted by the current host can also give some assurance on the integrity of the agent code.

Predictably, the above mentioned methods for detection all have its associated down sides. It might be computationally infeasible for an host to examine every line of code for every agent prior to execution. There are also instances where the host might learn nothing on the intent of the code prior to execution if the code has been intentionally obfuscated [11] or encrypted [32]. To require every agent to be signed could mean that many un-signed *non-malicious* agents would have to be turned away thus limiting wide adaption of the mobile agent paradigm by the general public. Agents that have executed on other trusted hosts are still subject to tempering on its migration from one host to another. For example, if the itinerary of an agent shows that the agent migrated from a trusted host, then to an un-trusted host, prior to its migration to the current host. How confident is the current host that the agent hasn't been tempered with by the un-trusted host? This issue can obviously be resolved by limiting access to agents with an itinerary of only trusted hosts. However, this again limits the usefulness of the mobile agent paradigm.

Prevention: A key requirement of host prevention is ensuring agents cannot interfere with other agents or their environments. This can be accomplished through the use of what's know as *reference monitors* to establish isolated domains for each agent and to control all resources access by an agent. Reference monitors uses a number of traditional security techniques such as: process isolation, resource access control and cryptographic primitives for information enciphering and authentication. More recently, the use of interpreted programming languages has also offered an additional layer of abstraction against agents. The most popular are Java™[16] and SafeTcl [26].

The Java ™programming language implements what's known as a *sandbox* model of security. As mentioned earlier, static type checking of download code is a powerful tool for prevention of attacks. For detection, Java ™uses what's called a Security Manager. Similar to a reference monitor, the Security Manager control all access to system resources during runtime.

SafeTcl is an interpreted language used by AgentTcl [10]. SafeTcl uses a padded cell concept where un-trusted or suspicious code is executed within the padded cell before being allowed to be executed within the main interpreter.

Unlike detection techniques, prevention techniques are more well established and are widely implemented. The overall security of any host system often depends on an organization's security policy and their willingness to follow through on its implementation. Security breaches even in well established systems and languages occur frequently due to lack of willingness to implement security policies already in place.

2.4.2 Agent Security

Agent security is widely believed to be more challenging when compared to host security. This can be partially attributed to the fact that the mobile agent paradigm is relatively new and partially to the autonomous encapsulated nature of an agent.

Treats against an agent from a malicious host can be categorized into four categories of: *integrity attacks*, *denial of service*, *confidentiality attacks* and *authentication risks* [2].

Integrity Attacks: Violations to an agent's code, data or states are considered integrity attacks. There are two prevalent categories of violations, *active* and *passive*. Example of passive violations occurs when a host intentionally delays or refuses the transmission of an agent. More serious are active violations when an host manipulates an agent's code, state or data to its benefit.

Denial of Service Attacks: Denial of service attacks occurs when an host *denies* an agent access to its resources. For example, a host might have previously been compensated to provide a certain set of services for an agent but, upon arrival of the agent, the host refuses the agent the previously agreed services. There are many reasons for a host to refuse agent such services but the most prevalent is the potential for financial gain.

Authentication Risks: Consequence of agent autonomy is the need for an agent to be more location aware than its traditional client-server counterpart. In the traditional mobile code paradigms of Figure 2, connection is established by the client to the server. In each of the model, authentication can proceed any transmission of code. For a mobile agent, authentication typically occurs after the arrival of the agent. Agents then are susceptible to masquerading attacks by a host. For example, suppose the airline agent believes its being hosted by a trusted major airline, but instead it has been *"hijacked"* by the FlyByNight host disguised as the major airline host.

Confidentiality Attacks: An agent is often embedded with private or sensitive information belonging to its owner. To compromise an agents confidentiality implies that the agent is either illegally accessed or its privacy is under attack.

The most obvious is a host which eavesdrops on an agent to steal information such as private keys or credit card information. However, proprietary algorithms of an agent could also be target of such attacks.

Agent security similar to host security should also implement two lines of defence. However, for an agent, the first line of defence should be the prevention of possible attacks by malicious hosts. In the case where prevention is unattainable due to factors such as technological limitations. Then detection mechanisms in terms of result authentication should be used to validate results obtained by an agent.

Prevention: The aim of prevention is to thwart possible attacks by a malicious host. Arguably, prevention of malicious host attacks is perhaps the toughest security challenge in the mobile agent paradigm. The various methods dealing with prevention of malicious hosts attacks are based on *trust-based computing*, *recording and tracking* and *cryptographic primitives.* The first two techniques will be discussed here while methods of using cryptographic primitives is reserved for Chapter 3.

Trust based computing relies on the availability of Secure Hardware (SH) to provide Secure Execution Environment (SEE) for an agent (see Chapter 4 for more details). Devices such as secure coprocessor [39], tamper resistant hardware [38] and SmartCards [19] have all been proposed to provide the SEE. However, such hardware are either theoretical or under development.

Recording and tracking mechanisms have also been proposed. The idea of using cooperating agents [30] is that two agents are used to track each others migration. Each agent forwards the location of previous, current and next hosts to the other co-operating agent. In essence each agent is keeping an itinerary of the other agent. If either of the agent senses inconsistencies, then appropriate preventative measures can be taken. This idea is based on the premiss that not all hosts are malicious and that at least one of the agent is being hosted by a *"trusted host"*[1] at any given time.

Detection: Without adequate preventative mechanisms to protect an agent, user of an agent is confined to detection mechanisms to determine authenticity the agent and its results. Conceptually, the simplest method for detection of tempering is the use of *"detection objects"* [22]. Detection objects are fictitious data closely matching results an agent might collect on remote hosts during its migration. For example, consider the airline ticket agent. The user of the agent can implant some fictitious offers within the agent prior to dispatching the agent onto the network. Upon the return of the agent, the user checks for the authenticity of the implanted results. The premiss is that if the implanted results are intact, then the agent and results are presumed to be authentic.

Agents can also record their path history to detect tampering [25]. For instance, as hosts are visited by an agent, the agent records the locations of the current and next host to be visited by the agent. Each entry is signed by the current host verifying the migration of the an agent. Upon return of the agent, its owner can verify the path taken by its agent. If the agent migration path strays from the path recorded, then the agent has been tampered with.

2.5 Remarks

This chapter covered some of the most important traits of the mobile agent paradigm including it's benefits and mostly security related drawbacks. The above countermeasure given to protect an agent all lack cryptographic strength. The next chapter will focus strictly on cryptographic techniques for the protection of agent.

[1]Chapter 4, discusses the concept of a trusted host. See chapter 4.1

CHAPTER III

CRYPTOGRAPHIC PRIMITIVES FOR MOBILE AGENT

This chapter describes different techniques for protecting an agent using cryptographic primitives. As previously discussed in section 2.4, protection of an agent first involves protection then prevention. This chapter will first describe techniques for protection followed by prevention. The use of cryptographic primitives offer a new set of challenges for an agent. The user of an agent cannot offer the level and trust security the agent enjoys in its home environment. Once dispatched an agent is under total control of the remote host. Cryptographic key even if it can be hidden from the remote host, cannot be protected if the agent is to use such key.

3.1 Cryptographic Prevention

3.1.1 Computing with Encrypted Functions

Prior to the introduction of *Computing with Encrypted Functions* (CEF) by Sanders and Tschudin [32], it was widely believed that software only solution to the problem of malicious host was impossible. CEF allows an agent to execute on remote potentially hostile environments autonomously without disclosing any information on the function of the agent or any cryptographic primitives within the agent. The basis for CEF is the use of a *homomorphic* PKE scheme that allows for non-interactive addition or multiplication of two encrypted cleartext messages through the manipulations of ciphertext only. Given the pair of functions e, d which are the encryption and decryption functions respectively and two ciphertext messages $e(x)$ and $e(y)$. Homomorphic encryption allows for the efficient calculation of $e(x + y)$ and $e(xy)$ without disclosing either values x or y. To illustrate, Alice A has a function f which Bob B is willing to execute for her with his input x. To hide the functionality of f from Bob, A encrypts f into another function $e(f)$. A then creates a program $P(e(f))$ which implements $e(f)$ and the procedures ADD and $MULT$, sends them along with $P(e(f))$ to B. B then executes the program $P(e(f))$ on x to obtain $P(e(f(x)))$, each call for addition uses the procedure ADD and multiplication uses $MULT$. Once execution of $P(e(f))$ is completed, B sends $P(e(f(x)))$ back to A who decrypts it to obtain $f(x)$. If f is a signature or encryption algorithm then A has effectively signed or encrypted information respectively without disclosing her private key (see Figure 4).

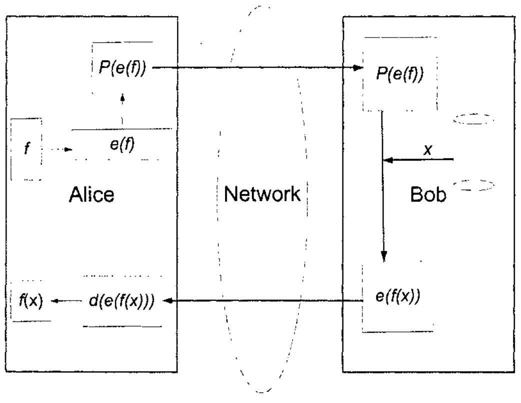


Figure 4: *Computing with Encrypted Functions.*

3.1.1.1 Undetachable Digital Signatures

The use of CEF was also extended to producing Undetachable Digital Signatures (UDS) where a signature routine can be *"attached"* to the function f such that

$$f_{signed} = s \circ f$$

Let f, s be rational functions and s is used by A to sign the message m where m is the output of f on some data x.

$$m = f(x)$$

Let v be the verification function of s, where

$$v(s(f(x))) = m$$

and v is public. To make the signature undetachable, A sends (f_{signed}, f) to B. B uses his input x to produce $(f_{signed}(x), f(x))$. Using the public verification function v anyone can verify that,

$$v(f_{signed}(f(x))) = f(x)$$

3.1.1.2 Remarks

CEF provides protection for both the agent and the host. The functionality of the agent is protected because of the encryption of f. Hosts are also protected because f is executed with the input x controlled by the host. CEF has since been shown to be useful for all *polynomial-time* functions by Cachin *et al* [4] and Kotzanikolaou *et al* [27] showed that it can be used with the RSA signature scheme. However the construction of encrypted functions is complex and has yet to be shown to work with a more wide range of functions.

Algeheimer *et al.* [1] later showed that non-interactive secure mobile agent computing schemes do not exist. Non-interactive refers to no interactions between the host and agent during the agent execution. CEF allows the secure evaluation of f to produce only a final result. Therefore, agents cannot react to host actions during its execution. Using the airline agent example, suppose the agent's previous best offer is c. If the output of f is to accept or reject the offer x by the host. Then there is nothing that keeps the host from continuing to make an offer x' until c has been completely exposed by the agent.

3.1.2 Sliding Encryption

Sliding Encryption [40] is a deterministic method for operations on public key cryptosystems. It allows for encryption of small amounts plain text into same size cipher text. A general description of the method will be given here. More details can be found in [40].

Let m be the size of a RSA key and m is a power of 2. Plain-texts are broken into small equal sized blocks $a_1, a_2, \ldots, a_k$ where $\forall i, 1 \leq i \leq k$ and $|a_i| = u$. Let v be the size of some random string and $t = u + v$ where $t \ll m$ and t divides m. To perform slide encryption requires the use of a stack $\mathcal{S}$, an accumulator $\mathcal{A}$ and an window $\mathcal{W}$. Elements of $\mathcal{A}[i]$ and $\mathcal{W}[i]$ are of size t. Let e, d be the encryption and decryption functions respectively. To encrypt, a_1 is put into the lower order of bytes

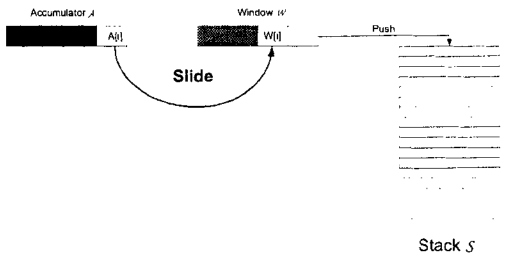


Figure 5: *Slide Encryption.*

of $\mathcal{A}[1]$, the upper v order of bytes contain random strings. The accumulator is then encrypted to obtain $e(\mathcal{A})$. $\mathcal{A}[1]$ is then slid into $\mathcal{W}[m/t]$. To encrypt a_2, a_2 is put into the lower order byte of $\mathcal{A}[1]$, the upper v order of bytes contain random strings. The accumulator is then encrypted and $\mathcal{A}[1]$ is slid into $\mathcal{W}[m/t - 1]$. This process repeats until $a_{m/t}$ has been slid into $\mathcal{W}[1]$ at which time $\mathcal{W}$ is pushed onto the $\mathcal{S}$, $push(\mathcal{W})$. The plain text $a_{m/t+1}$ to $a_{2m/t}$ uses the same process. This repeats until all k plain text blocks are encrypted. Decryption is the exact reverse of the encryption.

3.1.2.1 Remarks

Slide encryption was designed with minimizing agent code in mind. A smaller agent gives the agent better mobility, but because the decryption key stored in one location, any breach of security at that location would allow an adversary access to the plain text.

3.2 Cryptographic Detection

3.2.1 Partial Result Authentication Code

Partial Result Authentication code (PRAC) was proposed by Yee [39]. PRAC, requires that the agent and owner maintain a set of secret keys incrementally generated on each host the agent visits. The secret key is used to generate a cryptographic checksum (similar to MAC) of an agent's state and results on each host it visits. Once used, the secret key is destroyed by the agent before moving to the next host, thus insuring forward integrity. More formally, suppose that an agent is to visit a list of servers $s_1, \ldots, s_n$, if the agent encounters a malicious server s_c where $c \leq n$, the information collected at $s_1, \ldots s_{c-1}$ is preserved. Because only the agent and its owner know the secret keys and the agent's copy is destroyed after use, only the owner of the agent can authenticate any results.

3.2.1.1 remarks

The maintaining of the secret keys by the owner requires an agent to contact its owner from each host it visits. This reduces agent autonomy and because the keys are stored in one location (owner) any breach of security at that location could potentially reveal all the secret keys.

3.2.2 Cryptographic Traces

Vigna [35] proposed a process where the executing host is required to produce a *trace* of operations performed by the agent. The PKI is used for the generation of a hash and securing of communication between the agent owner and executing host. During agent execution, the host records (logs) the operations performed by the agent. Upon termination of execution, a hash of the agent operations and is current state is calculated. A copy of the trace is stored by the host so that it can be produced later at the request of the agent owner. The hash is then forwarded to the agent owner. Thus allowing an agent owner to re-execute the agent with data supplied by the host and compare the hash of the re-execution with the one supplied by the host. To produce a cryptographic trace requires multiple rounds of communication between the owner of the agent and the host executing the agent. Figure 6 illustrates the rounds of communication to produce the execution trace.

1. $A \xrightarrow{m_1} B \mid m_1 = A_s(A, B, i_A, t_A, K_A(p))$
2. $B \xrightarrow{m2} A \mid m_2 = B_s(B, A, i_A, h(m_1), M)$
3. $A \xrightarrow{m_3} B \mid m_3 = A_s(A, B, i_A, B_p(K_A))$
4. $B \xrightarrow{m_4} A \mid m_4 = B_s(B, A, i_A, h(m_3))$
5. $B \xrightarrow{m_5} A \mid m_5 = B_s(B, A, i_A, K_B(S_B), h(T_B^p), t_B)$
6. $A \xrightarrow{m_6} B \mid m_6 = A_s(A, B, i_A, h(m_5))$
7. $B \xrightarrow{m_7} A \mid m_7 = B_s(B, A, i_A, A_p(K_B))$

Figure 6: *Cryptographic Tracing Steps.*

In Figure 6, Alice A is the owner of the agent and Bob B is the executing host. Let m_i be a message where $1 \leq i$. Let (A_p, A_s) and (B_p, B_s) be A and B's public and private keys respectively and $A \xrightarrow{m_i} B$ is the passing of m_i from A to B. Let $h(m_i)$ be the hash value of m_i and h is a one way hash function. Singing (e.g. Produce a DS) of a message by A or B is indicated by A_s or B_s respectively.

Step 1: In Figure 6, the message m_1 is from A to B as indicated by the first two fields of each m_i. The singing of all m_i insures authenticity of messages to the receiving party. The third field i_A is a unique identifier for all messages during this execution request. A time stamp t_A is also included in m_1 to ensure *"freshness"* of the message and to guard against *reply-attacks.* The last field, $K_A(p)$ is the program p (agent) to be executed by B encrypted with a secret key K_A chosen by A.

Step 2: B has options of either to accept or reject the request which is contained in the message M. A hash of m_1 is returned to the agent owner to ensure to A that m_1 was received by B correctly.

Step 3: If M is an accept message, then A responds with m_3 which contain the K_A required by B to decrypt the program p.

Step 4: An acknowledgment message m_4 is sent by B to A indicating that B has successfully decrypted p. Again a hash of the previous message received by B is returned to A. Now that p has been decrypted, B proceeds to execute p and the trace, T_B^p is recorded by B.

Step 5: Once p has completed execution, S_B, the state (results of execution) of p, is recorded and encrypted with a random key K_B to obtain $K_B(S_B)$. Then a hash $h(T_B^p)$ is generated along with another time stamp t_B. B then proceeds to send m_5 to A.

Step 6: When m_5 is received by A, A replies with an acknowledgement message m_6 requesting for K_B to decrypt the state of agent S_B.

Step 7: If all previous steps were successful, B sends A m_7 which contains K_B. A can now decrypt state of the agent to obtain the state S_B.

If A suspects that B has cheated while executing p, A can request that B send T_B^p and the data used to produce T_B^p. A first computes $h(T_B^p)'$ and compares it with

$h(T_B^p)$ received in m_5 to check the authenticity of T_B^p. If the $h(T_B^p)' = h(T_B^p)$, A executes p to produce $T_B^{p}{}'$. If $T_B^{p}{}' \neq T_B^p$, A knows that B has cheated.

3.2.2.1 remarks

The limitation of using cryptographic tracing is the large number of rounds of communication required per execution. This somewhat undermines some of the key benefits of the mobile agent paradigm of autonomy and independence.

3.3 Remarks

The techniques discussed in this chapter all provide an agent with the capability to use cryptographic primitives on a remote host. CEF aims not only to provide privacy for the agent data but also any potentially proprietary agent algorithm. However, as mentioned earlier, the generation of the encrypted function is only limited to polynomial functions. SE offers an agent better mobility by limiting the size of encrypted data. However, the strength of the scheme depends on the securing of the private key. The next chapter introduces *Secret Sharing Scheme* (SSS) for mobile agent cryptography. Although, its aim is not as bold as CEF, its application is similar to SE but because the private key is shared, the compromising of the private key is much more challenging for an adversary.

CHAPTER IV

SSS FOR AGENT CRYPTOGRAPHY

This chapter will introduce the idea of using SSS for mobile agent cryptography. As mentioned earlier, the distribution of the private-key amongst n provides additional protection against a adversary. This chapter will outline the protocol for initialization, distribution and hiding of shares within agents. Later in the chapter, a modified version of the protocol will be introduced which allows an agent on remote *trusted hosts* (TH) to distribute agents of own.

4.1 Trusted Hosts

Here, TH will be defined as any host which can provide an agent with a *Secure Execution Environment* (SEE) including protection against the following attacks as categorized by [2]. *Integrity attacks*, an agent is secure from tampering by the host and other entities such as other agents. *Availability refusal*, an authorized agent cannot be denied access to resources which has been universally agreed upon to be vital to the proper function of an agent such as, CPU cycles, RAM, storage space and network bandwidth to migrate to other hosts. *Confidentiality attacks*, an agent private assets cannot be destroyed, tempered with or made public by the host or other entities. *Authentication risks*, when requested, a host must be able to provide appropriate credentials (e.g. Certificate from a CA) so that the agent can authenticate the identity of the host.

It is assumed that the home environment of an agent is trusted. Therefore, the first and obvious TH is an agent's home environment. The process of initializing agents for distribution involves the generation of a pair of public and private keys.

The importance of keeping the private-key secure is obvious, therefore, the key generation process will only be performed on a TH. All hosts are classified as either *trusted* or *non-trusted.* TH are host which can provide an agent with a SEE mentioned above. Classification of a host as un-trusted does not imply maliciousness, but rather, it cannot provide an agent with an execution environment that fulfills all the requirements set forth by SEE.

It is still an open question whether SEE is possible with only software based solutions by using current technology [1]. Some believed that such environment will only be available through specialized *Secure Hardware* (SH) strictly designed for agent execution [39, 19, 37]. In [39], a SH in the form of a secure coprocessor is under development which will allow for a SEE for Java ^{TM}agents. In [20], a limited capacity Smartcard was proposed that interacts with an agent's code to hide the functionality of the code from a host. In [37], the proposed hardware provides agent code a *tamper proof environment* in which to execute.

The proposed SH all rely on cryptographic primitives of PKI. The common principle behind each of the above mentioned SH is the use of PKE. Each SH has its own associated public/private key pair. The public-key is made public while the private key is embedded into the SH either during the manufacturing process or by the manufacturer after its production. The assumption is that SH manufacturers are large multi-national corporations with valued global reputation which they wish to protect. Another obvious key assumption of SH is that the host system employing the SH cannot tamper with the SH or gain access to it's private-key. Here is a example use of SH. Suppose an agent owner wants to dispatch an agent to n hosts $H_1 \ldots H_n$. Once the agent is created, the owner obtains the public-key of H_1, encrypts the agent before sending the agent to H_1. When H_1 receives the agent, it decrypts the agent with its private-key, then proceeds to execute the agent. Upon completion of execution of the agent, H_1 encrypts the agent with the public-key of H_2 before send the agent to H_2.

This process continues until H_{n-1}, at which time the public-key of the owner is used to encrypt the agent. This process has effectively hidden the function of the agent along with any secrets the agent might be carrying.

Although SH theoretically offer unambiguous protection for an agent, unfortunately such hardware is not yet available. However, for ease of discussion of the protocol that follows, the assumption will be made that such hardware is available and is employed by limited number of hosts. The purpose for SH is to extend the security of an agent's home environment to a remote host. Therefore, any solution that can offer an agent SEE can be used, including non-technological social means. For example, a certain number of dedicated hosts which are socially bound to offering secure resource access to agents such as a University network, or financially bound such as paid TTP. A concept often discussed in the mobile agent paradigm to act as an impartial mediator [38, 35].

4.2 Secret Sharing

Secret sharing schemes (SSS) was introduced independently by Shamir [33] and Blakey [3] in 1979. SSS allows a secret to be distributed by a trusted dealer to a set of participants which permits only specific subsets or a threshold number of participants to recover the secret.

4.2.1 Shamir's Threshold Scheme

Shamir uses a polynomial (t, n)-threshold scheme, where t, n are positive integers and $t \leq n$. A secret value k can be shared by a trusted dealer D amongst a set of participants P, where $P = \{P_i | 1 \leq i \leq n\}$ (See Appendix A, Figure 12). The recovery of k requires at least a set of M participants where $|M| \geq t$ and $M \subseteq P$ (See Appendix A, Figure 13). For instance, let $t = 3$ and $n = 5$. The value k is shared amongst 5 participants and any 3 can regenerate k using their shares.

4.3 The Protocol

The goal of agent autonomy is to minimize number rounds of communication between an agent and its *Originator*[1] O. Therefore, the protocol's aim is to allow agents to use cryptographic primitives on remote hosts while restricting communication between O and each agent to only one round (send and receive). More precisely, let O be the originator (creator) of n agents, $1 \leq n$. Let C the number of rounds of communications between O and n agents. Therefore, the goal is such that $C = n$. Note that O recursively inherits any agents generated on remote hosts by its children. For example, if O generates n agents and the n agents combined generates g agents then O has $n + g$ agents.

4.3.1 Agent Initialization

Again, the point will be emphasized that agent initialization is to only be performed on a TH in order to protect the private-key. Agent initialization involves the generation of the public/private key pair which will be used for encryption and decryption respectively. Share generation is performed once the private-key has been created. To protect each share, a random number is generated for each share. The random number is combined with each share prior to distribution to agents. Figure 7 outlines the steps required for initialization.

Here, a formal description of agent initialization will be given. Let D be the trusted dealer in Shamir's algorithm, and D is the originator of agents $\mathbf{A}$, where $\mathbf{A} = \{\mathbf{A}_i | 1 \leq i \leq n\}$. Let H be a set of remote hosts, where $H = \{H_i | 1 \leq i \leq n\}$. Let $\mathcal{K}$ be the set of all possible keys and $\mathcal{K} = \mathbb{Z}_p$, such that $p \geq n + 1$ is a prime and $\mathbb{Z}_p$ is a finite field of size p. Let $\mathcal{S}$ be the set of all possible shares and $\mathcal{S} = \mathbb{Z}_p$. Let s_i be the shares given to each $\mathbf{A}_i$, $(\forall i, s_i \in \mathcal{S})$. A random number $w_i \in \mathbb{Z}_p$ is generated

[1]Note, in previous chapters, an agent's originator was referred to as *owner* or *agent owner*. In the protocol the dealer D is also the owner and originator of agents.

by D for each $\mathbf{A}_i$.

Let k_{pub} and k_{priv} be the public and private key pair generated by D respectively. Let sig_{priv} be the signature function using k_{priv}.

1. D generates a key pair (k_{pub}/k_{pri}) such that $\{k_{pub}, k_{priv}\} \in \mathcal{K}$.
2. D publishes k_{pub}.
3. D calculates shares s_i using k_{priv} for each $\mathbf{A}_i$.

$$s_i = k_{priv} + \sum_{j=1}^{t-1} a_j x^j \ mod(p)$$

4. D also generates a random number w_i for each $\mathbf{A}_i$ and combines each $s_i + w_i$ to obtain $z_i = s_i + w_i$. D then stores a copy of each w_i in a secure location.
5. D signs each agent, $sig_{k_{priv}}(\mathbf{A}_i||z_i)$ with k_{priv} before destroying k_{priv}.
6. D attaches with each $\mathbf{A}_i||z_i||sig_{k_{priv}}(\mathbf{A}_i||z_i)$ and transfers each $(\mathbf{A}_i||z_i||sig_{k_{pirv}}(\mathbf{A}_i||z_i))$ to H_i.

Figure 7: *Agent Initialization*

In step 2 of the agent initialization (Figure 7). The requirement that k_{pub} be published is to allow all agents access to the public-key used for encryption. The publishing of k_{pub} is to reduces the size of an agent. It is possible for agents to carry k_{pub}. In step 4, the random value w_i added to each s_i conceals the actual share s_i from un-trusted hosts. Also, by concealing s_i, in the event that a malicious host is able to capture all z_i, without the random values w_i, s_i will remain hidden. The signing of the agent in step 5 allows D to authenticate each $\mathbf{A}_i$ upon its return and k_{priv} is destroyed because it can be regenerated upon return of t agents.

4.3.2 Encrypting

Let $e_{k_{pub}}$ be the encryption function using k_{pub}. As results $R_{i,j}$ (Results collected by agent A_i on host j.) are collected by $\mathbf{A}_i$ on $H_{i,j}$, $j \geq 1$, before $\mathbf{A}_i$ moves to host $H_{i,j+1}$ (*Note that the movement of* $\mathbf{A}_i$ *from host* $H_{i,j}$ *to* $H_{i,j+1}$ *is for notational purposes only. Agent migration does not require an strict itinerary.*), $\mathbf{A}_i$ retrieves k_{pub} from the publicly accessible location and encrypts $R_{i,j}$ with k_{pub} to obtain $e_{k_{pub}}(R_{i,j})$.

4.3.3 Decrypting

Let $d_{k_{priv}}$ be the decryption function using k_{priv}. Let $ver_{k_{pub}}$ be verification function for $sig_{k_{priv}}$. Decryption requires at least t of n agents to return with their shares. Upon return of each $\mathbf{A}_i$, D verifies each $\mathbf{A}_i$, if $ver_{kpub}((\mathbf{A}_i||z_i), sig_{k_{priv}}(\mathbf{A}_i||z_i)) = true$ D retrieves the value w_i from the secure location and extracts each $s_i = z_i - w_i$. If $|\{i : ver_{pub}((\mathbf{A}_i||z_i), sig_{priv}(\mathbf{A}_i||z_i) = true\}| \geq t$, though polynomial interpolation (See Appendix A Figure 13) using each s_i, D regenerates k_{priv}, D can now use $d_{k_{priv}}$ to decrypt $d_{k_{priv}}(e_{k_{pub}}(R_{i,j}))$ to obtain $R_{i,j}$.

4.4 *Agent Distribution on Remote Trusted Hosts*

Suppose $\mathbf{A}_i$ is on a remote TH H_i, and $\mathbf{A}_i$ wishes to distribute l agents $\mathbf{A}'$ where $\mathbf{A}' = \{\mathbf{A}'_i \mid 1 \leq i \leq l\}$ from H_i. Let SS be $\mathbf{A}_i$'s storage space (See Figure 1). Let w'_i be a random storage location within SS generated by $\mathbf{A}_i$ and y_i is the contents of SS at location w'_i, $y_i \in$ SS (*See Figure 14*). Let $W = \{w'_1||w'_2||\ldots||w'_l\}$ and $W \in \mathbb{Z}_p$. Let $\mathcal{S}'$ be the set of all possible shares and $\mathcal{S}' = \mathbb{Z}_p$. Let s'_i be the shares given to each $\mathbf{A}'_i$, $(\forall i, s'_i \in \mathcal{S}')$.

In the original protocol, the random values w_i was stored in a secure location on D's home environment. However, now $\mathbf{A}_i$ is on a remote host and must migrate at least once (back to its home host). Therefore, the random number generated to mask shares must be hidden within $\mathbf{A}_i$. To accomplish this, instead of masking each share with the random number w_i directly. The modified protocol generates a random storage location w'_i within SS. The contents y_i at location w'_i is used to mask each s'_i (See Figure 14). The combined randomly generated memory locations W is also shared amongst l agents as r_i. Figure 8 illustrates the modified protocol for agent distribution on remote TH. Note that in Figure 8, $\mathbf{A}_i$ is represented by D.

Let w_i' represent a random memory location in $\mathbf{A}_i$'s SS. Let y_i be the contents at w_i'. Let r_i be shares of W generated for each $\mathbf{A}_i'$.

1. D generates a keys pair (k_{pub}/k_{pri}) such that $\{k_{pub}, k_{priv}\} \in \mathcal{K}$.
2. D publishes k_{pub}.
3. D calculates shares s_i' using k_{priv} for each $\mathbf{A}_i'$.

$$s_i' = k_{priv} + \sum_{j=1}^{t-1} a_j x^j \ mod(p)$$

4. D also generates a random memory location w_i' for each $\mathbf{A}_i'$ and combines its contents y_i with s_i' to obtain $z_i' = s_i' + y_i$ (See Figure 14).
5. D calculates shares r_i for each $\mathbf{A}_i'$ using W.

$$r_i = W + \sum_{j=1}^{t-1} a_j x^j \ mod(p)$$

6. D signs each agent, $sig_{k_{priv}}(\mathbf{A}_i'||z_i'||r_i)$ with k_{priv} before destroying k_{priv}.
7. D attaches with each $\mathbf{A}_i'||z_i'||r_i||sig_{k_{pirv}}(\mathbf{A}_i'||z_i'||r_i)$ and transfers each $(\mathbf{A}_i'||z_i'||r_i||sig_{k_{priv}}(\mathbf{A}_i'||z_i||r_i))$ to H_i'.

Figure 8: *Agent initialization on remote hosts.*

4.4.1 Encrypting

The procedure for an agent to perform encryption on results gathered on remote hosts is the same as outlined in Section 4.3.2.

4.4.2 Decrypting

Decryption is performed only on the home host (User in Figure 10) regardless where the agent was generated. The decryption process is also similar to the procedure outlined Section 4.3.3. But because the value y_i in stored within $\mathbf{A}_i$'s SS, decryption can be performed only when $\mathbf{A}_i$ returns to its home host. First the value r_i is extracted from each $\mathbf{A}_i'$ to generate the combined random locations W through polynomial interpolation. Using each w_i' the values y_i is retrieved from $\mathbf{A}_i'$'s SS. Using each s_i', again through polynomial interpolation, k_{priv} can be regenerated.

4.5 Remarks

The distributing of the private-key amongst agents offers greater protection for the key. Not only will a malicious host have to capture at least t agents, it must also find the shares within each agent. For example, suppose a host has all of the agents and is able to find the values z_i, without the random value w_i, s_i is considerably harder to obtain. Independence and autonomy are key characteristic of an agent. Therefore, the ability to distribute agents on remote hosts have some obvious advantages. However, this also posses new challenges. How to hide a secret within an agent? Through the use of secret sharing, and information hiding within an agent's SS it was shown that information and private-keys can be hidden within a number of agents. The use of Shamir's threshold scheme also allows for inherent fault-tolerance because only t of n shares are required to regenerate the private key. Thus, the robustness of the algorithm allows regeneration of the private-key even if $n - t$ agents are destroyed, lost, or tempered with.

CHAPTER V

SECRET SHARING WITH RSA KEYS

This chapter will demonstrate how the protocol of Chapter 4 can be implemented with RSA [28] PKE algorithm. RSA keys can be used for encryption/decryption and for DS.

5.1 RSA Overview

Let $\mathbf{n} = \mathbf{pq}$ be the RSA modulus and $\mathbf{p}, \mathbf{q}$ are primes. Let a, b be the encryption and decryption keys respectively where:

$$ab \equiv 1\ mod(\phi(\mathbf{n})) \quad \text{and} \quad \phi(\mathbf{n}) = (\mathbf{p}-1)(\mathbf{q}-1)$$

The values $\mathbf{p}, \mathbf{q}, b$ are private and $\mathbf{n}, a$ are public. Given the message m encryption is defined as:

$$y = e_k(m) = m^a\ mod(\mathbf{n}),$$

decryption is defined as:

$$d_k(y) = y^b\ mod(\mathbf{n}) = m$$

signature is defined as:

$$sig_k(m) = m^b\ mod(\mathbf{n})$$

and verification of $sig_k(m)$ is defined as:

$$ver_k(m, sig_k(m)) = true \Leftrightarrow x \equiv y^a\ mod(\mathbf{n})$$

5.2 Agent Initialization

Let sig_b be the signature function using private value b.

1. D generates the primes $\mathbf{p}, \mathbf{q}$ and calculates the RSA modulus $\mathbf{n}$.
2. D calculates the private value b and the public value a, then destroys the $\mathbf{p}, \mathbf{q}$ and publishes the public values $\mathbf{n}, a$ in a accessible location.
3. Using the private value b, D generates s_i for each $\mathbf{A}_i$.
$$s_i = b + \sum_{j=1}^{t-1} a_j x^j \ mod(p)$$
4. D generates a random number w_i for each $\mathbf{A}_i$ and distributes $z_i = s_i + w_i$ to each $\mathbf{A}_i$.
5. D signs each agent, $sig_b(\mathbf{A}_i || z_i)$ with b before destroying b.
6. D attaches with each $\mathbf{A}_i || z_i || sig_b(\mathbf{A}_i || z_i)$ and transfers each $(\mathbf{A}_i || z_i || sig_b(\mathbf{A}_i || z_i))$ to H_i.

Figure 9: *Agent Initialization Using RSA Keys.*

5.3 RSA Encrypting

Let e_a be the encryption function using the public key a. As results $R_{i,j}$ are collected by $\mathbf{A}_i$ on host $H_{i,j}$, $j \geq 1$, before $\mathbf{A}_i$ moves to host H_{j+1} (Again note that movement of $\mathbf{A}_i$ from host $H_{i,j}$ to $H_{i,j+1}$ is for notational purposes only. Agent migration does not require an itinerary.), $\mathbf{A}_i$ retrieves a from the public accessible location and encrypts $R_{i,j}$ with a to obtain:

$$e_a(R_{i,j}) = (R_{i,j})^a \ mod(\mathbf{n})$$

5.4 RSA Decrypting

Let d_b be the decryption function using key b. Let ver_a be verification function for sig_b. Decryption requires at least t of n agents to return with their shares. For

$1 \leq i \leq t$, upon return of each $\mathbf{A}_i$, D verifies each $\mathbf{A}_i$, if

$$ver_a\big((\mathbf{A}_i||z_i),\ sig_b(\mathbf{A}_i||z_i)\big) = true$$

D retrieves the value w_i from the secure location and extracts each $s_i = z_i - w_i$. If

$$|\{i : ver_a\big((\mathbf{A}_i||z_i),\ sig_b(\mathbf{A}_i||z_i)\big) = true\}| \geq t$$

though polynomial interpolation using each s_i, D regenerates the private key b. D can now use d_b to decrypt:

$$d_b\big(e_a(R_{i,j})\big) = \big(e_a(R_{i,j})\big)^b\ mod(\mathbf{n}) = R_{i,j}$$

CHAPTER VI

AGENT TRACKING

6.1 *Agent Collision*

A host that contain high concentration of relevant information will eventually attract multiple agents from the same family[1] . For example, a family of agents are created to find a television on behalf of its creator. It is conceivable that more than one agent from that family will eventually visit the same major electronics retailer. Agent collision occurs when a host is visited more than once by agents from the same family either individually or concurrently as depicted in Figure 10.

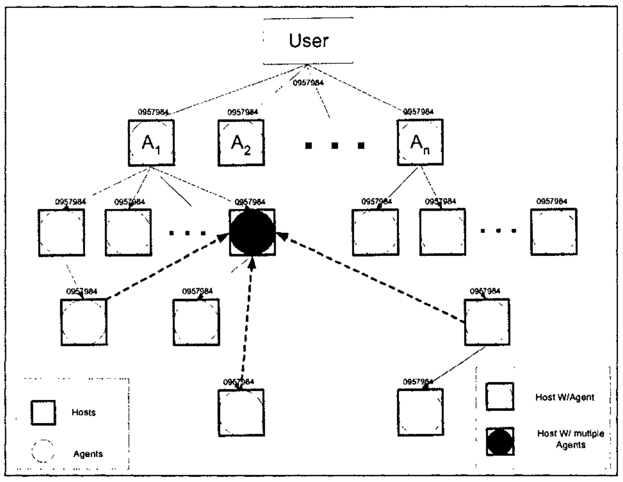


Figure 10: *Family of agents with same unique ID showing agent collision.*

[1]All agents created by a user to perform the same task. Agents in Figure 10 are from the same family.

6.2 Cookies

Cookies are small files an agent stores on each host it visits. All cookies generated by agents of the same family share the same unique $\mathcal{ID}$ (see Figure 10). When an agent arrives, it first checks for a file on the host with the same $\mathcal{ID}$ as the agent. The existence of a file with the same $\mathcal{ID}$ on a host means the host has already been visited by a member of its family. Otherwise, the agent creates the file and proceeds with execution. To avoid overloading agent storage environments on remote hosts, cookies have a limited life span. Once expired, the cookie can be deleted by the remote host.

6.2.1 $\mathcal{ID}$ Generation

*6.2.1.1 **

$\mathcal{ID}$ Generation There are two requirements for the generation of $\mathcal{ID}$: the $\mathcal{ID}$ should be as unique as possible and the generation of $\mathcal{ID}$ must use a well-established publicly available one-way hash function (e.g. The MD5 [29] algorithm).

The generation of an unique $\mathcal{ID}$ requires that a message m be the input of the hash function h, $\mathcal{ID} = h(m)$. Any random string can be used, but to ensure high probability of uniqueness, the suggestion is that m contain information such as user's IP address along with purpose of the group of agents and some *salt* (random numbers). For example, $m = (211.112\ldots||$"return;YVR,YYZ..."$||salt)$. Once the $\mathcal{ID}$ has been generated, m must be stored in a secure accessible location to be produced when requested by TTP for cookie authentication (see section on Cookie Authentication).

6.2.2 Cookie Security

Possible attacks against cookies include the following: 1. Hosts use cookies as a tool for *denial of service* attacks. Because of the uniqueness of the $\mathcal{ID}$, it would be unlikely that a host can generate the unique $\mathcal{ID}$ prior to an agent's arrival. 2. Hosts can hide or remove cookies before they expire. This ability is also of little concern because a legitimate cookie on a host means the host has already been visited by a member of the agent family. 3. Hosts once visited by an agent, creates their own agents with the same $\mathcal{ID}$, and distributes the agents onto competitors hosts to plant cookies. This scenario is possible, particularly for e-commerce applications where there is a potential for financial gain. A possible solution to this type of denial of service attack is the use of a TTP to authenticate cookies.

6.2.3 Cookie Authentication

Cookie authentication can be initiated by a host, or an agent that suspects that a cookie on the current host is fake. The authentication process is performed by the host with the suspect cookie. Before the cookie authentication process is described, a couple of assumptions will first be made. 1. All agents, including ones generated on remote hosts know the location of their originators. In Figure 10, all agents in the diagram belong to User; therefore User is the originator. 2. A host also knows the location (e.g. IP) of an agent's originator.

The authentication process requires the participation of a TTP; a TTP could be any TH mentioned previously in this paper, with a signed digital certificate. The certificate is used to authenticate the TTP to the User and the Host (Figure 11). Connections between the Host and TTP, and the User and TTP are secure (e.g. Using SSL [13] with the TTP's certificate). The authentication protocol requires the following routines.

Req_Auth$(\mathcal{ID}, UserIP)$**.** Authentication request sent by the suspecting host to the TTP to verify the $\mathcal{ID}$ of the cookie.

$\mathcal{ID}$**.** The unique number that identifies a family of agents.

$UserIP$**.** The location of the User that created the agent.

Req_Msg$(\mathcal{ID})$**.** Message request sent by the TTP to the host located at $UserIP$ (User in Figure 11).

$Msg(m)$. Response from the User located at $UserIP$ to the TTP.

m. Input of h that generated $\mathcal{ID}$.

$h(m)$. Hash value on the message m.

$Auth(true/false)$. Reply from TTP to the suspecting host.

$true/false$. Whether $h(m) = \mathcal{ID}$ or $h(m) \neq \mathcal{ID}$ respectively.

The host with the cookie can request a TTP to verify the cookie on its behalf. The host with its request sends the unique $\mathcal{ID}$ of the cookie and the location of User $UserID$ to TTP. The TTP then requests from User the message m which generated the unique $\mathcal{ID}$. The TTP uses the hash function h to generate $h(m) = \mathcal{ID}$. If $h(m) = \mathcal{ID}$, TTP sends a true reply to the Host. (see Figure 11)

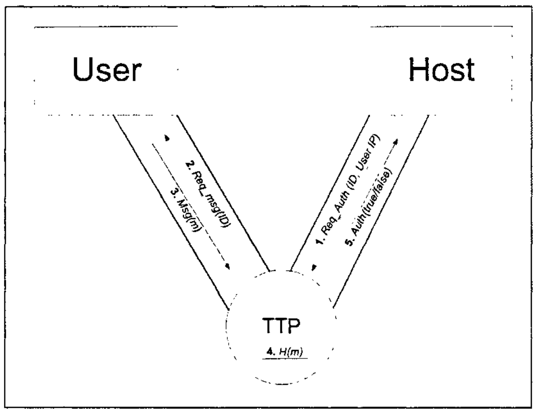


Figure 11: *Cookie Authentication*

6.3 Remarks

In Section 6.2.3, two assumptions were made. The first assumption was that every agent knows the location of its originator. This assumption was made because in the model described above, every agent must return to their respective originators.

This assumption is also important for the cookie authentication process. If a host is to check authenticity of a cookie, it must know the location of originator (User in Figures 10,11) of the agent. The second assumption was that a remote host knows the originator of an agent it executes. This fact is especially important because to authenticate a cookie, the current host (host with the cookie) sends the TTP the message $Req\_Auth(\mathcal{ID}, UserIP)$ to User. This message contains $UeserIP$, which should be location of User. However if the current host is not certain where an agent comes from, then there is possibility that although the final response from TTP is $Msg(m) = true$, there is no certainty that the cookie is authentic.

For example, given two hosts X, Y and host X and Y are competitors. Let host X be a malicious host. Suppose agent $\mathbf{A}_1$ visits host X first and $\mathbf{A}_1$'s originator is O. Now host X knows both the purpose of $\mathbf{A}_1$ and it's unique $\mathcal{ID}$. Host X then generates its own agent A_1' and sends it to host Y and A_1' plants a cookie on host Y with the same $\mathcal{ID}$ as $\mathbf{A}_1$. Suppose $\mathbf{A}_1'$ arrives at host Y before $\mathbf{A}_1$. When $\mathbf{A}_1$ arrives two things could happen. First, $\mathbf{A}_1$ could simply move on thinking that host Y has already been visited. Second, $\mathbf{A}_1$ suspects that the cookie is fake and requests the current host to authenticate the cookie. If the current host doesn't know the true originator of the cookie is from host X and believe the cookie is from O. When host Y authenticates the cookie, the request will be sent to TTP which then sends the request to O. After the authentication process is complete, the message from TTP will show that the cookie is authentic when in fact its fake.

CHAPTER VII

CONCLUSION

This thesis explored the mobile agent paradigm from the security perspective. Some key benefits of a mobile agent was discussed as well as how they can contribute to the overall network computing community. Various applications which can benefit from an agent's flexibility, mobility and scalability was also investigated and their security requirement highlighted.

Mobile agent computing as with most emerging technology has many unresolved issues. The most significant of which are in the area of security. Technologies such as programming languages and network infrastructures are all in place for deployment of agents. However, until the establishment of a well defined set specifications which the network community as a whole is willing to adopt, mobile agent development will still remain ad hoc in nature.

The contribution of this research is the use of secret sharing schemes as a tool for security. Using secret sharing scheme, it was shown that an agent with a public-key can perform encryption on remote hosts while protecting the private-key for decryption though secret sharing. All the tools required for implementation such as secret sharing schemes and PKE are available and proven secure. The example showing implementation using the RSA PKE demonstrates that the protocol is practical.

Agent tracking using a cookie concept borrowed from the web browser technology showed how to avoid agent collision. The security issues with using such technology was also discussed and solutions proposed.

In future work, the hope is to explore other uses for secret sharing in the mobile agent paradigm. Another possible branch of this research is the use of CEF for the generation of shares on remote hosts because as mentioned earlier, CEF has been shown to work with polynomial functions[4].

APPENDIX A

SHAMIR'S THRESHOLD SCHEME

A.1 Initialization and Share Distribution

Initialization and Share Distribution

1. D chooses n distinct non-zero elements of $\mathbb{Z}_p$ denoted by x_i, $1 \leq i \leq n$ and $p \geq w+1$. For $1 \leq i \leq n$, D gives each P_i the value x_i.
2. To share the key $k \in \mathbb{Z}_p$, D secretly chooses at random $t-1$ element of $\mathbb{Z}_p, a_1, a_2, \ldots, a_{t-1}$.
3. For $1 \leq i \leq n$, D calculates $y_i = a(x_i)$, where
$$a(x) = k + \sum_{j=1}^{t-1} a_j x^j \; mod(p)$$
4. For $1 \leq i \leq n$, D gives the share y_i to P_i.

Figure 12: *Shamir's (t,n)-threshold scheme in $\mathbb{Z}_p$. Initialization and Share Distribution.*[34]

A.1.1 Initialization and Share Distribution Example

This is a small example of to illustrate Shamir's Threshold Scheme [34].

Let $t = 3$, and $p = 17$ and $n = 5$ and $k = a_0 = 13$

Given the polynomial:

$$a(x) = a_0 + a_1 x + a_2 x^2 \; mod(p)$$

Step 2 Figure 12) Let $a_1 = 10$ and $a_2 = 2$

Step 3 Figure 12)

$$
\begin{aligned}
a(1) &= 13 + 10 + 2 & mod(17) \\
a(1) &= 8 & mod(17) \\
a(2) &= 13 + 10(2) + 2(2)^2 & mod(17) \\
a(2) &= 7 & mod(17) \\
a(3) &= 13 + 10(3) + 2(3)^2 & mod(17) \\
a(3) &= 10 & mod(17) \\
a(4) &= 13 + 10(4) + 2(4)^2 & mod(17) \\
a(4) &= 0 & mod(17) \\
a(5) &= 13 + 10(5) + 2(5)^2 & mod(17) \\
a(5) &= 11 & mod(17)
\end{aligned}
$$

Step 4 Figure 12) Let $i = \{1, 3, 5\}$, $P_1 = 8$, $P_2 = 10$, $P_5 = 11$.

A.2 Key Reconstruction

Key Reconstruction
The regeneration of k is achieved though polynomial interpolation. Suppose $P_{i_1}, \ldots, P_{i_t}$ want to regenerate the k, they know that

$$y_{i_j} = a(x_i)$$

and $1 \leq i \leq t$ where $a(x) \in \mathbb{Z}_p[x]$ is the secret polynomial chosen by D. Since $a(x)$ has degree at most $t-1$, $a(x)$ can be written as

$$a(x) = a_0 + a_1 x + \ldots + a_{t-1} x^{t-1}$$

where a_0 is the key and the coefficient $a_0 \ldots a_{t-1}$ are unknown elements of $\mathbb{Z}_p$. Since $y_i = a(x), 1 \leq i \leq t$, M can obtain t linear equations with t unknowns $a_0, \ldots, a_{t-1}$ with all arithmetic done in $\mathbb{Z}_p$. If the equations are linearly independent, there will be a unique solution and a_0 is the key.

Figure 13: *Shamir's (t,n)-threshold scheme in $\mathbb{Z}_p$. Key Reconstruction.*[34]

A.2.1 Key Reconstruction Example

Let $t = 3$, and $p = 17$ and $n = 5$ and $x_i = i$ and $M = \{P_1, P_2, P_3\}$ with shares $8, 10, 11$ respectively(See A.1.1).

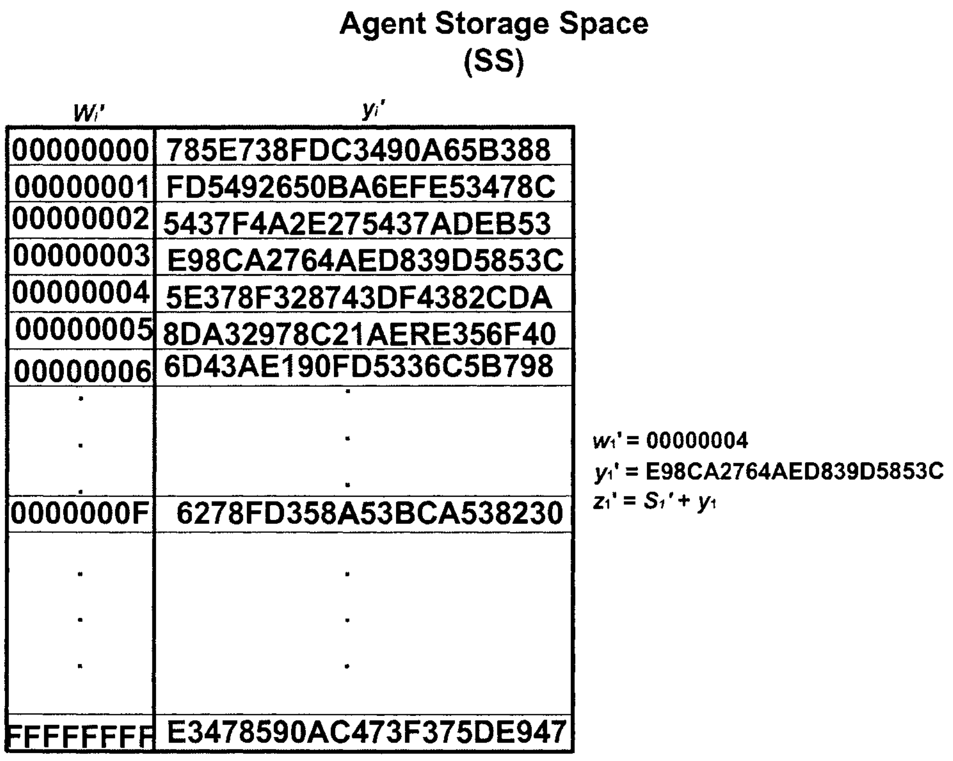
$$\begin{aligned} a_0 + a_1 + a_2 &= 8 \quad \mathrm{mod}(17) \\ a_0 + 3a_1 + 9a_2 &= 10 \quad \mathrm{mod}(17) \\ a_0 + 5a_+ 8a_2 &= 11 \quad \mathrm{mod}(17) \end{aligned}$$

Solving the system of linear equations we get $a_0 = 13, a_1 = 10$ and $a_2 = 2$. Since $k = a_0 = 13$, therefore the key has been successfully regenerated.

Given the polynomial:

APPENDIX B

SHARE HIDING WITHIN AGENT'S STORAGE SPACE


Agent Storage Space (SS)

w_i'	y_i'
00000000	785E738FDC3490A65B388
00000001	FD5492650BA6EFE53478C
00000002	5437F4A2E275437ADEB53
00000003	E98CA2764AED839D5853C
00000004	5E378F328743DF4382CDA
00000005	8DA32978C21AERE356F40
00000006	6D43AE190FD5336C5B798
. . .	. . .
0000000F	6278FD358A53BCA538230
. . .	. . .
FFFFFFFF	E3478590AC473F375DE947

$w_1' = 00000004$
$y_1' = \text{E98CA2764AED839D5853C}$
$z_1' = S_1' + y_1$

Figure 14: *Share hiding within the Storage Space (SS) of an agent.*

REFERENCES

[1] Algesheimer Joy, Cachin Christian, C. J. K. G. Cryptographic security for mobile code. Tech. Rep. RZ 3302 (# 93348), 2000. `http://citeseer.nj.nec.com/algesheimer00cryptographic.html`.

[2] Bierman, E., and Cloete, E. Classification of malicious host threats in mobile agent computing. In *Proceedings of the 2002 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology* (2002), South African Institute for Computer Scientists and Information Technologists, pp. 141–148.

[3] Blakley, G. Safeguarding cryptographic keys. In *Proceedings of the AFIPS 1979 National Computer Conference* (1979), vol. 48, pp. 313–317.

[4] Cachin, C., Camenisch, J., Kilian, J., and Muller, J. One-round secure computation and secure autonomous mobile agents. In *Automata, Languages and Programming* (2000), pp. 512–523. `http://citeseer.nj.nec.com/cachin00oneround.html`.

[5] Chess, D., Grosof, B., Harrison, C., Levine, D., Parris, C., and Tsudik, G. Itinerant agents for mobile computing. In *Readings in Agents*, M. N. Huhns and M. P. Singh, Eds. Morgan Kaufmann, San Francisco, CA, USA, 1997, pp. 267–282.

[6] Chess, D., Harrison, C., and Kershenbaum, A. Mobile Agents: Are They a Good Idea? Tech. Rep. RC 19887 (December 21, 1994 - Declassified March 16, 1995), Yorktown Heights, New York, 1994. `http://citeseer.nj.nec.com/chess95mobile.html`.

[7] D. Johansen, R. van Renesse, F. S. An introduction to the tacoma distributed system version 1.0. `http://www.tacoma.cs.uit.no/papers/tacoma.SOSP95.ps`, 1995.

[8] Farmer, W. M., Guttman, J. D., and Swarup, V. Security for mobile agents: Authentication and state appraisal. In *Proceedings of the Fourth European Symposium on Research in Computer Security* (Rome, Italy, 1996), pp. 118–130. `http://citeseer.nj.nec.com/farmer96security.html`.

[9] Fuggetta, A., Picco, G. P., and Vigna, G. Understanding Code Mobility. *IEEE Transactions on Software Engineering 24*, 5 (1998), 342–361. `http://citeseer.nj.nec.com/fuggetta98understanding.html`.

[10] GRAY, R. S. Agent Tcl: A flexible and secure mobile-agent system. In *Fourth Annual Tcl/Tk Workshop (TCL 96)* (Monterey, CA, 1996), M. Diekhans and M. Roseman, Eds., pp. 9–23. `http://citeseer.nj.nec.com/gray97agent.html`.

[11] HOHL, F. A model of attacks of malicious hosts against mobile agents. In *In Proceedings of the ECOOP Workshop on Distributed Object Security and Jth Workshop on Mobile Object Systems: Secure Internet Mobile Computations* (1998a), pp. 105–120.

[12] IBM. Aglets workbench. `http://www.trl.ibm.com/aglets/`.

[13] IETF. Transport layer security. `http://www.ietf.org/internet-drafts/draft-ietf-tls-rfc2246-bis-03.txt`.

[14] JANSEN, W. Countermeasures for mobile agent security, 2000. `http://citeseer.nj.nec.com/jansen00countermeasures.html`.

[15] JANSEN, W., AND KARYGIANNIS, T. Nist special publication 800-19 - mobile agent security, 2000. `http://citeseer.nj.nec.com/jansen00nist.html`.

[16] J.S. FRITZINGER, M. M. Java$^{\text{TM}}$security. `http://java.sun.com/security/whitepaper.ps`.

[17] KOTZANIKOLAOU PANAYIOTIS, K. G., AND VASSILIOS, C. Mobile agents for secure electronic transactions, 1999. `http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole/security/kbc00%.pdf`.

[18] LANGE, D. B., AND OSHIMA, M. Seven good reasons for mobile agents. *Communications of the ACM 42*, 3 (1999), 88–89.

[19] LOUREIRO, S. Mobile code protection, 2001. `http://citeseer.nj.nec.com/loureiro01mobile.html`.

[20] LOUREIRO, S., AND MOLVA, R. Mobile code protection with smartcards. `http://citeseer.nj.nec.com/408410.html`.

[21] M. STRAER, J. BAUMANN, F. H. Mole - a java based mobile agent system. Special Issues in Object-Oriented Programming, Workshop Reader ECOOP'96, dpunkt.verlag, pp. 327–334. `http://www.informatik.uni-stuttgart.de/ipvr/vs/Publications/1996-strasser-0%1.ps.gz`.

[22] MEADOWS, C. Detecting attacks on mobile agents, 1997. `http://citeseer.nj.nec.com/meadows97detecting.html`.

[23] MIAO, C., AND WEI, R. Secret sharing for mobile agent cryptography. In *In Proceedings of the 1st International Conference on Communications Networks & Services Research (CNSR), 2003 (To Appear)* (Moncton, Canada).

[24] Milojicic, D. Mobile agent applications. IEEE Concurrency, 1999, pp. 80–90. http://www.computer.org/concurrency/pd1999/pdf/p3080.pdf.

[25] Ordille, J. J. When agents roam, who can you trust? In *First Conference on Emerging Technologies and Applications in Communications (etaCOM)* (Portland, OR, 1996).

[26] Ousterhout, J. K., Levy, J. Y., and Welch, B. B. The Safe-Tcl security model. *Lecture Notes in Computer Science 1419* (1998), 217–?? http://citeseer.nj.nec.com/ousterhout97safetcl.html.

[27] P. Kotzanikolaou, M. B., and Chrissikopoulos, V. Secure transactions with mobile agents in hostile environments. In *Information Security and Privacy* (2000), Proceedings of the 5th Australasian Conference ACISP, number 1841 in Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 289–297. http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole/security/kbc00%.pdf.

[28] Rivest, R. L., Shamir, A., and Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21*, 2 (1978), 120–126.

[29] Rivset, R. The md5 message digest algorithm, rfc 1321. 1992. http://theory.lcs.mit.edu/%7Erivest/Rivest-MD5.txt.

[30] Roth, V. Secure recording of itineraries through co-operating agents. In *ECOOP Workshops* (1998), pp. 297–298. http://citeseer.nj.nec.com/roth02secure.html.

[31] Roth, V., Jalali, M., Hartman, R., and Roland, C. An application of mobile agents as personal assistants in electronic commerce. In *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000)* (Manchester, UK, 2000), J. Bradshaw and G. Arnold, Eds., The Practical Application Company Ltd., pp. 121–132. citeseer.nj.nec.com/roth00application.html.

[32] Sander, T., and Tschudin, C. F. Protecting mobile agents against malicious hosts. *Lecture Notes in Computer Science 1419* (1998), 44–. http://citeseer.nj.nec.com/sander98protecting.html.

[33] Shamir, A. How to share a secret. *Communications of the ACM 22*, 11 (1979), 612–613.

[34] Stinson, D. *Cryptography Theory and Practice.* Chapman and Hall, isbn: 0849385210, Chapman and Hall, 1995, pp. 327–329.

[35] Vigna, J. Cryptographic traces for mobile agents, 1998. http://citeseer.nj.nec.com/vagina03cryptographic.html.

[36] White, J. E. *Mobile Agents.* MIT Press, isbn 0-262-52234-9, MIT Press, 1997, pp. 437–472.

[37] Wilhelm, U. Cryptographically protected objects. http://citeseer.nj.nec.com/74363.html, 1997.

[38] Wilhelm, U. G., Staamann, S., and Buttyán, L. Introducing trusted third parties to the mobile agent paradigm. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, J. Vitek and C. Jensen, Eds., vol. 1603. Springer-Verlag, New York, NY, USA, 1999, pp. 471–491. http://citeseer.nj.nec.com/wilhelm99introducing.html.

[39] Yee, B. S. A sanctuary for mobile agents. In *Secure Internet Programming* (1999), pp. 261–273. http://citeseer.nj.nec.com/article/yee97sanctuary.html.

[40] Young Adam, Y. M. Sliding encryption: A cryptographic tool for mobile agents. In *Fast Software Encryption. Lecture Notes in Computer Science, no. 1267*, E. Biham, Ed. Springer-Verlag, Berlin Germany, 1997, pp. 230–241.

[41] Zakon, R. H. Hobbes internet timeline. http://www.zakon.org/robert/internet/timeline/.