INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa · University of Ottawa

Development of Security Features for the FIPA Architecture

by

Min Zhang

A thesis submitted to the
School of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

M. A. Sc.
in
Electrical and Computer Engineering

Ottawa-Carleton Institute of Electrical and Computer Engineering

School of Information Technology Engineering
Faculty of Engineering
University of Ottawa

Ottawa, Ontario, Canada

May, 2001

I hereby declare that I am the sole author of this thesis.

I authorized the University of Ottawa to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Zhang Min

I further authorize the University of Ottawa to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutes or individuals for the purpose of scholarly research.

Zhang Min

The University of Ottawa requires the signature of all persons using or photocopying this thesis. Please sign below, and give address and date.

ABSTRACT

In a highly heterogeneous computing environment, interoperability among different agent platforms is made possible by the Foundation for Intelligent Physical Agent (FIPA) specification. As long as they are all FIPA-compliant, agents of different systems or providers can communicate and interact directly using Agent Communication Language (ACL). However, neither the FIPA specification nor most of its implementations such as FIPA-OS (FIPA Open Source) fully address potential security threats to agents and agent platforms. In this thesis, we discuss the security concerns in FIPA and propose a two-layer architecture to add security features: a basic FIPA-OS agent platform as the management and communication infrastructure, and a security layer as the security extension. This architecture provides agents with two security-related services: a secure communication service and a secure execution environment service. The secure communication service prevents any eavesdropping or interference from the outside network. The secure execution environment service protects server resources and agent services from unauthorized access. The architecture's design and components are also described in terms of the two security services. In addition, this thesis describes a trust model of the two-layer architecture, which consists of a set of assumptions concerning the relationships of all entities.

ACKNOWLEDGEMENT

Firstly I would like to express my special thanks to my supervisor Professor Ahmed Karmouch. His valuable guidance and advice, academic support and encouragement were very helpful to me and have made possible the progression of my research work and the completion of this thesis.

I would also like to thank Mr. Roger Impey from Nation Research Council, whose valuable feedback and technical suggestions help me through all the stages of my research work.

Thanks to all the colleagues of the Multimedia Information and Mobile Agent Research Lab, who have given me continuous assistance to transcend many of the obstacles that I faced in the journey of the research.

Finally, I should express my boundless gratitude to my family: my husband and my parents, whose consistent encouragement and support have helped me through all my studies and research work.

ACRONYMS

ACC:	Agent Communication Channel
ACL:	Agent Communication Language
AMS:	Agent Management Server
APSM:	Agent Platform Security Manager
CA:	Certificate Authority
CGC:	Credential Granting Center
DF:	Directory Facility
FIPA:	Foundation for Intelligent Physical Agent
FIPA-OS:	Foundation for Intelligent Physical Agent - Open Source
HAP:	Home Agent Platform
IIOP:	Internet Inter-ORB Protocol
MASIF:	Mobile Agent System Interoperability Facility
PS:	Policy Server
SACC:	Secure Agent Communication Channel

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Network computing – Mobile Agent Technology

By definition, a software agent is a computer program that acts autonomously on behalf of a person or an organization [1] [11]. An agent has its own thread of execution and performs tasks on its own initiative. Usually, an agent executes in a certain environment called an agent platform or an agent server, which helps to finish its task. Compared with the traditional Client/Server computation model, agent technology is a new computation model that is more applicable in distributed processing field, and this new model shifts current computing environment to be more highly distributed and heterogeneous.

A mobile agent is a type of agent that can migrate from one host to another [2]. In the paradigm of mobile agent technology, each mobile agent is an independent computation unit on behalf its owner. Rather than communicating with other software agents remotely, a mobile agent migrate to the remote host, and interact with them locally by the peer-to-peer communication. The environment where mobile agents execute is called a mobile agent platform or a mobile agent environment. No matter the difference in name, they

both are responsible for the life cycle control of mobile agents, including their communications or interactions and of course the mobility management.

Advantages of the mobile agent technology are fully discussed in many papers. Two major advantages contribute greatly to the widely acceptance of mobile agent technology: (1) Reduction of network traffic. Usually the cost (e.g. time) in the transmission of an agent's code is less than that of the huge data back and forth. (2) Enabling system to be more asynchronous and autonomic. In some cases, a system objective can be divided into smaller parallel tasks. Mobile agents are created and dispatched to work asynchronously with these tasks as their initiative goals. These advantages of mobile agent technology result in the improvements in system efficiency and flexibility.

Therefore the mobile agent technology is currently receiving extraordinary attention and is being proposed in a variety of application fields, such as electronic commerce [3], network management [4] and distributed information retrieval [5].

1.2 Security Threats in Network computing and Agent Technology

In contrast to its great advantages, some difficulties and problems exist in agent technology as well. Among them, the threat to security is one of the important challenges preventing the wider acceptance of agent technology. Especially when an agent is able to

execute on a remote unknown platform rather than its safe home platform, and in a heterogeneous and open computing environment like the Internet.

In the context of an agent system, a malicious entity may launch attacks to others in both passive and active ways. The typical means of attack include:

(1) Breach of privacy. This means the theft of an entity's content or an illegal or undesired access.
(2) Damage. This indicates the destruction or subversion of an entity.
(3) Masquerading. This means that an unauthorized entity claims the identity of others.
(4) Denial of Service. This means partially or completely impeding an entity's services by consuming an excessive amount of the entity's computing resource.

Figure 1.2.1 shows the attack model of an agent system. In this model, the potential security risks are classified into four categories: agent against agent platform, agent platform against agent, agent against agent and outside network against mobile agent.

An agent platform is vulnerable to attacks from malicious agents, especially mobile ones, unless countermeasures are taken. In the threats of agent against agent platform, an agent may try to destroy the server host or gain unauthorized access to the agent platform. The attacks launched by a malicious agent may be passive, such as the monitoring of communications, the pilfering of sensitive information of the platform or server. Attacks may also be active, such as the damage or destruction of host resources by deletion,

modification or addition, unauthorized access, or by nuisance attacks such as denial of service and other types of harassment.

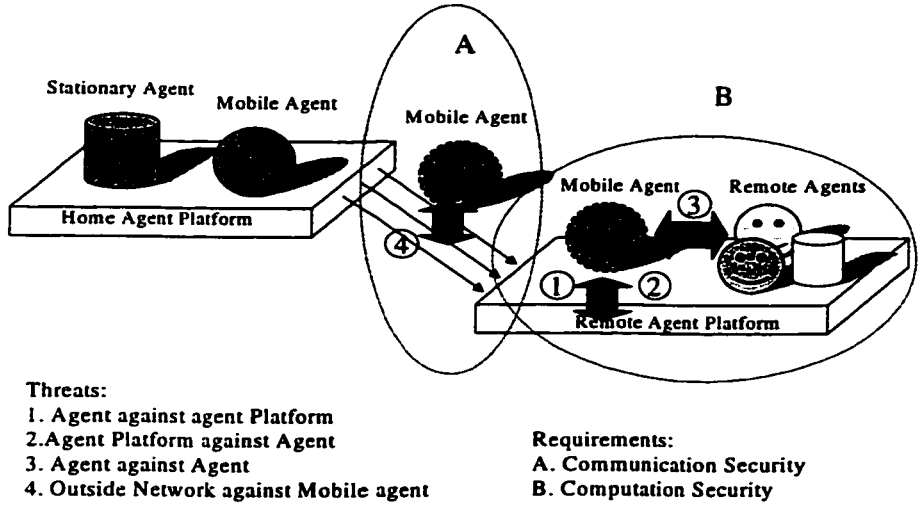

Figure 1.2.1 Attack Model of Agent System

The code and data of an agent (especially a foreign one) are also exposed to the risk of tampering when the agent is moving through the network or executing in a remote host. In the latter case, the attacks may come from the current platform. If so, preventing the attacks is more difficult because the agent is totally exposed to the agent platform, making interception and tampering easy. For example, the server may try to extract sensitive information (such as credit card numbers in E-Commerce) out of the agent without the agent's acknowledgment or detection. As Chess argued in [6], it is almost impossible to prevent these attacks without the presence of tamper-proof hardware.

Besides the threats from agent platform, The attacks to an agent may also come from neighboring agents when it arrives at the destination host. Some agents may exploit security weaknesses of other agents or launch attacks against other agents to hinder their performance or take their services forcibly.

The last category of threats is that of the outside network against the mobile agent. Mobile agents are vulnerable to attacks when they are migrating across the network or communicating with its home site. Passive attacks include eavesdropping, traffic analysis and breach of privacy; active attacks include message modification and forging, masquerading, and so on.

Therefore, we can see that both agents and agent platforms may be the source of attacks, not to mention those coming from malicious, or simply curious third parties. The need for protection is clear.

1.3 Objective: providing generic FIPA-compliant security features

In the initial research of agent technology, no specification or definition can be followed among researchers, therefore a diversity of agent platforms or agent servers were defined. Each of them has its own platform-specific service ontology, encoding mechanism and communication protocol, e.g. Agent Tcl [7], Grasshopper [8], Concodia [9], and Voyager

[10]. Though well designed, most of these systems are not compatible. And the lack of compatibility has limited the growth of agent and mobile agent technology.

In 1997, this situation changed significantly with the publication of two standards, Mobile Agent System Interoperability Facility (MASIF) [11] and Foundation for Intelligent Physical Agent (FIPA) [12]. MASIF is a basic set of interfaces rather than a specific agent technology. The interfaces can be used by developers to make complex systems with a high degree of interoperability. Unlike MASIF, FIPA is an abstract architecture that can be shared by different platforms. As long as they are FIPA-compliant, agents of different systems or providers can communicate and interact with each other directly by Agent Communication Language (ACL). All in all, these standards enable the interoperability of different agent platforms and thereby expedite the development of agent based applications.

In FIPA, however, potential security threats related to agents and agent platforms are not fully addressed in either the standard or most of its implementations. For example, there are now several agent platforms that declare their support of the FIPA agent standard such as FIPA-OS [13], JADE [14], Grasshopper [8], and ZEUS [15]. They all support agent's interactions in format of the ACL message, which is defined in the FIPA specification. But as to the security risks in these interactions, none of them declare their support of the security architecture in FIPA1998 standard. Also the security architecture in FIPA standard is still evolving. As a result, the previous security architecture has been declared obsolete and no completed picture for agent security architecture can be derived

from the latest version FIPA2000. The reasons are manifold and will be discussed in later chapter 2.

Therefore the principal motive behind our work is to discuss potential threats and the security weakness in a multi agent system, particularly in the context of FIPA specification — a message oriented agent system. Based on our discussion of the attack model, we propose a framework that provides security services to protect the cooperation and interactions of different entities in a multi agent system.

In this thesis, we are expecting to add generic FIPA-compliant security features to the FIPA architecture that can be implemented differently in a diversity of agent platforms. Therefore agents built on these agent platforms are able to communicate and interact with each other safely. Similar to the goal of FIPA standard, our proposal seeks to be an abstract architectural solution. That is, it focuses on the functionality of the components in the architecture and independent from specific implementation techniques. In addition, we will provide one implementation of this architecture on a Java-based FIPA agent platform called FIPA-OS (FIPA Open Source), as well as the experiment results.

1.4 Main Contributions and Outline

In this thesis, we propose a two–layer framework based on FIPA-OS agent platform, which is employed as the agent management and communication infrastructure. A

security layer is designed as a extension layer to add security features to the FIPA-OS. This security layer provides security-related services to agents on top of it, namely the secure communication service and the secure execution environment service. The secure communication service eliminates the threats of detection, modification and masquerading from the open network, whereas the secure execution environment service diminishes the threat of mobile agent using unauthorized resources and services. Three major features are adopted in our system:

Authentication:

By definition, authentication is the process of verifying a claim of identity. Authentication is a basic security mechanism, helping to prevent the threat of masquerading and unrestrained access. Two types of authentication are used in our design: platform authentication and agent authentication. Platform authentication is done by the mutual authentication protocol proposed in later chapters. This establishes the authenticity of two communicating agent platforms by verifying both platforms' claims of identity. Agent authentication is unidirectional; it is performed by agent platforms to verify the identities of agents and their owners and to ensure the control of proper access rights.

Secure Communication:

A secure communication channel between two FIPA agent platforms can prevent most common third party threats. The establishment of secure channels enables secure communication between platforms. Each agent platform therefore uses authentication,

encryption, integrity protection and replay detection to thwart network communication attacks such as masquerading, eavesdropping and tampering. In FIPA, Agent Communication Language (ACL) is a standard communication language that defines the encoding semantics and the pragmatics of the messages. By inserting security parameters into an ACL message, an agent is able to activate the secure communication service of the current platform. Before it leaves the platform, the content of the message will be replaced by a set of security parameters containing the encrypted message.

Resource Monitoring:

The resource monitoring mechanism protect some platform resources and agent services from unrestrained access; and ensure that only authorized agents are given access to these resources and services. Resource monitoring is achieved by the authentication and authorization mechanism in an agent platform which provides a safe binding between the visiting agent and the local environment. It enables the visiting agent to apply for the resources it needs and enables the platform to control the access rights in format of security policies. But at the same time, it ensures that the agent cannot breach the system's security by accessing resources or services it is not authorized to use. The agent is eligible to have access or not depending on the authentication of its identity.

The rest of this thesis is organized as follows: in chapter 2, we give a brief introduction of FIPA specification and the analysis of the security threats and weakness in FIPA. Furthermore, relevant cryptographic technologies are introduced shortly as the background knowledge of our design and implementation. Chapter 3 describes the two-

layer security architecture and the main components in this architecture in terms of two security services it provides. Then the secure communication service and the secure execution environment service are described in detail respectively. The trust model is also presented in this chapter. Chapter 4 presents the implementation algorithms and details, as well as the experiment results and the evaluation of these results. Finally we give the summary and suggestions for future work in chapter 5.

CHAPTER 2

TECHNOLOGY BACKGROUND

In this chapter, we give a brief introduction to the technology background of our system, which include two major parts: the FIPA standards and the cryptography technology. In the first part, we have a quick review of the agent platform defined in FIPA specifications and especially the security architecture described in FIPA. We also give a brief introduction to the FIPA-OS agent platform, on which our implementation is based. In the part of cryptography technology, we firstly give the definition and basic requirements of a cryptosystem, then followed by the introduction of cryptographic algorithms, one-way hash functions, the digital signature and authentication protocols, which are all used in our system in order to satisfy the requirements of a cryptosystem.

2.1 FIPA Specification

Foundation for Intelligent Physical Agent (FIPA) is an international non-profit association of companies and organizations that share the effort to produce specifications of generic agent technology. The FIPA specification published by them is a set of standards for agent-based systems, which aims at the integration and high degree of interoperability of different types of agent platforms. As an agent standard, FIPA does not attempt to define the internal architecture of agent platforms or their implementation

techniques. Instead, FIPA focuses on the necessary interfaces that support interoperability between agent platforms. This means that FIPA describes only an abstract architecture, its necessary functional elements as well as their relationships. A broad set of possible concrete architectures can be derived from this abstract architecture, and agent platforms under these concrete architectures are able to communicate since they share the same abstract architecture. Therefore, agents of different system or providers, as far as they are all FIPA-compliant, are able to communicate and interact with each other.

Towards a complete set of standards, FIPA provides specifications that cover the entire scenarios in an agent's lifecycle, including the agent identification, locating each other, exchanging information, interacting with user and migrating from one platform to another. The first FIPA specification is issued in October 1997, which covers basic requirements such as the agent management, the agent communication and the agent/software integration. In addition to these normative parts, FIPA97 provides some informative parts such as personal assistant and audio-visual entertainment & broadcasting. FIPA98 updates the previous core specifications in some aspects (e.g. the agent management part), and add some new parts such as the human/agent interaction part, the agent security part, the agent mobility part and the ontology service part. However, Both FIPA97 and FIPA 98 specification are currently declared obsolete by a more IETF-like new experimental version called FIPA 2000.

2.1.1 Main Components

Since the FIPA standard defines at an abstract level how two agents can locate and communicate with each other. In this section, we describe briefly these architectural elements as well as their functions and relationships. In FIPA, an agent platform is the physical infrastructure in which agents are able to work. Physically, an agent platform consists of one or more machines, the operating system, the agent support software and the FIPA agent management components. Logically, an agent platform consists of three mandatory components, namely an Agent Management Server (AMS), a Directory Facility (DF) and an Agent Communication Channel (ACC). When an agent registers with the AMS of a specific agent platform, the platform is considered as the home agent platform (HAP) of this agent. Figure 2.1.1 illustrates the agent management reference model given in FIPA 97.

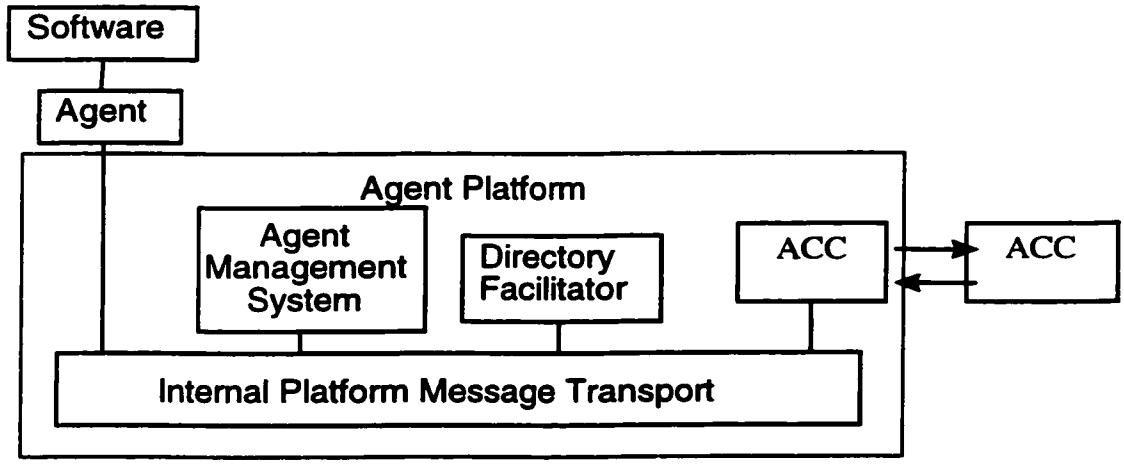

Figure 2.1.1 FIPA Agent Management Reference Model

AMS represents the authority of an agent platform, responsible for managing the operations of the platform, including the creation/deletion of agents, dynamic registration of agents and the migration of agents. Naturally no matter how many machines an agent platform crosses, there is only one AMS in one agent platform, which represents the unique authority across all these machines. In addition, an AMS maintains an index of all the agents which are currently registered with the agent platform, therefore it also provides the agent name service to all agents. The index includes an agent's unique name, which is generally composed of the agent's HAP address and a unique identifier, and its associated transport address for current agent platform.

DF is a special type of agent element which provides "yellow pages" services to other agents. An agent can query the DF for information of other agents. Similiarly, an agent must register its services with the DF to make them searchable to others. Unlike the registration at AMS, registration at a DF is voluntary to an agent. Logically, all agents registered in a DF constitute a domain, which can be regarded as an organization of agents. Usually one agent platform can support multiple domains, and likewise, one domain can span multiple agent platforms. In addition, a DF can also register with other DFs, then in this way a much larger domain can be formed by a network of DFs.

In FIPA97, ACC is the default communication method that connects all agents within an agent platform and between agent platforms. Thus an agent must have access to at least one ACC to be connected to others, and then the ACC provides the message forwarding service to the agent. FIPA97 does not define the internal communication mechanism of

agent platforms, however it specifies that, in order to be FIPA-compliant, ACC must minimally support the IIOP which is the default inter-platform communication method supported among all agent platforms. Additionally, ACC should provide reliable and orderly message routing service to route messages between agent platforms. In later version (FIPA2000), ACC and internal message transport is united into a uniform called Message Transport System (MTS).

2.1.2 Agent Communication

The agent communication in FIPA standard is based on speech act theory [16], which is derived from the linguistic analysis of human communications. More precisely speaking, FIPA regards the agent communication as the interactions of different communicative acts. In practice, a communicative act is performed by an agent sending a message to another agent, which is in a specific format described in FIPA. But the transport mechanism of the messages is not specified in FIPA standard.

FIPA is a protocol-based standard under the message-passing paradigm. That is, an agent communicates with encoded text format messages. Agent Communication Language (ACL) is a standard communication language which defines the encoding semantics and the pragmatics of the messages. Therefore in FIPA, whenever an agent needs to communicate, it must understand ACL, and at the same time the ACL enables an agent to communicate with others in a more semantically rich level. One example ACL message is shown in figure 2.1.2. Usually an ACL message must have a communicative act type to

denote the principle meaning of this message, e.g. "inform" in the example message. The body of the message is a sequence of key-value pairs representing the parameters of the message. The semantics of the message are specified formally in the document using a logical formalism known as SL.

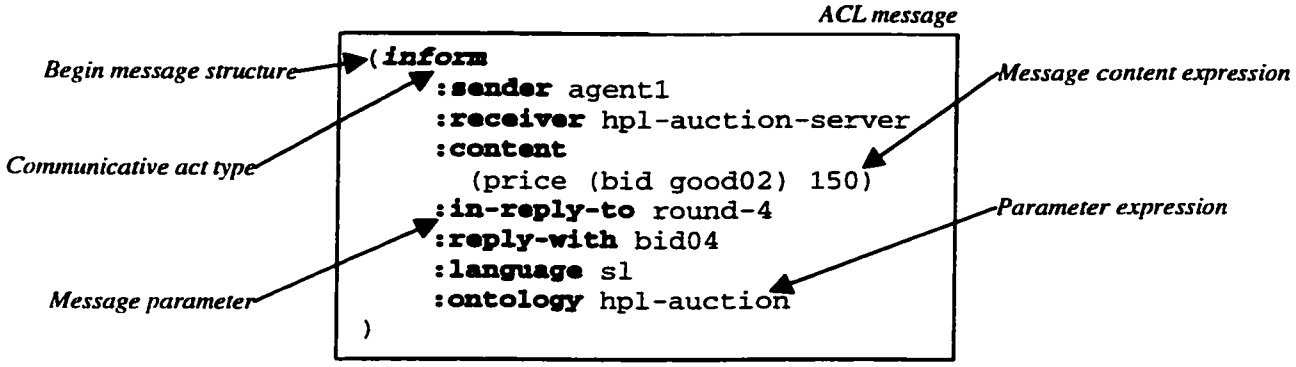

Figure 2.1.2 an Example of ACL Message

In one word, agent communication and interaction in FIPA is defined in terms of communication acts based ACL messages. To be understood, an ACL message should be processed with regard to its temporary position within a particular interaction sequence between two agents. That is, the processing of an ACL message involves the understanding of the communication acts, both the understanding of the ACL message content and the understanding of the semantics of the request.

As to the transportation protocol, FIPA defines a baseline protocol for agent platform interoperability. Rather than developing a tight specification of a communications

protocol, FIPA adopts an available protocol developed specifically for interoperability. FIPA states that in order to be FIPA compliant, an agent platform must minimally support IIOP. In this manner, the specification of a small IDL interface is sufficient to guarantee interoperability between agent platforms. However FIPA does not preclude the use of other additional common protocols.

2.1.3 Security Concerns

The specification on security of FIPA reference model was firstly listed in the agent management and security specification part in FIPA98. But until now, no FIPA-compliant agent platform report its support to this security architecture. In the FIPA2000 experimental standard, there is no separate specification for agent security, neither is it mentioned in the agent management specifications, but only as an additional note annex in the FIPA abstract architecture. Only one simple form of security is concerned in FIPA2000, that is the message validity and message encryption. Therefore, in our following discussion on the security concerns in FIPA, we can only use the security model in FIPA98 as a reference.

In the agent security management model given in FIPA 98, an Agent Platform Security Manager (APSM) is introduced to be responsible for maintaining the agent platform and the infrastructure security policies, as well as enforcing these security policies of its domain (figure 2.1.3). In order to guarantee security, APSM should be responsible for inside communications between agents, which is called the transport-level security, and

in the meanwhile creates audit trails to take precautions. In addition, when communicating with other platforms, the APSM of an agent platform should be able to negotiate the requested inter- and intra-domain security services with other APSMs on behalf of the agents in its domain. FIPA98 states that an APSM, according to its own discretion, can upgrade the level of security that is requested by an agent. However it cannot downgrade the level of services requested by an agent. If the service level requested by an agent cannot be provided, APSM may refuse the request and inform the sender agent.

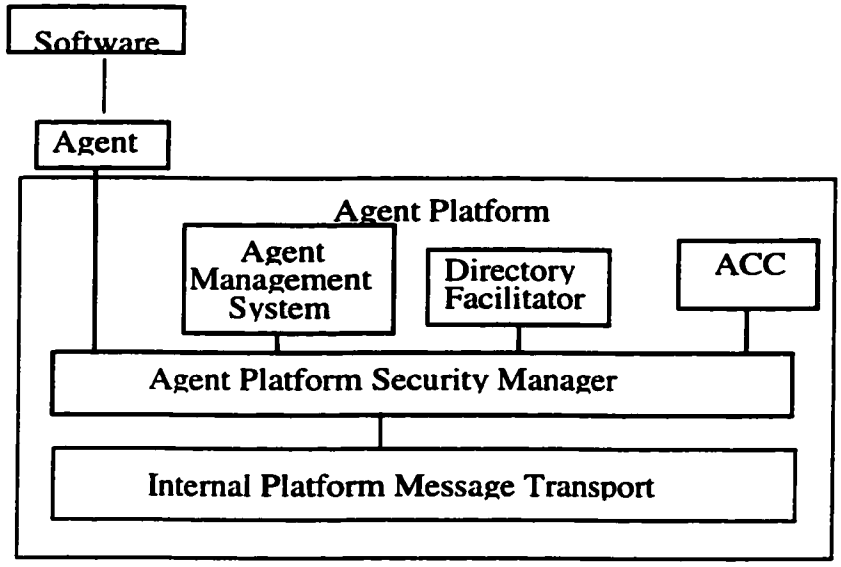

Figure 2.1.3 Agent Platform Security Model in FIPA98

In the FIPA98 security model, in addition to the new component (APSM) that is added to this security model, roles of some existing elements such as AMS and DF are enhanced. AMS is enabled to enforce the authentication of agents, based on public key

cryptography. That is, AMS may possess a public key pair and an associated certificate, which provide the basis for inter-platform security in the message transport service. Also, the AMS can keep the agents' key pairs in secure storage (e.g. hashed) for the usage of mapping to the user identities. Besides, the DF is enabled to accept security-related services. For instance, when an agent registers with DF its services, it can specify some additional parameters such as ":agent-certificate", ":owner-certificate" and ":security-encapsulation-method", therefore when some agents apply for these services, they will be required (by the DF) for certain means of authentication and encapsulation.

FIPA assumes the trust relationship between agents and their home agent platforms. As pointed out in [17], these relationships include trust between agents and AMS, trust between service provider and DF, trust between service user agents and DF, and trust between agents and the ACC.

2.1.4 Why No Complete Picture on Security in FIPA?

As we previously mentioned that one of the challenges preventing the wider acceptance of agent technology is the threat to security. Yet security remains one of the unresolved issues in the most recent version of FIPA standards. Currently there is no coherent and completed specification available for agent security in FIPA standard. The reasons for this are many, and are discussed below.

Firstly, the fundamental difficulty is that FIPA allows different agent platforms to cooperate and interact. It therefore defines only the abstract architecture and logical components of each agent platform. Similarly, in order to include security in the abstract architecture, there needs to be a clean security abstraction for the different models of (concrete) implementation. However it is difficult to design architectural elements that assure security at the abstract level while interacting with different environments or implemented differently at the concrete level. With today's technology, it is difficult to provide a satisfactory solution to the problem of combining a single security abstraction with different implementations. A good deal of familiarity with current mechanisms is required as they all address security concerns with their own techniques and policies.

Secondly, it is also important to mention that a complete security design cannot be ensured solely within the scope of an agent architecture. It should be an integral part of the software infrastructure in which the agent platform is embedded, as well as of the security of the hardware. For instance, a very secure system should include the security of related software and the operating system, or even the physical security of the computing environment. The developers of secure agent systems may need expertise in network security, computer security or other interrelated field.

We have to say that it is difficult to design a general agent security architecture that is suitable for all applications and implementations, which largely depends on implementation. Therefore we need to balance between the requirement of security level and the interoperability level. This also involves the argument of the agent-aware security

or the non-agent-aware security. Some may argue that security should always be invisible to the agent or the user, and should be mandatory to all components. However the opposing argument may insist that agents in general should be able to be aware of or control the level of the security they require. In our opinion, agents should be aware of the security infrastructure. On the one hand, agents must follow some mandatory security policies, which prevent the entire system security from being compromised. On the other hand, to be efficient and reasonable, in some special cases they should also be enabled to choose the security services or not, according to their own requirements.

2.1.5 FIPA-OS (Open Source)

There are now several agent platforms which declare their support to the FIPA agent standards, e.g. FIPA-OS, JADE, Grasshopper and ZEUS. Our Java-based implementation employs FIPA-OS (FIPA Open Source) as the underlying platform, because it is freely available and comes with an open source code. Moreover, FIPA-OS provides interfaces in API format, which enables the platform to be enhanced. FIPA-OS is a Java-based implementation from Nortel Networks Inc which conforms to FIPA agent standard and implements the mandatory elements in the FIPA 97 specification such as AMS, DF and ACC. Recently Nortel Networks Inc has moved their development support of a set of new versions from FIPA-OS 1.01(Sep, 1999) to FIPA-OS1.03 (Sep, 2000).

To be more flexible, FIPA-OS provides switchable implementations to some components. Also, FIPA-OS provides some optional components as "Parser Factory" and

"JessAgent Shell" in addition to the mandatory components in FIPA reference model. A high-level architecture of FIPA-OS is given in below figure 2.1.4. It can be regarded as a non-strict layered model, that is, entities in not adjacent layers are able to access each other directly.

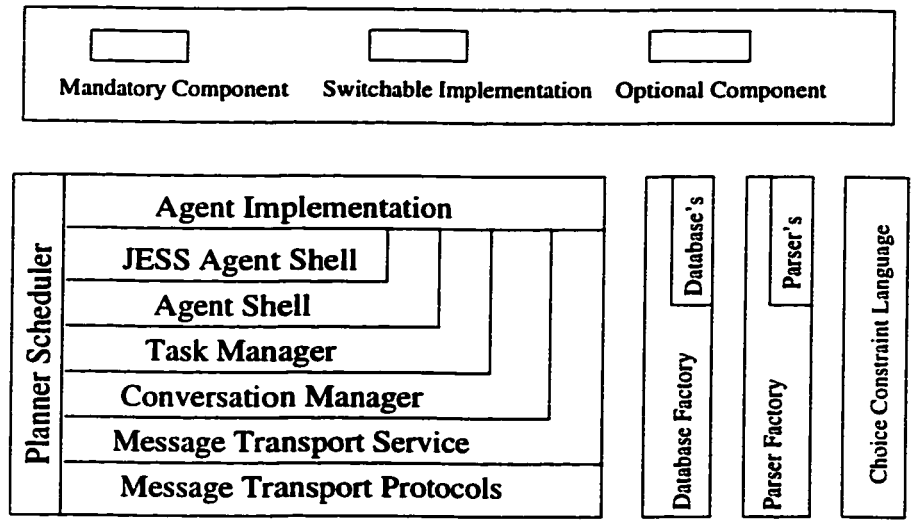

Figure 2.1.4 Components within FIPA-OS

In this figure, FIPA-OS provides agent shells (class FIPAOSAgent) to developers as a base class. Agents which are developed using these shells can use built-in support for message management (e.g. message transport, message retrieval and message buffering) and registration with platform. In a higher layer, the JessAgent provides an interface to Java Expert System Shell (JESS) so that an agent can use a knowledge base to reason. This interface may help the developers to write FIPA-OS intelligent agents.

FIPA-OS also enables the development of agents without the restriction of agent shells. In the task manager layer, the functionality of an agent is split into smaller units known as "tasks". FIPA-OS enables the development with highly reusable tasks, and an agent can execute multiple tasks simultaneously. A task manager takes care of routing incoming messages and other events to the right task. In a much lower layer, a conversation manager provides the ability to track conversation states at the performative level, and then group messages of the same conversation together. FIPA-OS supports the development with individual conversations as well. Finally, the message transport service layer provides to developers the basic ability to send and receive messages to an agent implementation.

According to the evaluation of FIPA-OS 1.03 given by Mikko [18], FIPA-OS fulfilled the requirements for a FIPA-compliant agent platform well, as well as its software quality interoperability and scalability. However, one shortcoming of FIPA-OS mentioned in the paper is that, FIPA-OS exploits too many third-party tools, such as XML-parsers, ObjectSpace Voyager and Apache Web-server. These tools should be included in FIPA-OS itself or at least be included in its distribution package.

2.2 Cryptography: a Key Technology for Information Security

A cryptosystem or a cipher system is a method of disguising messages so that only certain people can get the real message, and cryptography is the way to create and use cryptosystems. Usually in a cryptosystem, the original message is called a plaintext, and

the disguised message is called a ciphertext. The procedure to convert plaintext into ciphertext is called the encryption, and the converse procedure to convert ciphertext into plaintext is called the decryption. The people disguising a message is the sender, while the people who is supposed to be able to get the real message is called the recipient. Figure 2.2.1 illustrates the relationships of these elements.

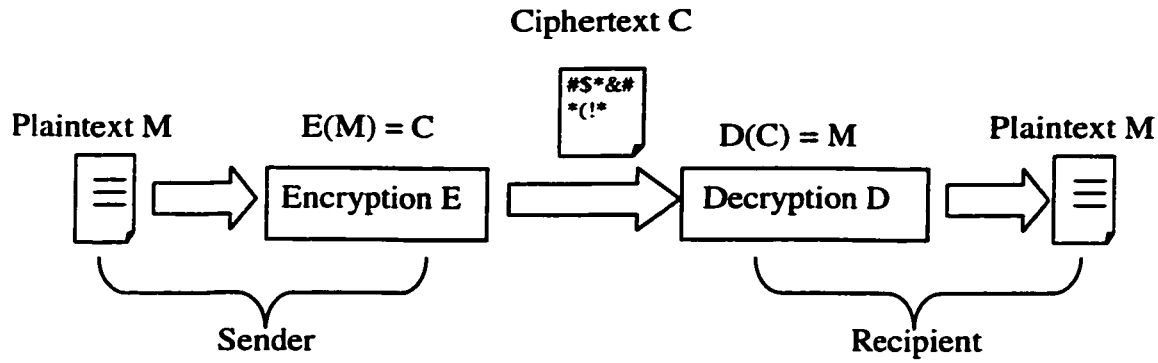

Figure 2.2.1 Basic Components in a Cryptosystem

In general, there are four basic requirements of a cryptosystem, namely confidentiality, authentication, integrity and non-repudiation.

- **Confidentiality** of a message assures the recipient that the message was kept secret during its transmission, only the recipient or someone authorized can view it, an eavesdropper will never be able to view or derive the original plain text. Generally, before sending, the original plaintext is encrypted to ciphertext with one or several cryptographic algorithms. When receiving, the ciphertext is decrypted by the same algorithms.

- **Integrity** of a message enables the recipient to verify whether the message has been modified or substituted during the transmission. Attackers cannot tamper with the message without being detected. Calculating a one-way hash of the message and sending it along with the message is the common method.
- **Authentication** enables the recipient to ascertain the origin of the message, that is who sent this message. Therefore no attacker can send a message to the recipient and pretend to be another sender. Usually, the sender's digital signature of a message is used to prove his/her identity.
- **Non-repudiation** allows the recipient to show evidence and therefore convince others that the sender did send the message, which the sender cannot deny. The message cannot be sent by others or made by the recipient. Generally, additional information such as a timestamp is needed in addition to the sender's digital signature.

2.2.1 Symmetric-key vs. public-key cryptography

Conventional cryptosystems are based on symmetric key cryptography, the so-called secret key cryptography. In the symmetric key cryptography, the same key is used for both encryption and decryption and shared between the sender and the recipient. Thus the encryption algorithm greatly depends on the secure distribution and storage of the secret key in each side. In a more recently asymmetric key system, the so-called public key cryptosystem, a pair of related key are used, one public key used in encryption and one private key used in decryption. Private keys are kept secret for personal use, whereas public keys can be publicly distributed without compromising the security of the system.

Therefore the key distribution and management between the sender and the recipients is much safer in public key cryptosystem than that in conventional symmetric key cryptosystem.

However, the computational performance of the public key encryption is inferior to that of symmetric key encryption. Because usually the effective attack on symmetric key systems is an exhaustive key search, whereas in public key systems, most well known public key cryptosystems (such as RSA) are subject to "short-cut" attacks (e.g., factoring) which is more efficient than exhaustive key search. Therefore, in order to achieve equal security, private keys in public key systems must be larger (e.g., 1024 bits for RSA) than secret keys in symmetric key systems (e.g., 64 in DES or 128 bits in IDEA). This leads to the fact that a public key cryptosystem is often less efficient than a secret key cryptosystem.

From the discussion above, we can see that symmetric key and public key cryptography have a number of complementary advantages, which enable them to be exploited in the same cryptosystem with their different strengths. We found that the public key cryptography is good at efficient signatures (particularly non-repudiation) and key management, while the symmetric key cryptography is efficient in the encryption or the decryption of data. Therefore, they can be possibly integrated into one system by using public key encryption techniques in authentication and key distribution, and then using a symmetric key system in the encryption and decryption of messages. This is because generally data encryption is the most time consuming part of the encryption process, and

the public key scheme for key establishment is a small fraction of the total encryption process between the sender and recipient. In this scenario, we can take advantage of both the long-time nature of the public key scheme and the performance efficiency of the symmetric key scheme.

2.2.2 Cryptographic Algorithms

DES, Triple-DES

The Data Encryption Standard (DES) [19] is the most well known symmetric key block cipher with openly and fully specified implementation details. It is published by National Security Agency (NSA) in mid of 1970s, based on an algorithm developed by IBM.

DES takes 64-bit plaintext blocks as input, outputs 64-bit ciphertext blocks. However, the effective size of the secret key is only 56 bits. The operations consist of 16 rounds of permutation and S-Box substitution, P-Box permutation. In each round (i.e. the *i*th round), L*i* and R*i* are the left and right halves of current block, K*i* is the 48-bit key in this round, then the substitution and the permutation operations can be described as:

$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$

Because DES is designed for security and hardware implement efficiency, it is expensive to break it by software methods only. However, because of its limited key length and the

rapid growth in speed of computing machines, it seems DES will become less secure as time goes on.

Some variants of DES are proposed to help its shortcomings, one of them being triple DES. As the name indicates, triple DES operates on 64-bit data blocks, and uses the DES cipher three times. There are several forms of triple DES, some use two 56-bit keys and some use three keys. Because the former is more popular, some people may describe triple-DES as E(K1,D(K2,E(K1,x))).

Since triple DES is based on the DES algorithm, it is very easy to modify existing software to use triple DES. Its longer key length also eliminates many of the shortcut attacks that can be used to break DES therefore providing the advantage of reliability. In the near future triple DES is a promising choice for the security needs of highly sensitive information.

RC2, RC5

RC2 and RC5 [20] are proprietary symmetric algorithms of RSA Data Security Inc., and are considered as alternatives to DES. Both RC2 and RC5 have an extremely compact description, which is suitable for hardware or software implementation. RSA claims that RC2 and RC5 are faster than DES when implemented in software.

Unlike DES, RC2 and RC5 are variable-key length algorithms. The RC5 block cipher has a word-oriented architecture for variable word sizes, thus the number of rounds and the

key byte-length are all variable. By specifying different key lengths, RC2 and RC5 can be configured to provide greater or lesser security. Usually a 128-bit key makes a brute force attack on RC2 or RC5 not feasible. Because they have special export status, several products such as Lotus Notes and Netscape Navigator make uses of these algorithms.

IDEA

IDEA (International Data Encryption Algorithm) is published by Xuejia Lai and James Massey in 1990 [21]. It is one of the most secure block algorithms available now because of its impressive theoretical foundations.

IDEA is an iterated block cipher that operates on 64-bit blocks of plaintext. The procedure is quite complicated. It carries out a series of modular arithmetic and XOR operations on segments of the 64-bit plaintext block, using the sub-keys generated from the key. The encryption scheme uses totally fifty-two 16-bit sub-keys. The 128-bit key length in IDEA, which is longer than twice the key length of DES and is longer than that of triple-DES also, provides adequate security. Besides, it is a newer algorithm and has not been studied as much as DES. Therefore, IDEA encryption is substantially faster and generally considered to be much more secure than DES encryption.

IDEA algorithm is patented in the United States and some other countries which restrict its commercial applications. But its usage in PGP makes it the most widely used encryption algorithm in the Internet electronic mail, and individual users of PGP have a royalty free license to use the IDEA algorithm.

RSA

The RSA algorithm [22], which is named after its inventors R. Rivest, A. Shamir and L. Adleman, is the most widely used public-key algorithm. Currently, it is widely used to provide both secrecy and digital signatures. Security of the RSA algorithm is based on the intractability of the integer factorization problem.

The RSA algorithm can be roughly described as follows: in order to generate a public and private key pair, firstly a user should find two random large prime numbers, p and q, then compute the value of $n = pq$. Afterwards, the user randomly selects another prime number e (encryption key) which is relatively prime with $(p-1)(q-1)$. The decryption key d can be derived by the formula $d = e^{-1} \bmod ((p-1)(q-1))$, using the extended Euclidean algorithm. Now the numbers (e,n) are the public key, numbers (d,n) are the private key (n is public, only d is private), and the prime numbers p and q are not needed and should be discarded.

In order to encrypt a message m, the user must divide it into a number of blocks, the length of each piece is less than the number n. Then for each message block m_i, the user can calculate a cipher message block c_i, simply by following the formula $c_i = m_i^{\ e} \bmod n$ (each c_i has the same size of m_i.) Similarly, in order to decrypt a message, the user has to follow another formula $m_i = c_i^{\ d} \bmod n$. The message block can be recovered correctly because: $c_i^{\ d} \bmod n = (m_i^{\ e})^{d} \bmod n = m_i^{\ ed} \bmod n = m_i^{\ k(p-1)(q-1)+1} \bmod n = m_{i*} m_i^{\ k(p-1)(q-1)} \bmod n = m_i * 1 == m_i$. Finally, the user concatenates all the blocks m_i and then gets the recovered message m.

Although there are numerous ways to speed up RSA encryption and decryption in software and hardware implementations, even with those improvements, RSA encryption/decryption is substantially slower than the commonly used symmetric key encryption algorithms such as DES. Therefore in practice, RSA encryption is most commonly used for the transport of symmetric key encryption algorithm keys and for the encryption of small data items.

2.2.3 One-Way Hash Function

Roughly speaking, a hash function is a method to map strings of arbitrary finite length to strings of fixed length. It takes a message as the input and produces an output which is referred to as a "hash". From a high level, different hash functions can be divided into two categories: one is the unkeyed hash function which leads to modification detection codes (MDC), it is used to detect the modification of a message only. The other is the keyed hash function that leads to message authentication codes (MAC) or data authentication codes (DAC), it can be used to authenticate the sender in addition to detect modification. However, generally people refer to unkeyed hash functions when they use the term "one-way hash function". The generation of MAC and MDC is simply illustrated in figure 2.2.2.

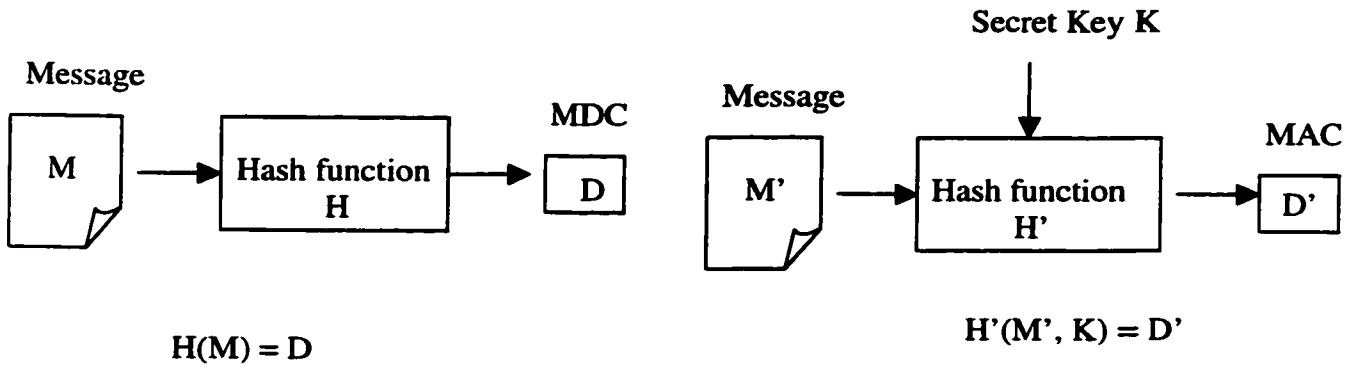

Figure 2.2.2 MAC and MDC

Unlike encryption operations, one-way hash functions cannot be reversed, and they are constructed in such a way that the probability of two pieces of arbitrary plain text being hashed to the same value is very small. It is theoretically possible that two distinct messages could be hash into the same value, this is called a collision. Collisions cannot be avoided entirely, but in a good hash function, the probability of collisions should be extremely low, which means that, any minor change in plain text can be reflected in a different hash value. Usually, one-way hash values range from 112 to 160 bits long. The longer the hash value, the more secure the function. In the following part of this section, we will introduce some one-way hash function algorithms and give a short overview of MAC as well.

MD4, MD5

MD stands for "Message Digest". MD4 is a 128-bit one-way hash function designed by Ron Rivest [23]. Because collisions have been found in its compression function computations, MD4 is no longer recommended for use as a collision-resistant hash function. MD5 is a strengthened version of MD4. MD5 is similar to MD4 in design, and

also produces 128 bits hash value. Some improvements are made in MD5 and it is more complex. MD5 is publicly available and it is currently widely used by most e-mail security programs such as PEM, PGP, and S/MIME. However, it has also now been found to have weaknesses in compression function.

SHA

SHA (Secure Hash Algorithm) [24] is based on MD4, and was proposed by NIST in 1994 for use with the digital signature standard. SHA-1 modifies some core steps in MD4, and is different with it in other aspects such as the length of hash-value, chaining variables and rounds in compression function.

Because the hash value of SHA is 160 bits long, it provides increased security compared to the 128 bits long MD5, and is considered stronger. This long hash-value also helps to defeat brute-force attack, which makes SHA a highly recommended hash function.

MAC

Message authentication code (MAC) is a distinct class of key dependent one-way hash functions, which takes both a message and a secret key as inputs, instead of a message only. Since it is not feasible for anyone to produce the same output of the message without the knowledge of the key, and only someone with the identical secret key can verify the hash, MAC can be used to provide integrity and message origin authentication as well as the identification in symmetric key schemes.

There are some specific MAC algorithms such as Message Authenticator algorithm, RIPE-MAC and IBC-Hash. Besides, a MAC can be derived from any ordinary one-way hash function. One simple way to make a key-dependent hash function is to encrypt the hash value of a one-way hash function with a symmetric algorithm (i.e. CBC-MAC). Another method is to concatenate the key and the message, then do the computation of this concatenation with a one-way hash function [25]. Also, a MAC can be changed to a one-way hash function by simply making the key public to everyone.

2.2.4 Digital Signature

Generally speaking, the digital signature of a message is a piece of information which is generated depending on some secret known only to the signer, and additionally on the content of the message being signed. It is verifiable and non-repudiatable. In other words, when a dispute (such as whether a user has signed a document or not) arises, an unbiased third party is able to resolve the matter equitably.

Digital signatures can be implemented in both symmetric cryptosystems and public key cyptosystems. In a symmetric cryptosystem, messages are signed by a secret key shared by two users. Therefore one user cannot verify to the third-party that a message is actually sent from the sender and not faked by himself, whereas in a public key crcyptosystem, which is preferred in a digital signature, each user has a unique private key to identify himself and sign his messages. Further, since public key algorithms are too inefficient to sign the entire document, in practice, hash functions are used for data

integrity in conjunction with digital signature schemes. Typically a message is hashed first, and then the hash value, as a representative of the message, is signed in place of the original message. Additionally, the original message can be encrypted before being sent out. It is more reliable to sign on the original message than the encrypted message. A general process of the generation and the verification of a digital signature are illustrated in following figure 2.2.3.

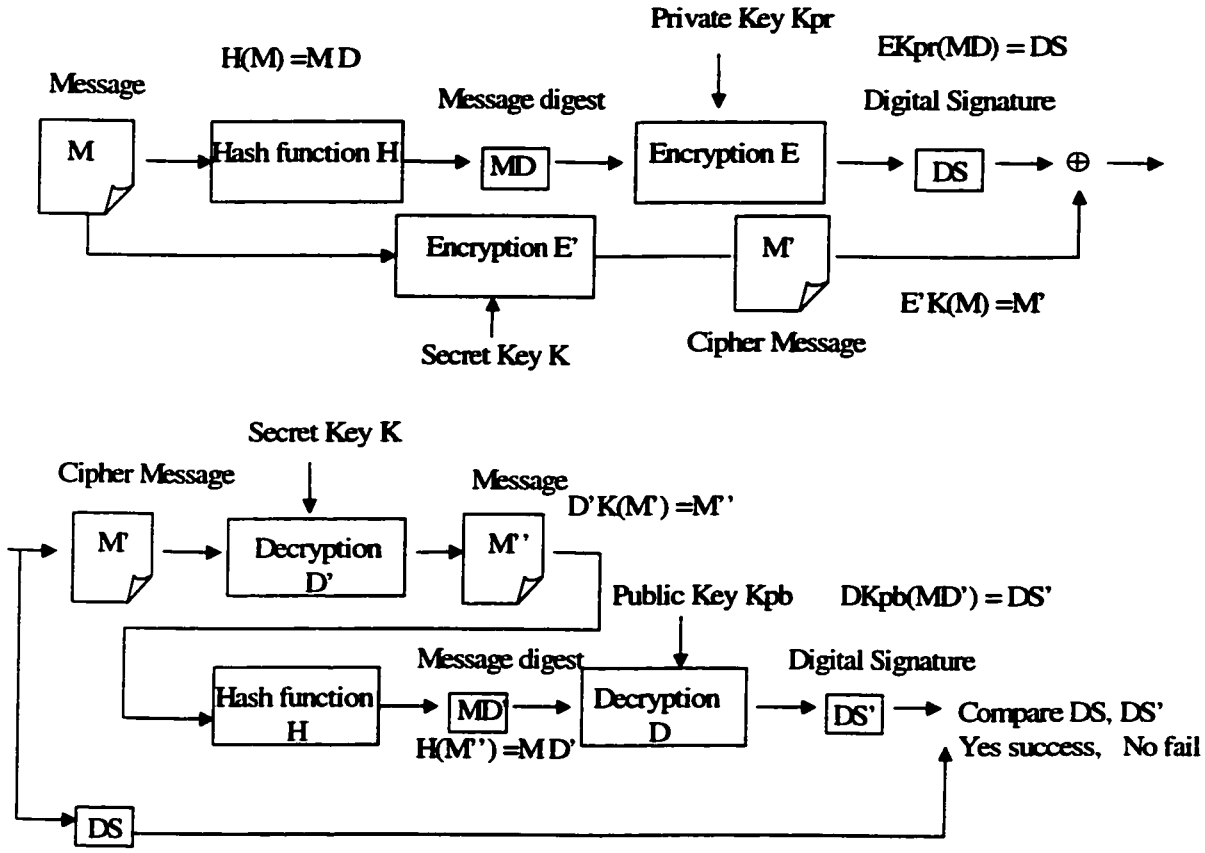

Figure 2.2.3 Generation and Verification of Digital Signature

The first digital signature method is the RSA signature scheme, which remains to be one of the most practical and versatile techniques available today. Subsequent research has resulted in many alternative digital signature techniques such as DSA which offer significant advantages in terms of functionality and implementation.

Digital signature has many applications in information security, including authentication, data integrity, and non-repudiation. One of the most significant applications of digital signatures is the certification of public keys in large networks, in which a trusted third party is able to bind the identity of a user to a public key by means of a certificate. Afterwards, other users can authenticate the public key of a user without assistance from a trusted third party.

2.2.5 Authentication Protocols and Key Exchange Protocols

Authentication is by definition a process to verify one's claim of identity, and an authentication protocol is designed to help some involved parties to authenticate each other. Usually, authentication can provide assurance that the proper users or servers are being contacted by using the possession of a secret (e.g. password, secret key or private key).

Password is the simplest authentication mechanism based on sharing of a secret. In a password based authentication protocol, each user has its own specific password. Usually instead of storing the original passwords, most systems only store one-way functions of

the passwords, which just enable them to differentiate valid passwords from invalid passwords, and therefore lower the threat of revealing all passwords to someone who breaks through the system. However, in an open network instead of a workstation, the user's password is vulnerable while in transmission from user to remote host. Generally, public key cryptography or secret key cryptography can be used to help this, that is, an authentication protocol can be based on conventional cryptosystem or a public key cryptosystem, or both.

Unlike one-way authentication, in which only two principles or parties are involved, one needs to verify the identity of another, in a mutual authentication, pairs of principles or parties are involved, who need to satisfy themselves mutually about each other's identity. Some protocols combine the authentication process and key exchange to solve this problem. Suppose two users called Alice and Bob do not know each other, but wish to talk through open network secretly. Usually a third party (usually called trusted server) is introduced which is trusted by both Alice and Bob, and proves that they are talking with the people they wish, then helps them to achieve a secret key that is shared only by themselves. In a secret cryptosystem, the trust party may have a specific secret key shared with each of the users. Therefore the trust server is able to authenticate each user individually and provide him/her a proof to authenticate with each other. Whereas in a public key system, all users keep their own private keys that nobody else can access, and the trusted server keeps the public keys of all users. Because only the user can access its own private key, no one else can impersonate him/her by replying the correctly encrypted challenges.

In the rest of this section, we give two examples of authentication protocols, one on symmetric schema called Needham-Schroeder, and the other is on public key schema called DASS.

Needham-Schroeder

This protocol invented by Needham and Michael Schroeder [26] using symmetric cryptography can effectively prevent replay attacks, figure 2.2.4 illustrates the main steps of this protocol. One of the variants of this protocol is Kereros.

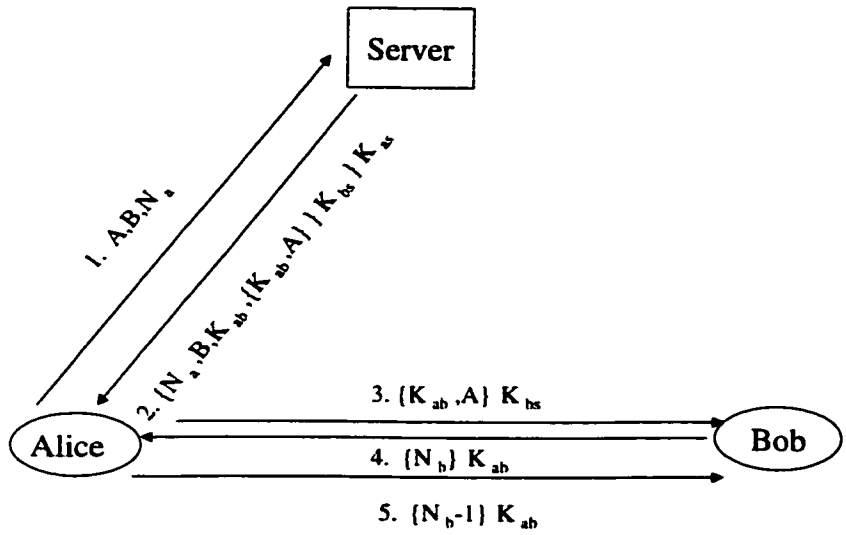

Figure 2.2.5 Needham-Schroeder Authentication and Key Exchange Protocol

(1) Alice sends a message to Server consisting of her name, Bob's name, and a random number.
(2) Server generates a random session key. It encrypts a message consisting of a random session key and Alice's name with the secret key she shares with Bob. Then it encrypts Alice's random value, Bob's name,

the key, and the encrypted message with the secret key he shares with Alice. Finally, it sends her the encrypted message

(3) Alice decrypts the message and extracts K_{ab}. She confirms that N_a is the same value that she sent Server in step 1. Then she sends Bob the message that Server encrypted in its key.

(4) Bob decrypts the message and extracts K. He then generates another random value, N_b. He encrypts the message with K_{ab} and sends it to Alice.

(5) Alice decrypts the message with K. She generates $N_b - 1$ and encrypts it with K. Then she sends the message back to Bob.

(6) Bob decrypts the message with K and verifies that it is $N_b - 1$.

DASS

Another important protocol is DASS (Distributed Authentication Security Service) protocol [27] which provides mutual authentication and key exchange. DASS uses both public-key and symmetric cryptography. Each user has its private key and the server has their public keys. This protocol follows the following steps:

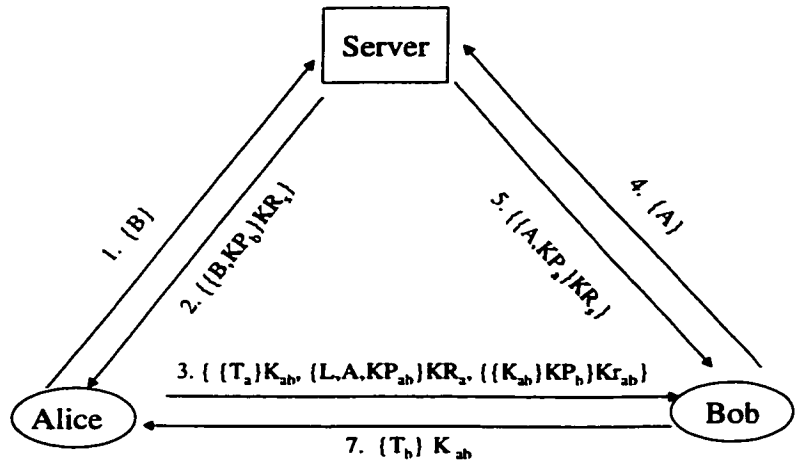

Figure 2.2.6 DASS Authentication and Key Exchange Protocol

(1) Alice sends a message to Server, consisting of Bob's name

(2) Server sends Alice Bob's public key, signed with Server's private key. The signed message includes Bob's name.

(3) Alice verifies Server's signature to confirm that the key she received is actually Bob's public key. She generates a random session key K_{ab}, and a random public-key/private-key pair KP_{ab}/KR_{ab}. She encrypts a timestamp with K_{ab}, then she signs a key lifetime L, her name A, and KP_{ab} with her private key KR_a. Finally she encryptes K_{ab} with Bob's public key, and signs it with KR_{ab}. She sends all of this to Bob.
(4) Bob sends a message to Server, consisting of Alice's name, A.
(5) Server sends Bob Alice's public key, signed in Server's private key, then signed message includes Alice's name.
(6) Bob verifies Server's signature to confirm that the key he received is actually Alice's public key. He then verifies Alice's signature and recovers Kp. He verifies the signature and uses his private key to recover K. Then he decryptes T_a to make sure this is the current message.
(7) If mutual authentication is required, Bob encrypts a new timestamp T_b with K_{ab} and sends it to Alice.
(8) Alice decrypts T_b with K_{ab} to make sure that the message is current message.

2.2.6 Others -- Base64 Encoding and Decoding

Although it may seems like an "encryption algorithm", Base64 coding is in fact an encoding scheme defined in MIME (Multipurpose Internet Mail Extensions, RFC 1521). It is designed to be a robust scheme for all possible transformations during the transmission of a message in the Internet. Therefore, the main function of Base64 is to translate binary octets to ASCII code, in which format the message can be transmitted through most email gateways. This encoding is virtually identical to the one used in Privacy Enhanced Mail (PEM) applications, which is defined in RFC 1421. Base64 coding uses a 65-character subset of US-ASCII, and enables that each printable character is represented by 6 bits. The output is limited into text lines with each line containing exactly 64 printable characters, except that the last line might be shorter. The encoding and decoding algorithms of Base64 are simple, and the encoded data are always larger than the unencoded data.

CHAPTER 3

FRAMEWORK DESIGN

In this chapter, we present the detailed design of our security framework. The motive behind the security framework design is to create a "secure agent space" where a diversity of reliable agent platforms are safely inter-connected, different types of agents execute autonomously, interacting with each other under the agent communication language. In this agent space, the agent-to-agent communication is secure no matter if they execute in the same agent platform or not. Agents are able to execute in any of these reliable platforms safely in addition to their own home platforms.

3.1 Security Framework

In order to add security features to the FIPA standards, we propose an architecture consisting of a two-layer agent platform. A FIPA-OS agent platform is employed as the agent management and communication infrastructure, while an upper layer is designed as a security extension to the FIPA-OS. As we mentioned before, the FIPA-OS has implemented the mandatory elements defined in the FIPA97 specification and also supports agent interoperability in that specification. The separate security layer provides security-related services to agents. Figure 3.1.1 roughly illustrates this architecture.

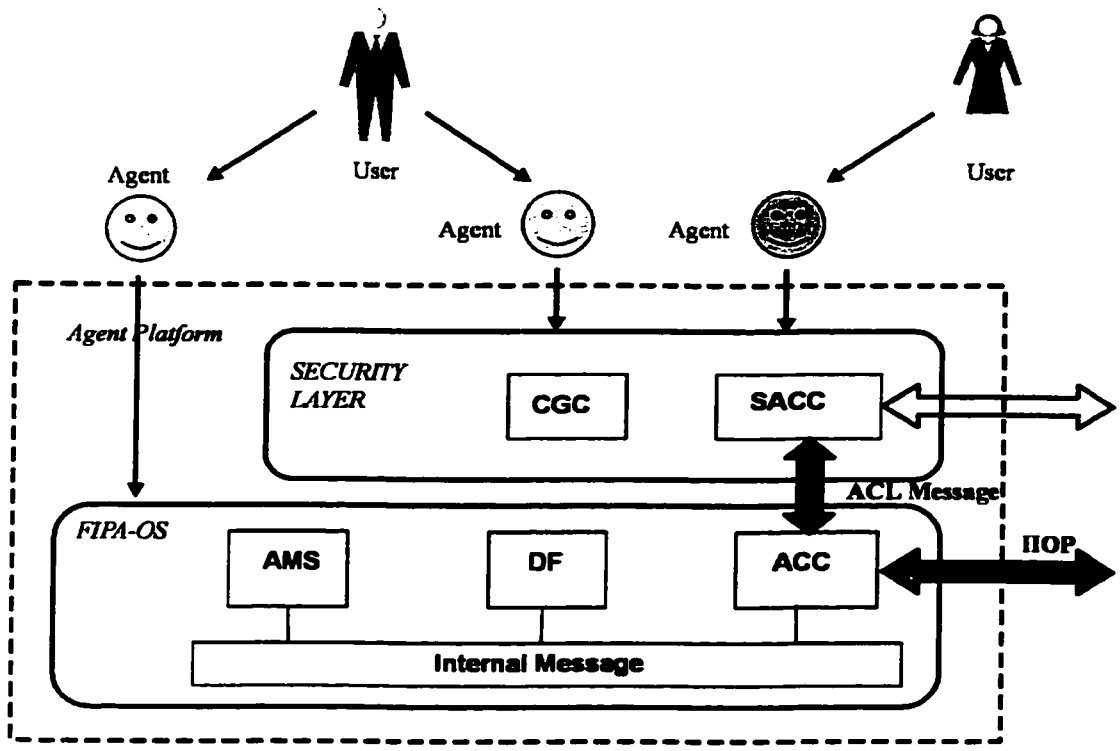

Figure 3.1.1 Two – Layer Architecture

The relationship between these layers is twofold. First, the security layer relies largely on the underlying agent platform for communication and agent management. For instance, major components of the security layer such as the Secure Agent Channel Communication (SACC) and the Credential Granting Center (CGC) are agents registered in the AMS of the local FIPA-OS agent platform. The SACC implements its secure communication service partly by requesting service provided by the ACC in the FIPA-OS. Second, the separation of the security layer from FIPA-OS frees security mechanisms and policies from specific agent platforms. Only services defined in the FIPA specification are used by components in the security layer. That is, the security

layer only involves the defined functions of the underlying agent platform rather than the details of its implementation techniques.

The principal benefit of the separation of the security layer from the basic agent platform layer is therefore the independence of the security mechanisms. Different platforms with specific security mechanisms are able to interact under the same security architecture. Hence, this architecture focuses more on the security functions of a layer than on the specific techniques necessary to implement those functions.

There are many advantages to this two-layer architecture. First, as mentioned before, the FIPA aims to integrate, and achieve a high degree of interoperability between, different platforms. But since the security specifications in FIPA97 and FIPA98 have been declared obsolete in the recent FIPA2000, FIPA currently has no coherent agent security architecture. Our architecture will therefore provide security services, while still complying with the goal of the FIPA. Second, security preparations or cryptographic calculations consume a lot of time and computation power. As a result, for reasons of efficiency, some agents may wish to omit security services for unimportant messages. This two-layer architecture provides both security services and basic agent platform services simultaneously, thereby balancing the importance of the message with the need for efficiency.

The security layer in an agent platform ensures security services. Secure communication and execution environment services are provided to address the requirements known

respectively as Communication Security (type A in fig 1.2.1) and Computation Security (type B in fig 1.2.1).

3.1.2 Terminology

The major entities of our system are agents (mobile and stationary), agent platforms, and users. Throughout this thesis we use terms that may be ambiguous or used differently elsewhere, therefore below we state our use of those terms first: user, agent, and agent platform.

By user (or legitimate user), we mean a human being who wish to fulfill his/her goal by using the services of our system, and is permitted to do so. An agent is a user-defined program, performing its task as planned, which is also a FIPA-OS agent. Unlike a system agent, which is created as a component of the agent platform, each user-defined agent must be relevant to a user who created it or using it, called the owner of the agent. An agent platform is the combination of a security layer and a basic FIPA-OS agent platform. Often, an agent executes on an agent platform on behalf of a user.

Each entity in the system, be it a user or an agent or an agent platform, needs a proof of its identity. Different types of proofs for them will be described in the trust model in next section. Besides, in this thesis, we use the term of "resource" as an abstract specification of the information service provided by the system, e.g. a special file (in format of text), a piece of music (audio file) or a clip of movie (video file).

3.2 Security Concerns and Trust Model

Since it is very unlikely to implement an agent system that is resistant to all kinds of security risks, in this section we clarify the types of attacks our design addresses, as well as the problems we are not trying to solve.

In our system, we explicitly address the threats of (a) the mobile agent against the execution environment (an agent platform in our system) (b) outside network against the mobile agent's communication and migration. These are the threats of category 1 and 4 respectively in the attack model. Besides, the attacks from a malicious agent to other agents such as the masquerading attack are prevented implicitly by the authentication of agents. Our approach provides two security services to agents, which are known as the secure communication service and the secure execution environment service.

The secure communication service eliminates some potential threats from network communication. The threat of platform masquerading is prevented by the agent platform authentication and its digital signature in messages, and the threat of eavesdropping is diminished by encrypted ACL messages which are processed in security layer before they are sent out.

The secure execution environment service eliminates the threat of unauthorized access to an agent platform. Agents are required to present valid permission issued by current platform before they are allowed to access some special resources or services. Therefore critical resources and services are under the protection and control of the agent platform.

Given the attack model in chapter 1 (fig 1.2.1), one category of security threats we do not touch is the risk of the malicious agent platform against the agent. A lack of effective methods prevented our architecture from dealing with the risk from a malicious agent platform. Instead, both the security layer and the basic layer in the agent platform — functional components such as ACC, AMS, DF, SACC and CGC, as well as the internal message communication of the platform — are regarded as safe. However, we differ slightly from the assumption in the FIPA specification discussed in [17] by considering trustworthy only those platforms known to a Certificate Authority (CA). In other words, we consider an unknown platform, or one without adequate proof (a digital certificate) from the CA, to be suspicious. Other platforms may therefore refuse to send critical messages and may prevent the migration of agents to this platform. The CA is the root of all trustworthy relationships in our system.

Our basic assumptions about the trust relationship of all entities, including agent platforms, agents and users, are given below. These assumptions constitute the trust model in our system, as illustrated in figure 3.2.1. All other relationships among entities outside these assumptions are not considered trustworthy.

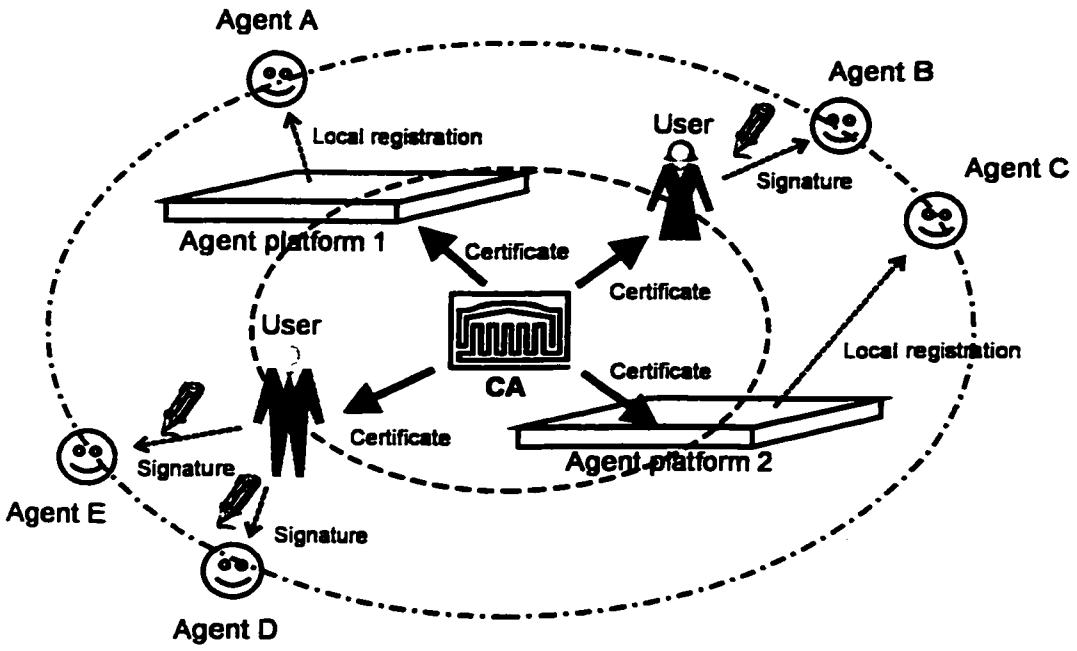

Figure 3.2.1 Trust Model

• A CA is reliable to all entities within a system's scope: the CA is the root of trust. All certificates issued by the CA are therefore trustworthy.

• Users or agent platforms with genuine certificates are trustworthy: the CA will issue certificates only to those users or platforms it knows very well, whose intentions and actions can be guaranteed to be not malicious. Platforms holding valid certificates are therefore trustworthy, and a user with a valid certificate is a legitimate user.

• Agents signed by trustworthy users are trustworthy: the user who has signed an agent, whether mobile or stationary, must have a full knowledge of its workflow and behavior, and thus all the possible consequences of its operation. Legitimate users and

platforms are responsible for any agents that they have signed, therefore would never sign a susceptible agent.

• The agent trusts its home agent platform: these platforms are not malicious. When an agent registers to its Home Agent Platform (HAP), it assumes that the HAP is trustworthy, and therefore entrusts its profiles to it. This platform includes components in the FIPA-OS platform layer (e.g. ACC, AMS, DF), components in the security layer (SACC, CGC, Authenticator), and the internal communication channel.

3.3 Secure Communication Service

The secure communication service diminishes the potential risks in the agent-to-agent communication, either passive attacks such as eavesdropping, or active attacks such as message modification. In our design, the security layer, mostly the components centered with Secure Agent Communication Channel (SACC), thwarts network-based attacks by a combination of methods such as authentication, encryption and integrity protection. Cryptography is the key technology in protecting communication safe.

3.3.1 Communication Model

Usually, a complete communication model of an agent-based system includes the intra-platform communication and the inter-platform communication. Since one assumption in our trust model is the safe internal platform communication, once an agent (mobile or

local) is approved to be registered in the current platform, its local communication with other agents is regarded as safe and therefore needs no protection. Only threats in the inter-platform communication are considered in our system.

In order to provide safe inter-platform communications, an agent platform in our system will try to set up a secure communication channel with each of its neighboring platforms during its initiation. The establishment of the links is based on the platform authentication, because an agent platform is trustworthy only with a valid certificate. Furthermore, if two platforms can agree on the security level and the cryptographic options in a negotiation, then a platform-to-platform secure communication link is ready to provide the platform level secure communication service.

However, in some cases the platform-to-platform link is not secure enough to satisfy an agent's need. An agent may have more restrictive requirement in communication, that is, an agent may wish to confirm which agent sent the message rather than knowing only from which platform the message is sent. In this case, an additional agent authentication is performed between two agents in a higher level called the application level in format of agent's cooperation actions. Afterwards, each agent is able to add its own signature for a message, which enables the recipient agent to verify the sender's identity, before sending the message to the secure link. We can say that a so-called agent-to-agent secure communication link is set up between these two agents.

If we consider a platform as a secure domain, then SACC can be regarded as the only secure communication channel of this domain. Note that when an agent use the secure communication service, from the agent's perspective, the communication between two different platforms are via SACC. However, from the platform's perspective, all message communications are still via ACC. When more than one platforms are being contacted, the SACC will keep a specific cipher module for each of the platforms, as shown in figure 3.3.1.

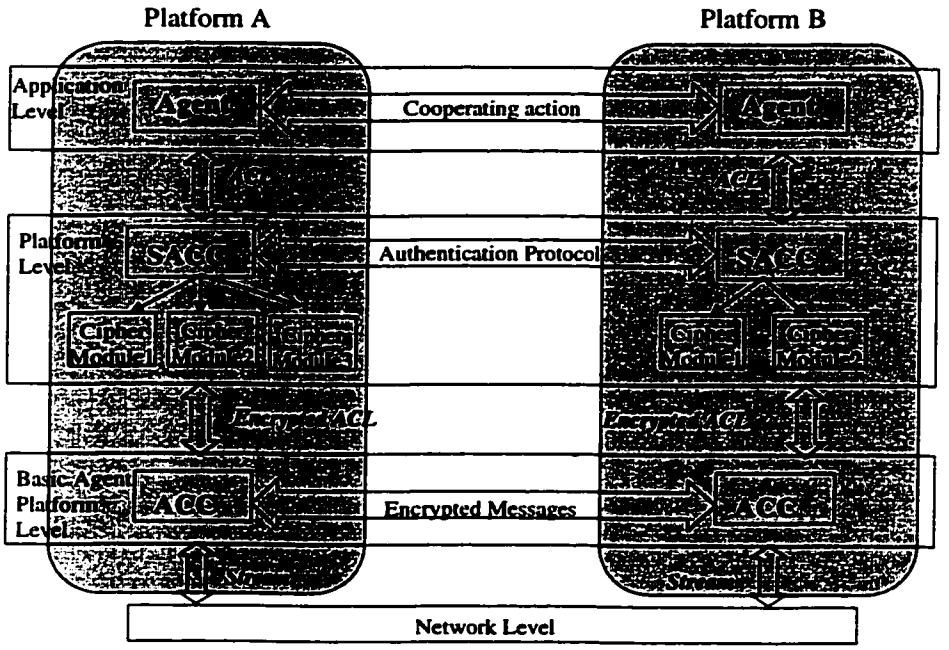

Figure 3.3.1 Communication Model

3.3.2 Platform Network

As we mentioned, our objective in agent communication is to connect a diversity of agent platforms into a secure platform network. Although an agent platform may not have a direct safe link to each of other platforms, it must have point-to-point secure channels to some of its logically adjacent nodes, or more precisely, a secure channel from one SACC to another. These connections may be in different security levels, e.g. "low", "medium", "high" or "none", as a result of their negotiation.

The establishment of a secure link can be divided into two stages called a mutual authentication stage and a bilateral negotiation stage. The mutual authentication enables one platform to know and verify the identity of the other platform. The bilateral negotiation enables both platforms to share the same security level and cryptographic algorithms and parameters. As a result, both platforms will share the same cipher module, which includes a session key, a signature object, a symmetric cipher object and an asymmetric cipher object, etc. After the secure link is set up successfully, in the sender platform, outgoing messages can be encrypted and then encapsulated by this cipher module in SACC. And a reverse procedure is done by the SACC of the receiver platform.

Obviously, this two-stage mechanism, mostly the bilateral negotiation, brings about some advantages: it imposes no assumption that all platforms in the system must have the uniform security capability. Different platforms can use different algorithms according to

their actual needs and situations. Moreover, this mechanism supports different security levels between platforms, therefore providing more options to agent platforms.

After a platform's initiation, it has a individual secure communication channel with every adjacent node. Then, as more and more platforms are initiated, different security links are connected one by one and finally a secure platform network comes into being. In the delivery of a message from one platform to another, maybe several secure paths are available, and possibly one of them is the direct path from the sender platform to the receiver platform. The other paths, more than one stop, consist of several links that are concatenated one by one. Usually, the direct path is automatically chosen in a message's delivery for reasons of efficiency, convenience and high speed. However, in some special circumstances, an agent might choose an indirect link to send out its message with the help of some intermediate platforms. One possible reason is that no direct secure link exists or the security level that the direct link supports is inadequate. To send a message through an indirect path requires the sender agent have full knowledge of all intermediate sites and indicate these sites explicitly in the message. Because of its complexity, more than two intermediate platforms are not feasible in practice.

We give an example in figure 3.3.2. It shows that there are three links between platform A and Z. When a message requires high level of security, the route A-C-E-Z is the only one that is qualified. When a message requires medium level security, route A-D-F-Z is the right one, however the route A-C-E-Z is also eligible. But when only low level is

required for a message, all three routes are qualified and apparently the best route is A-Z. The choice of other two routes will make the message much more complex.

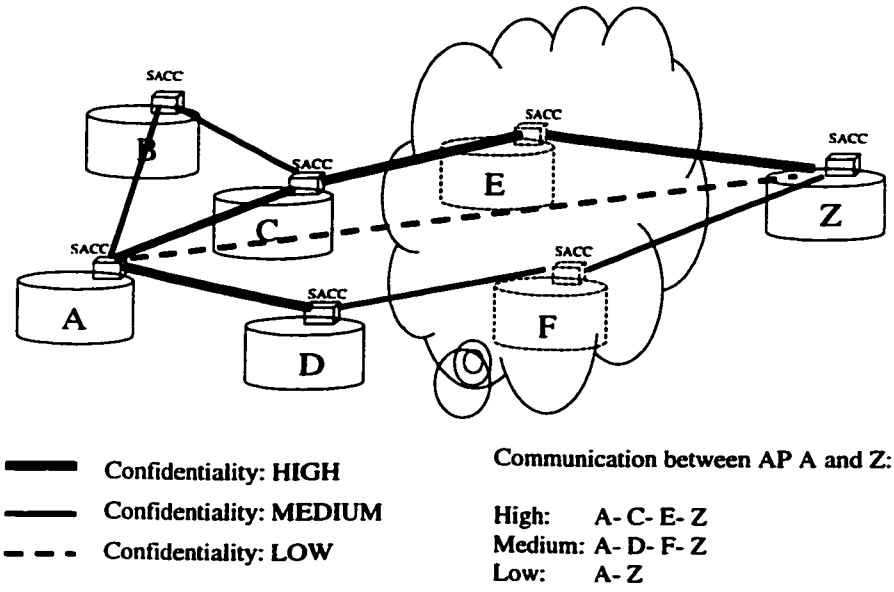

Figure 3.3.2 Security Communication Links Network

3.3.3 Mutual Platform Authentication

In our system, the primary goal of mutual platform authentication is to verify each platform's identity, and to establish the authenticity of platforms. Platform authentication is a mandatory process and must be performed before the secure communication link can be provided to agents. If the authentication process with a platform fails, then any message with security requirements cannot be forwarded to that platform directly, and the security layer will return a "fail" message to the sender agent.

Our authentication protocol is based on the public key cryptography as well as a commonly trusted third party called the Certificate Authority (CA). In this protocol, each legitimate platform must have its own private key and public key pair. The CA signs and issues certificates to support both platforms, which include platforms' names and public keys. We acknowledge that this protocol requires significantly more computation power than those based on secret key systems.

Since the certificates are signed by the CA's private key, they may be distributed safely without being tampered with. In our authentication protocol, the CA does not play an active role. Rather than demanding the partner's certificate from the CA, each platform gets the certificate directly from its partner. Note that, unlike most authentication protocols, the process of sharing the symmetric key has been moved from the authentication protocol into the negotiation protocol. The reason for this is simple: only when two parties agree on the same cryptographic options and security level they can begin to exchange proper keys. Our authentication protocol is illustrated in the following figure 3.3.3.

(1) Platform A initiates the process with a "Hello" message to Platform B.

(2) Platform B sends back its certificate ($Cert_B$) to platform A, including B's name (B) and public key (PK_B), signed by the CA's private key (PR_C), and an encrypted timestamp $E(T_B)PR_B$ encrypted by its private key (PR_B).

(3) Platform A verifies CA's signature to confirm that the key it received is actually Platform B's public key. Then it decrypts the timestamp with the newly derived PK_B. Once confirmed, it sends its certificate ($Cert_A$) to Platform B, including A's name (A) and

public key (PK_A), also signed by the CA's private key, and a newly encrypted timestamp $E(T_A)PR_A$.

(4) Platform B verifies CA's signature to confirm that the key it received is actually Platform A's public key. Then it decrypts the timestamp T_A with the newly derived PK_A to make sure that it is from Platform A. Platform B then sends a reply message indicating success and giving the negotiation protocol that is to be followed.

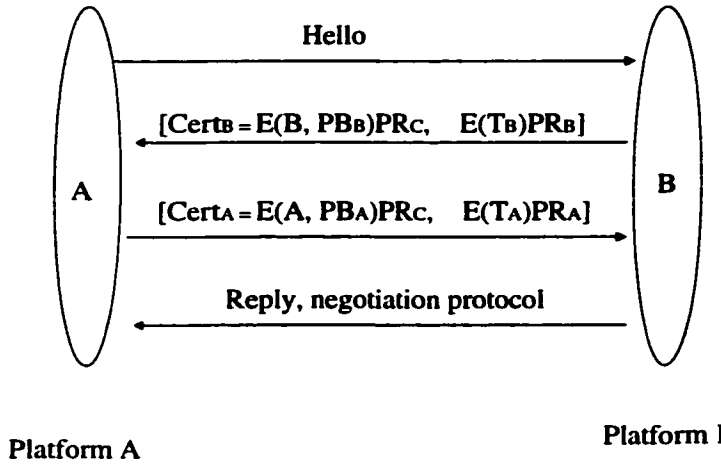

Figure 3.3.3 Authentication Protocol

As we can see, this authentication protocol is mainly based on the exchange and verification of certificates. However, in order to get the correct public key of a platform and verify the platform's claim, it is not enough to send the certificate only, the platform should also prove that this certificate it sent is not stolen from others. Here an additional timestamp is introduced because only the original platform has its private key and can provide the proper encrypted timestamp. This protocol is also resistant to the man-in-middle attack.

If its partner platform is successfully authenticated, the platform will start the next step known as the bilateral negotiation. Otherwise, this partner platform is marked "unknown" and the level of this secure link is marked "none", showing that the no direct secure communication service is available from current platform to this platform.

However a potential problem exists in this protocol, which is mainly because of the fact that the platforms get their partners' certificates without contacting the CA. Although this helps greatly to relieve the traffic bottlenecks of CA and simplifies the protocol steps, it also brings about the problem of certificate updating. Suppose in a situation one platform performs maliciously and the CA revokes its certificate at once, but other platforms cannot be informed of this immediately, therefore this malicious platform is still able to assault another victim with its old certificate.

3.3.4 Bilateral Negotiation

Bilateral negotiation between two platforms occurs when the mutual platform authentication is successfully finished. Because they work mostly in a heterogeneous environment, different FIPA-compliant platforms cannot be assumed to use centrally designed cryptography strategies. Instead, each platform uses a strategy that provides the highest possible security configurations for itself regardless of the configurations of other agent platforms. Therefore, different platforms may have a diversity of cryptographic algorithms and may support different security levels. However, in order to understand

each other's cipher message, two platforms must agree on the cryptographic options first. This bilateral negotiation allows the platform to decide which option is to be selected in respect to its partner's cryptographic capabilities. All cryptographic algorithms available in our system are given later in chapter 4. Note that only the bilateral negotiation is considered in our system because the trust network of different platforms is set up gradually by pairs of point-to-point secure connection. More sophisticated negotiation protocols involving multi-parties are beyond our system's scope.

There are two types of synchronous negotiation protocols supported in our system: a preemptive negotiation protocol and a reactive negotiation protocol.

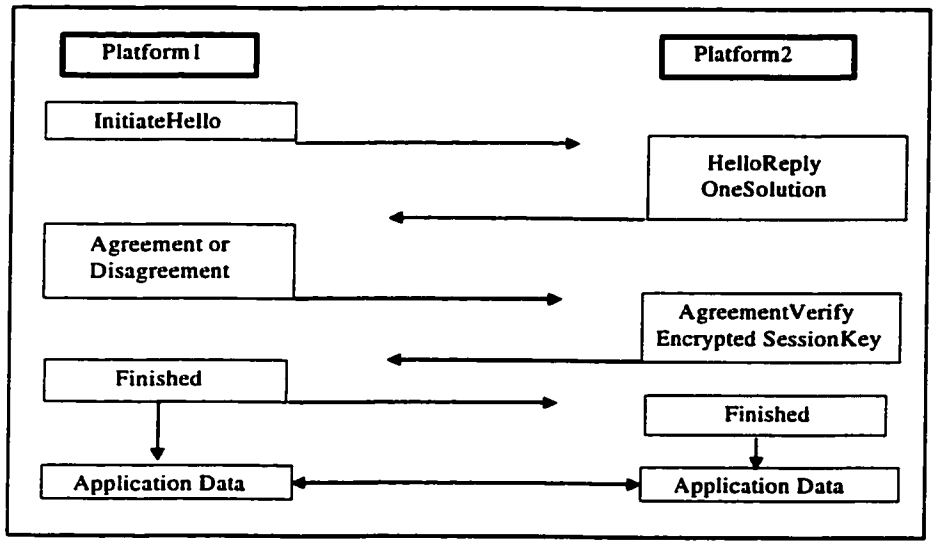

Figure 3.3.4 Preemptive negotiation protocol

Preemptive negotiation protocol: the preemptive negotiation protocol is simple but efficient. In this protocol, the receiver can decide either to agree or not to agree to the sender's cryptographic capabilities and suggestions. While efficient, the chance of unnecessary failure is rather high in this protocol, e.g. the sender might have a second cryptographic choice which matches the receiver, but no chance to show it to the partner platform. Figure 3.3.4 illustrates this protocol.

Reactive negotiation protocol: the reactive negotiation protocol allows the sender's capabilities are transferred to the receiver, with the receiver deciding either to accept them or respond with its own suggestions. This protocol is more likely to be successful because it allows both sides to make suggestions and compromise with each other. However, it may take several rounds before they can agree, resulting a greater cost in time. One round of the protocol is shown in figure 3.3.5.

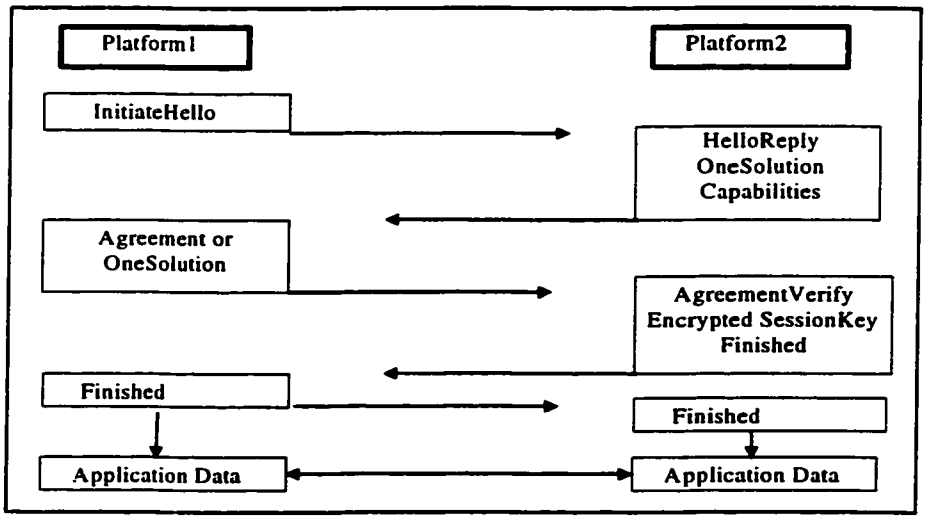

Figure 3.3.5 Reactive negotiation protocol

Platforms can statically choose one or other of these two types of protocols in respect to their focus on either the success rate or the efficiency level. However, both platforms will not "negotiate" on the type of negotiation protocol again, instead, only one platform (the sender platform) can make the decision and inform the other platform about the exact negotiation protocol to follow. This happens when the verification of both sides is successfully finished. Because the negotiation process (and therefore the authentication process) only happens when a platform is initiated or new platforms are added, the cryptographic algorithms for each neighboring platform are static during a platform's lifetime unless it restarts.

When this stage is finished successfully, the platform records the cryptographic options of each authenticated "adjacent" site in corresponding cipher modules. The cipher module consists of a symmetric cipher object, an asymmetric cipher object, a secret key, a digital signature object, and a reference to the keystore, which includes all public keys available and the current platform's private key. However, if any error occurs during this protocol, the security level of the link is marked "none", showing that no secure communication service is available to the platform.

3.3.5 Message Forwarding

In FIPA-OS, ACC is responsible for the message passing for both intra- and inter-platform communications, including the message forwarding and message routing.

However, in order to use the secure communication service provided by the security layer, an agent should send its outgoing messages directly to the SACC instead of the ACC of current platform. Afterwards, the SACC is responsible for forwarding these messages to the destination agent platform where the receiver agent is located. In this manner, SACC is able to handle a message before it is sent to the insecure outside network.

In format, the message sent to the security layer is slightly different from the ordinary ACL message that is sent via ACC. The difference lies in several types of information: the parameters of security requirements and real sender and receiver agent address. The additional information is inserted into an ordinary ACL message in format of name-value pairs, similar to the "envelope" parameter suggested in FIPA98. FIPA98 states that: "An envelope is a key-value tuple that contains message delivery and encoding information. It is included in the transport-message, and includes elements such as the sender and receiver(s) transport-descriptions. It also contains the encoding-representation for the message and optionally, other message information such as validation and security data, or additional routing data. A concrete instantiation of envelope is a mandatory element of every concrete instantiation of the abstract architecture." To be compliant with current FIPA-OS, we encoded them into the "content" tuple of an ACL message. Besides, some other tuples are added inside "content" in order to distinguish agent's messages from the system messages which are produced by the SACC (like in the initiation of platforms). For example, the value of tuple "type" indicates the type of a message, which can be an authentication message, a negotiation message or an agent's message.

Similar to the idea in FIPA98, SACC send the message as an agent's request. If SACC cannot satisfy the required security level, it refuses to send the message and in the meantime inform the sender agent, but cannot downgrade the security level an agent requires. An example message from an agent called "john" to the SACC of its platform called "somewhere" is given below:

```
(request
     :sender          john@iiop://somewhere.com:50/acc
     :receiver        sacc@iiop://somewhere.com:50/acc
     :ontology        fipa-agent-management
     :language        SL1
     :protocol        fipa-request
     :content
          (:real_sender john@iiop://somewhere.com:50/acc
           :real_receiver peter@iiop://agentland.com:50/acc
           :type           message
           :plain_text  ("I'm going to visit you tomorrow
                     afternoon.    Please wait me at the
                     gate.")
           :security_level low
           :signature
                     ("lk32+q=lFgkFdjJeWr98o42=12]sAopDfGrD"
                     )
          ...))
```

However, message forwarding in SACC is valid only when a secure link from the sender platform to the destination platform has been set up (either a direct link or an indirect one), which means that SACCs in both platforms are authenticated and trustworthy. This also means that a cipher model has already been prepared, which is useful in a SACC's message processing. If no secure channel exists, SACC refuses to forward the outgoing message, and at the same time sends a failure message back to the agent explaining it.

The sender agent may either lower the secure level, send it directly through the ACC, or even simply discard the message. Similarly for an incoming message, SACC forwards the message to the receiver agent as long as the secure link exists, but refuses to do so and sends "inform" message to the sender if not. The message below is a part of the previous example. It is sent from the SACC of the platform called "somewhere" to the SACC of a platform called "agentland".

```
(request
      :sender          sacc@iiop://somewhere.com:50/acc
      :receiver        sacc@iiop://agentland.com:50/acc
      :ontology        fipa-agent-management
      :language        SL1
      :protocol        fipa-request
      :content
          (:real_sender john@iiop://somewhere.com:50/acc
           :real_receiver peter@iiop://agentland.com:50/acc
           :type              message
           :cipher_text
                     (W6o68VEL+0SnuWMNgihNkfZi6yqwW
                       sdoe0e90SLIE38skdfUalskdjflsGer+podoqE
                       09Ew90hWoFD79U9v2R03249+02)
           :signature("lk32+q=lFgkFdjJeWr98o42=12]sAopDfGrD"
     )
           :txt_sig
                      (K6NpQyRfs/jwViQE7p/zmtDR0eFLxMeQPOwXr7
                      3L4433aIex1TenuhoSS8Dl+Hl03owEQADAYaqlM
                      TyrKAmomSsmY0sKSLzclXFg")

           ...))
```

In the destination platform of the message, when SACC receives the incoming messages, it will firstly check if the real receiver agent is in this platform and if the security module exists for the sender platform. If any answer is negative, the SACC will simply send a refuse message back to the real sender. If both answers are positive, then the SACC will

decrypt the message and forward it to the real receiver agent. Below is the example message which is sent from the SACC to the real receiver agent called "peter", in the receiver platform called "agentland".

```
(request
      :sender          sacc@iiop://agentland.com:50/acc
      :receiver        peter@iiop://agentland.com:50/acc
      :ontology        fipa-agent-management
      :language        SL1
      :protocol        fipa-request
      :content
          (:real_sender john@iiop://somewhere.com:50/acc
           :real_receiver peter@iiop://agentland.com:50/acc
           :type            message
           :plain_text  ("I'm going to visit you tomorrow
                      afternoon.    Please wait me at the
                      gate.")
           :security_level low
           :signature("1k32+q=lFgkFdjJeWr98o42=12]sAopDfGrD"
                      )
            ...))
```

3.5 Secure Execution Environment Service

The secure execution environment service is another security service provided by the security layer. Its purpose is to prevent unauthorized access of agents to some special platform resources or agents' services. This authorization mechanism of an agent platform provides a safe binding between the agent's code and the local environment. Therefore, the visiting agent is able to access the resources it needs but cannot breach system security and get unlimited access rights. The major idea in this service comes from the access control mechanism that is briefly reviewed in the following section 3.5.1.

In the trust model, we make the assumption that all agents signed by legitimate users will never be malicious and attack others deliberately. It may seems unnecessary in the secure execution environment service to introduce the access control mechanism in addition to the verification of the identity of an agent and its owner. However, as we pointed out before, our design goal is an abstract framework which encompass a diversity of agent platforms, and each type of platforms is able to define and enforce its own security policies, either strict in some cases or non-strict in others. Therefore we introduce the agent authentication and authorization mechanism to map an agent's identity into a set of access right permissions at each platform, which enable each platform to define its own mapping rules — security policies.

3.5.1 Access Control Analysis

Access control is the primary mechanism to provide the service of sharing information safely. In order to ensure the access of limited users to the shared resources, it associates the shared data with certain means of protections. Access control specifies this association in terms of security policies and their enforcement in a system. Traditionally, most security policies or mechanisms of access control can be grouped into two basic categories: the discretionary access control and the mandatory access control.

In Discretionary Access Control (DAC), security policies are defined by the owner/creator of the protected data. The owner of a resource can specify the access rights

of other users or pass his/her access rights to other users. One simple example is the access control matrix mechanism. In this mechanism, the protection system is modeled as a set of subjects and a set of objects, the access rights of a subject to an object is listed in an access matrix, and each entry lists the access right of one subject to one object. The access matrix, which we can call the system state, can be changed by a set of commands, which represent protection related subject actions such as "create", "destroy", etc.

A more complex example of DAC is the take-grant mechanism. This graph-based mechanism describes the transfer of access rights in systems instead of the state of access rights. The protection system consists of a set of subjects, a set of objects and a set of generic rights, among which two special rights are called "take" and "grant". The "take" right means that if a subject has a take right on an object, then it can take any rights of this object. Similarly, if a subject has a grant right on an object, then it can share any of its rights to this object.

In Mandatory Access Control (MAC), mandatory access restrictions are defined centrally without the user's discretion. Generally both users and the resources are classified according to the level of confidentiality. The system defines strict rules regarding which level of user clearance is required for accessing the data of a specific resource.

One example of MAC is the lattice based information flow mechanism [44]. The lattice mechanism views the protection system as a set of subjects, objects and security classes. Each subject and object have a security class associated with them. A security class

controls which subject can access which objects and how they are controlled. An information-flow policy is defined by the mathematical structure called lattice. Subjects can view the contents of an object when their level of trust is higher than or equal to the level of the object and when their categories are a superset of the categories of the object. They can modify the contents of an object when their level of trust is lower or equal to the level of trust of the object and their categories are a subset of the categories of the object. Some other mechanisms such as the bell-lapadula mechanism [45], combine the access matrix model with the classification hierarchy.

In addition, we need to mention a new DAC mechanism in the context of an untrusted open network. This is the "trust management" initiated by Matt Blaze and his colleagues [31][32]. The main idea of this access control mechanism is the representative of access rights with signed certificates. Each user is represented by his/her (private) key. One cryptographic key can delegate some parts of its authority to another key with a certificate. The certificates may form a complex network that reflects the complex relationship among users. In this approach, certificates are used directly for authorization rather than simply for authentication.

In order to decentralize the authority and management operations, they describe a trust management engine "PolicyMaker". A trust management engine is a separate system component, which uses a general-purpose and application independent algorithm to check proofs of compliance. It decides whether the security policy is conformed based on three sets of input: request, sets of credentials, and sets of local policies. In this

mechanism, there is no trusted computing base. On the contrary, all the participants are assumed to be untrusted as in an open network.

Recently, some new approaches such as Role Based Access Control [33,34] are receiving more attention as promising alternatives to traditional MAC or DAC. In the RBAC mechanism, permissions are associated with roles instead of users, and users can be deemed as a group of roles. A subject can be assigned different roles, and a role must be authorized to some operations, thus roles are the associations between subjects (users) and operations. Because the security management in RBAC is close to the structure of a nature organization therefore simplifies the system, RBAC is receiving more attention especially in commercial applications.

3.5.2 Related work

Many actual agent systems construct their security protection policy following certain types of access control. In this section, we give a short introduction to these mechanisms first.

In the PLANET project [35], a protection domain constrains the resource and the access rights of incoming mobile objects, thus protecting the agent server and the mobile objects themselves from illegal access by other malicious attackers. Different mobile objects in the same protection domain share the same access rights, their interactions need no domain-switching and can be executed efficiently. So in this model, the minimal resource

control unit and the only unit is the protection domain. In this project, each protection domain has a distinct virtual address space, and each virtual address space is completely separated, because the hardware inhibits object accesses beyond address-space barriers.

In Ara [36], the mechanism of "Allowance to Limit Resource Access" is used. An allowance is a vector of access rights to various system resources such as files, CPU time memory or disk space. The elements of this vector are related to quantitative or qualitative resource access limits. Domains of protected resources are dynamically created in the form of places, which consists of a name and an admission function. The admission function receives the agent's name and authentication status, and its desired local allowance as input parameters, it returns either the local allowance to be imposed on this agent, or a denial of admission. Each place has its own security policy, therefore they can have different admission functions. There are local allowances in different places, and the core system ensures that an agent never oversteps its allowance. The agent itself carries a global allowance after its creation. However, the local allowance will never exceed its global allowance.

As we can see, the basic requirement in protect agent platform is that various resources of the host are protected from malicious agents, at the same time, legitimate agents are given access to these resources. The common solution for accomplishing is to establish separate isolated domains for each agent, and control all inter-domain access. Whether it is called "Protected Domain" in PLANET [35] or "Region" in ARA [36], the mobile agents are restricted to execute in a "sandbox" where it can damage only limited resources, as

suggested in JVM [37]. The access rights are assigned to agents dependent on each agent's identity (authorization), and ensuring that the restriction cannot be violated (enforcement).

More recently, an alternative approach called Proof-Carrying Code (PCC) [38] has been proposed by Necula et al. The key idea behind PCC is that the code producer provides an encoding of a proof that the code adheres to a safety policy defined by the code consumer. The proof is encoded in a form that can be transmitted digitally to the consumer and then quickly validated using a specific proof-checking process.

3.5.3 Illustration Scenario

Our security mechanism in the secure execution environment service can be viewed as a combination of both the mandatory access control and the discretionary access control mechanism. The system level security polices are defined and managed centrally in the cooperation of abstract components, while platform level security policies are made up of policies defined in the Policy Server, which is owned by each different platform.

Following the basic idea of authentication of agents and the authorization to them, our proposal towards a secure execution environment is central to a so-called "permission credential". Three major components are involved here — a Credential Granting Center (CGC) to authenticate the mobile agents and issue permission credentials to them, a Policy Server (PS) to examine the access rights that can be authorized to the agents, and

authenticators to verify the authenticity of permission credentials. A complete process, from the agent authentication of the agents to the authorization of their access is illustrated in figure 3.5.1.

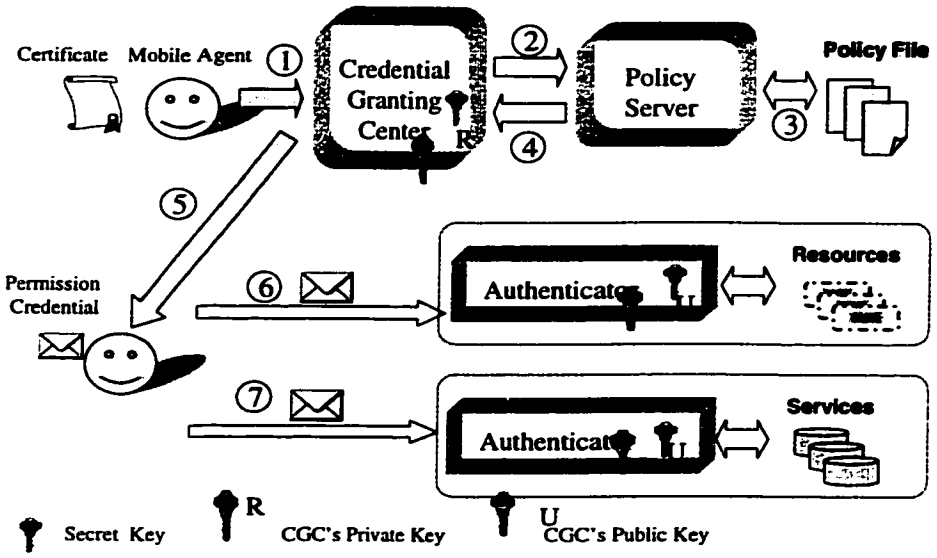

Figure 3.5.1 Secure Execution Environment

As the figure shows, when the agent arrives or starts, agent authentication is the very first step towards platform security. This process is mandatory for all incoming mobile agents. Agent authentication is performed by the Credential Granting Center (CGC) in the security layer of the platform. Agent candidates are required to provide valid proof of their identities when under authentication, that is, the digital certificates and signatures of the owner. Any fake proof such as an expired certificate or a false signature leads to the rejection of the agent's request for a credential.

Once it regards an agent as legitimate, the CGC needs to consult with the Policy Server for the authorizations that will be given. The Policy Server is an independent component that provides policy services such as adding, deleting and querying policies. It manages both the platform-level and application-level policies. However for platform security, we are concerned only with platform-level policies. Access rights for each possible agent are issued in the form of readable platform policy items, capable of being understood by the PS. The PS makes a decision based on pre-defined policy criteria and then sends its decision back to the CGC. On receiving the answer from the PS, a permission credential is created by the CGC, which includes all the permissions authorized to the agent, whether the agent applies for them or not. The agent is then free to apply those resource or services authorized to it in the permission credential, which acts as the identity of the mobile agent in the current platform. If the permission credential is verified as valid, an authenticator approves the agent's access requirement to the resource or the service.

Since the mobile agents are only one-hop in most cases, generating the permission credential allows the complex process of identity verification to be performed only once, no matter how many special resources and services an agent may need. The verification of credentials occurs whenever the resources are needed, thereby consuming a significantly lower amount of computation. By checking the signature of the CGC, and comparing the agent's fingerprints taken from the credential, the authenticator will decide if the credential is genuine, and if the agent presenting it is the original owner. If the answer to both answers is "yes", the authenticator will extract the access right from the credential and approve the access of the guarded resources or services.

3.5.4 Permission Credential

A permission credential is the valid proof of an agent's authorized access rights in an agent platform. In general terms, a permission credential consists of attributes associated with the agent's identity and its associated access rights. The identity-related attributes include the name of the agent, the owner of the agent, a timestamp, an expiry time, and most important, a hashcode of the agent as its unique "fingerprint". The access rights are written by the CGC in strings provided by the Policy Server.

The permission credential in our system is based on both public key and secret key encryption. The digital signature signed by the CGC's private key provides the integrity and authenticity necessary for a valid permission credential. Before being issued, credentials are encrypted by a secret key shared by the CGC and the authenticators in the current agent platform. The encryption of the whole credential permission ensures that its content can be understood by the authenticators only and that the content cannot be detected by other entities inside the platform. Once a credential is issued, it can be repeatedly used until it expires. Because the credential is encrypted by the CGC, it is safe to allow the agent to carry the credential to the verification spots without worrying that the agent might change it. And because the secret key shared between the CGC and the authenticators is platform specific, a credential is only valid in the platform where it is issued. If an agent tries to use its credential outside the platform, the examining authenticator will simply reject it because it cannot be read properly. An example of the

permission credential and its method of use in the ACL message is illustrated in the following figure 3.5.2.

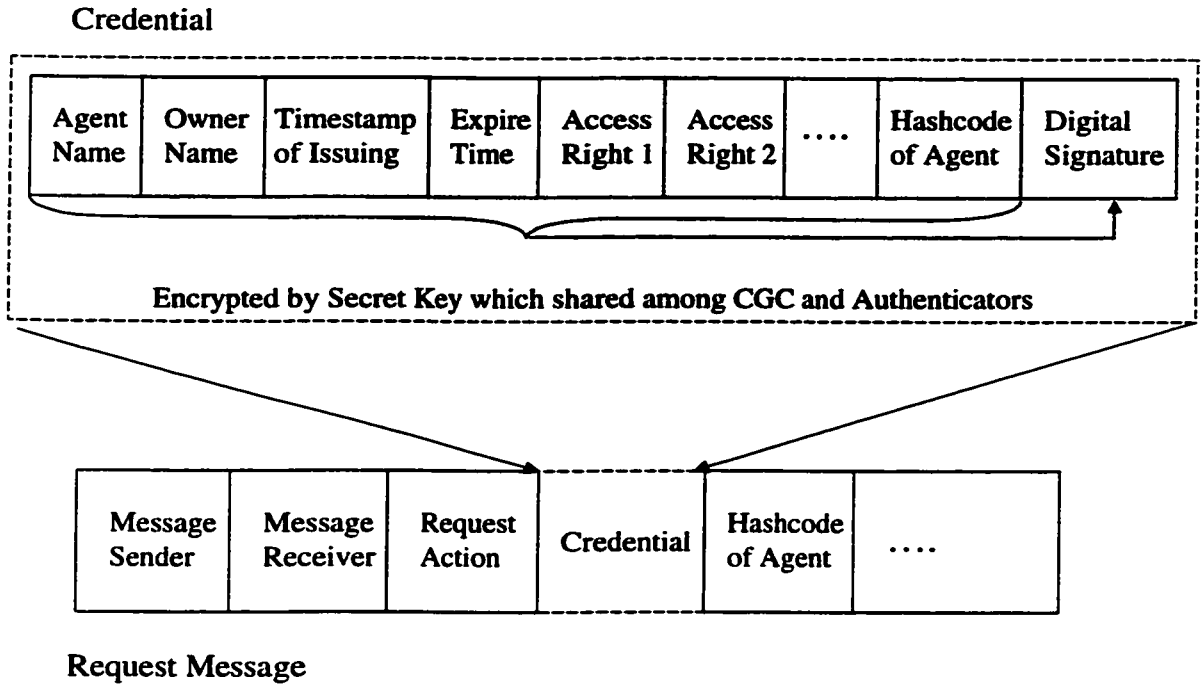

Figure 3.5.2 Permission Credential and agent's request message

For two reasons, the design of the permission credential ensures safety in distribution and in exclusivity to its owner agent. First, the design prevents the permission credential from being stolen and used by the agents of other users. The encrypted attribute of an agent's hashcode acts as the agent's fingerprint; only the original agent can provide the same hashcode and therefore pass the verifications of the authenticators. Any agent who extracts a permission credential from another agent and pretends to be that agent cannot provide the same hashcode. The agent will therefore be denied access to the resources and services recorded in the permission credential. Second, a permission credential

ensures that an agent cannot forge a credential or add more access rights by itself. This assurance comes from the encryption of the credential with the secret key shared only by the CGC and the Authenticators, and the signature of the CGC. Even an agent with a legitimate permission credential cannot know the credential's content, nor have the chance to forge it. In other words, they cannot add any unauthorized access rights by themselves.

CHAPTER 4

IMPLEMENTATION

In this chapter, we describe the technical implementation of the two-layer architecture which is based on the FIPA-OS agent platform. At the beginning of this chapter, we give a short introduction to the Java security model and the Java cryptographic architecture, which is the implementation basis in our system.

4.1 Java Security

The Java programming language with its security model [37] provides a high level of security. Because FIPA-OS is a Java-based platform, our implementation is compliant with the Java security architecture and the Java cryptography architecture, using their security APIs.

4.1.1 Java Security Architectures (JSA)

In order to provide a restricted environment for untrusted code from the open network, the Java platform provides a security model called the "sandbox" model. In the original sandbox model in JDK1.0, local codes are granted full access to vital system resources, such as the file system, while the remote code (an applet) can access only the limited

resources provided inside the sandbox. In this model, a security manager is responsible for determining which resource accesses are allowed. An improvement was introduced in the later sandbox model in JDK 1.1 over that in JDK1.0. That is, a digitally signed applet is treated like local code, with full access to resources, but unsigned applets are still run in the sandbox. Usually signed applets are delivered in signed JAR files.

The sandbox model is greatly modified in the new version JDK 1.2. In JDK 1.2, all local and remote codes are treated equally. Before they can be executed in the virtual machine, they are subjected to a security policy, which consists of a set of permissions. Each permission specifies an access to a particular resource, such as read and write access to a specified file or directory, or connect access to a given host and port. By default, applications run unrestrictedly as in the previous model, however it can be subject to a security policy when needed.

The runtime system maintains several individual domains, which acts as a sandbox, so applets can still be run in a restricted environment. The access rights in a domain are between a real sandbox and full access. After the examination of the security policy, each code (local or remote) will be assigned an environment to execute, it may be a sandbox, a domain or the full access. The new security architecture in JDK 1.2 is illustrated in the following figure 4.1.1.

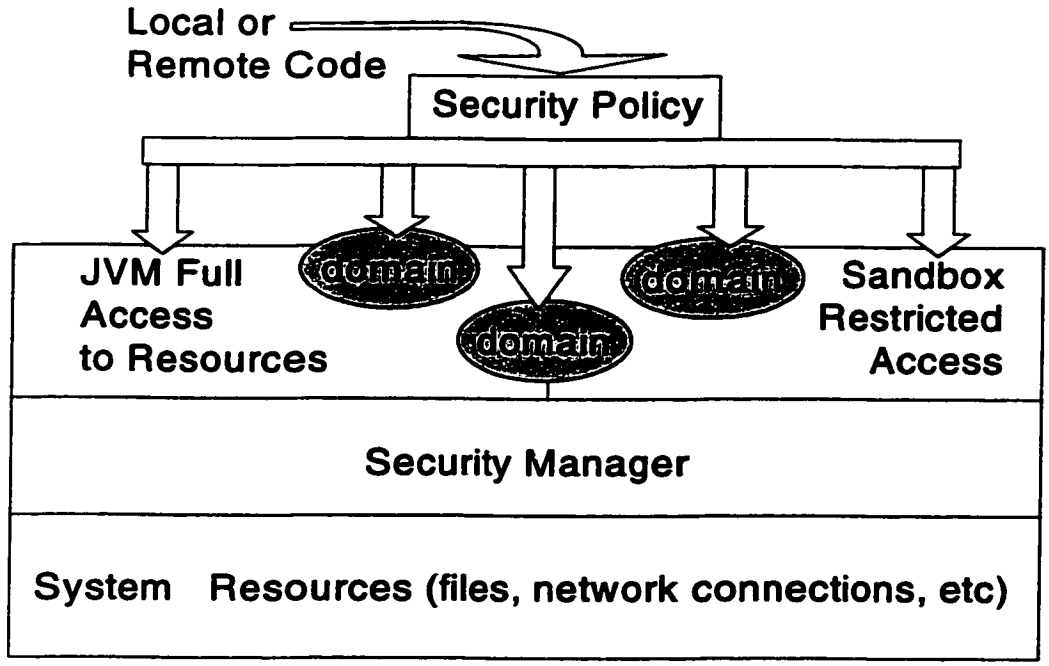

Figure 4.1.1 Java Security Architecture in JDK 1.2 [37]

4.1.2 Java Cryptography Architectures (JCA) and Java Cryptographic Extension (JCE)

Java Cryptography Architecture (JCA) was introduced in the first release of the JDK Security API in JDK 1.1. This is the framework for accessing and developing cryptographic functionality in the Java platform. In this architecture, different Cryptographic Service Providers (CSP) provide specific sets of packages, which supplies a subset implementation of the JDK API, and these implementations from different

providers are integrated seamlessly. Therefore, the JCA includes the provider architecture for multiple and interoperable cryptography implementations, and the Service Provider Interface (SPI) layer represents methods that must be implemented by all cryptographic service providers. Furthermore, JCA includes some implementations such as the message digest and the digital certificate in addition to the interfaces. However, these implementations are incomplete because of the restriction of US export regulations. The following figure 4.1.2 illustrates the JCA architecture.

As a result, JCA enables applications to use cryptographic concepts such as digital signatures and message digests without concern for the implementations or even the algorithms being used to implement those concepts. In this architecture, various implementations can work with each other, which can be described as the implementation interoperability, and the application is able to call cryptographic algorithms by names. For example, in our system, a platform is able to provide security services to the agents without their concern on the detail of cryptography algorithms. There is no doubt that the algorithm independence and the implementation independence extricate agents from the detail of cryptography, greatly simplifying the system.

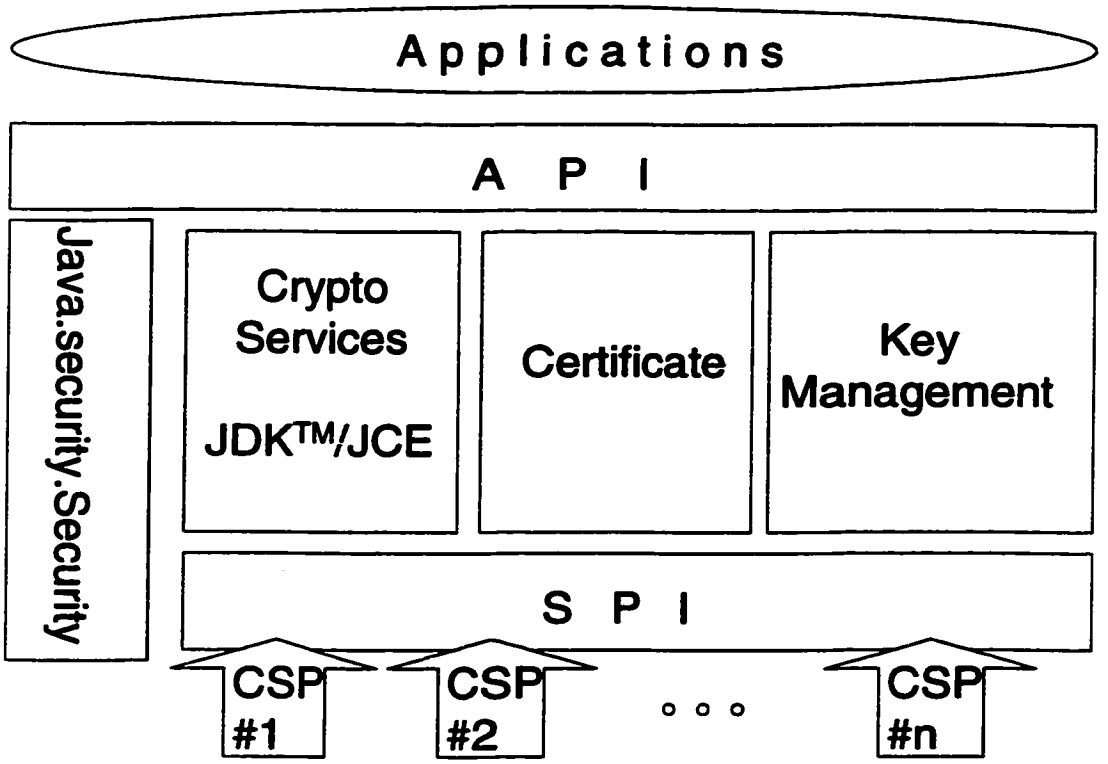

Figure 4.1.2 Java Cryptographic Architecture [37]

The Java Cryptography Extension (JCE) is released separately as an extension to the JDK and therefore JCA also. JCE extends the JDK in including APIs and implementations for encryption, key exchange, and message authentication code (MAC), because these cryptographic services were subject to U.S. export control regulations. JCE supports for encryption including symmetric, asymmetric, block, and stream ciphers, and also supports secure streams and sealed objects. Together the JCE and the cryptography aspects of the JDK (JCA) provide a complete, platform-independent cryptography API.

Currently, the recently released version of JCE 1.2.1 is available. In the design of JCE 1.2.1, other qualified cryptography libraries can be plugged in as service providers, and new algorithms can be added seamlessly. Unlike JCE 1.2, JCE 1.2.1 is exportable outside the U.S. and Canada.

4.1.3 Cryptix™ JCE Package and Our Supported Algorithms

In addition to JCA and JCE, we also use the Cryptix™ JCE [39] package as one of the major implementation sources of all algorithms used in our system. Cryptix™ JCE is a class library that contains a suite of cryptographic classes for Java and completely compatible with Sun's implementation. It implements the JCE, and also includes its own provider, including many popular cryptographic algorithms. Cryptix™ is originally published by Systemics Ltd. It is now a freely released product that can be downloaded from their web site internationally under their liberal license. For its features, Cryptix™ JCE supports almost all the popular algorithms such as:

Ciphers:	Blowfish, CAST5, DES, IDEA, MARS, RC2, RC4, RC6, Rijndael, Serpent, SKIPJACK, Square, TripleDES, Twofish
KeyAgreements :	Diffie-Hellman
Modes :	CBC, CFB-(8, 16, 24, ..., blocksize), ECB, OFB-(blocksize), openpgpCFB

Hashes : MD2, MD4, MD5, RIPEMD-128, RIPEMD-160, SHA-0, SHA-1, Tiger

MACs : HMAC-MD2, HMAC-MD4, HMAC-MD5, HMAC-RIPEMD-128, HMAC-RIPEMD-160, HMAC-SHA-0, HMAC-SHA-1, HMAC-Tiger

Signatures: RawDSA, RSA

Asymmetric ciphers: RSA/PKCS#1

To simply the system, we choose only some of those algorithms and integrate them into our system. To satisfy the requirement of confidentiality, authentication, integrity and non-repudiation, we support symmetric encryption, asymmetric encryption, message digest and signature respectively. An agent or a platform is able to choose a suitable algorithm from our supported algorithms that are listed in table 4.2.1.

Cryptography Type	**Algorithms**
Symmetric Ciphers	DES , IDEA, RC2, RC4, DES-EDE3
public-key Ciphers	RSA, DSA
Message Digests	MD2, MD4, MD5, SHA-1, SHA-0
Signature Schemes	MD2 & RSA & PKCS#1

Table 4.2.1 Cryptographic Algorithms Supported

4.2 System Environment and Configuration

Currently, our system has three enabled locations, each of which is represented by a personal computer, namely "Polaris", "Mirl" and "Taraus". On each of these computers, the two–layer agent platform prototype (see chapter 3) has been installed and configured. In order to simplify the system, each of these three sites maintains its own certificate and all system profiles. The system has been implemented on windows environment, specifically Windows NT 4.0, using Java technology including Java Security Architecture and Java Cryptography Architecture. Our implementation is programmed in Jbuilder2 environment, and all our codes are tested to be compliant with JDK1.2.

The KeyStore class in JCA supplies well-defined interfaces to access and modify the information in it, therefore all our certificates and keys are stored in this class. The KeyStore class may have different implementation types, and each type defines its own storage and data format of the information. Here we use the default JCE implementation, which implements it as a file, using a proprietary KeyStore format named JKS. In our implementation, each agent platform owns a keystore object which performs as a repository of keys and certificates, it is initiated when the security layer starts up. This keystore object protects each private key with its individual password and also protects the integrity of the entire keystore with a password, which may be different from

previous ones. In this keystore object, keys are transparently represented by key specification interfaces.

Also, a configuration file called "SecurityParameters.txt" is introduced in order to record the security capabilities of the platform. It consists of the cryptographic option items, which is the name-value pair, i.e. "SymmAlgorithm_name = DES ". When the security layer initiates, this configuration file is loaded as a Properties class in Java, and it is used when SACC performs the negotiation or provides the secure communication services. It is also loaded into the CGC to help perform the task of the authentication of agents and the verification of credentials.

4.3 Platform Initiation

Strictly speaking, the platform initiation can be divided into two phases: the initiation of the basic FIPA-OS platform and the initiation of the security layer. After the initiation of FIPA-OS platform, major components such as the AMS, the DF and the ACC are started and ready to provide services. During the following phase, major components in security layer including the SACC and the CGC initiate themselves and are ready to provide security services.

When the security layer of a platform initiates, it sends "HELLO" messages to all of its logically neighboring platforms in sequence, attempting to start the authentication and negotiation process with them. The list of these "neighboring" platforms is the same as

that in the profile of the ACC. Usually each platform starts up at a random time slot, therefore all platforms are started sequentially. However, the possibility exists that two or more platforms initiate concurrently. In the following sections, we will discuss these two different scenarios.

4.3.1 Scenario of Sequence Initiation

A complete sequential initiation process is shown in figure 4.3.1. In our implementation schema, the security layer of an agent platform obtains the list of "neighboring" platforms from the profile of the ACC. Afterwards the security layer (specifically the SACC) sends out a "HELLO" message to each platform on the list, and at the same time starts a timer, waiting for the reply messages from this platform. During this period, a SACC will reject any invitation to start an authentication or negotiation process from other platforms. In other words, it sends negative reply to any "HELLO" messages received in this period. When a time-out event occurs or a negative reply is received from the platform for which it is waiting, the security layer will move on to the next platform in the list. Note that in figure 4.3.1, the SACC is responsible for sending messages according to the list, the SACCListener is responsible for accepting or replying to messages and the negotiator is responsible for the enforcement of protocols.

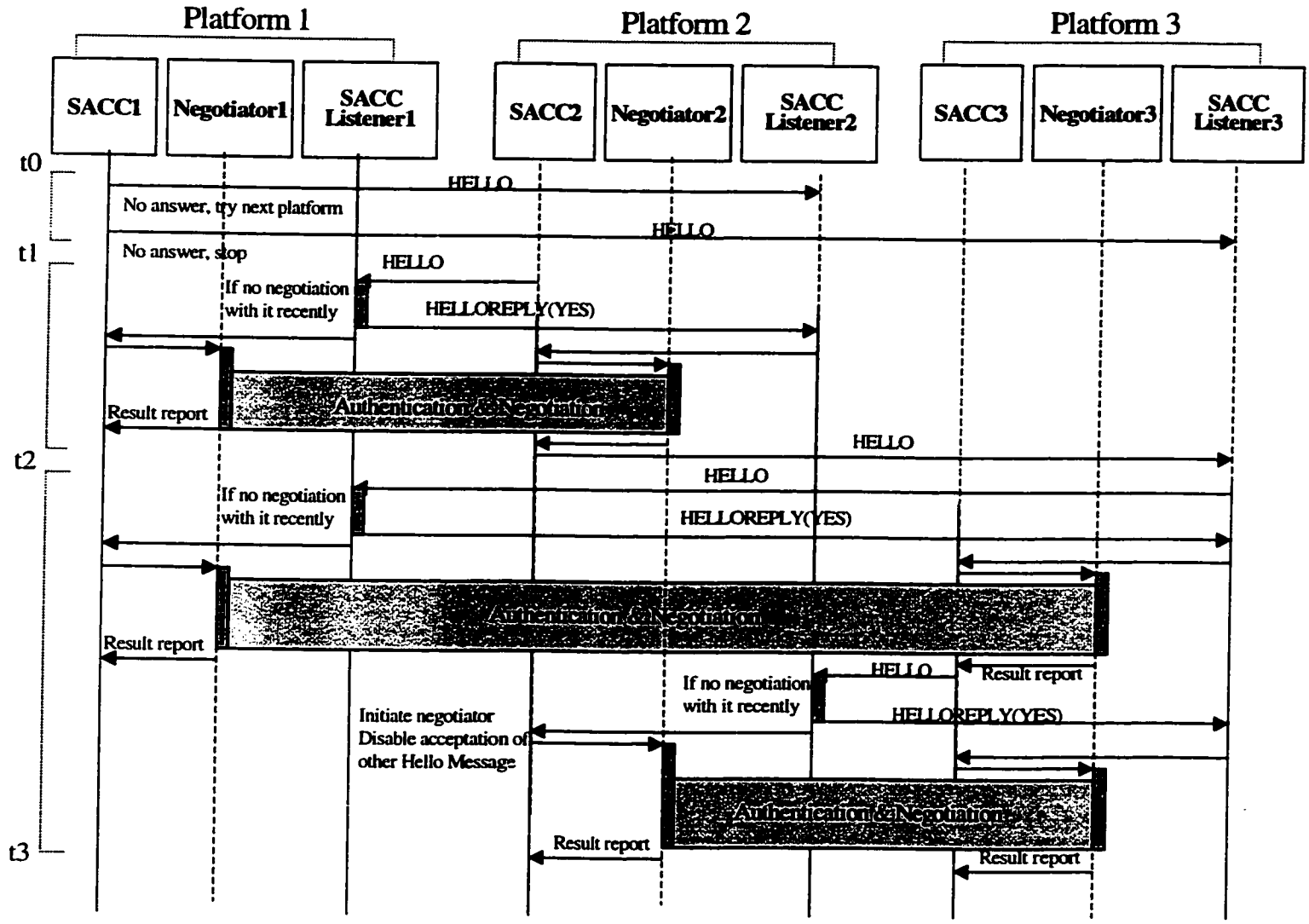

Figure 4.3.1 Scenario of Sequential Platform Initiation

4.3.2 Scenario of Concurrent Initiation

Although the possibility is fairly small, different platforms may be initiated concurrently. For example, the two security layers of different platforms may send "Hello" message to each other simultaneously. Or two platforms may try to contact with a third platform at

the same time. When messages collide, some special concurrent control mechanisms must be taken for the security layer to make decisions and therefore to finish the process successfully.

Suppose the security layer of one platform (Platform A) is expecting a positive reply message from a certain platform (Platform B), but instead, it receives a "HELLO" message from the same platform (Platform B). In this case, we say that a collision happens because both of them (Platform A and B) are trying to initiate the authentication protocol with the other. If the security layer simply discards this message as it discards all the other messages from a third platform, then these platforms will miss the chance to authenticate, negotiate and securely communicate with each other, because both sides give up. To avoid this failure, as shown in figure 4.3.2, each platform checks the timestamp parameter in the "HELLO" message with its own timestamp. The platform receives a more recent time discards the message and continues its wait, and the other SACC sends a reply message as a response.

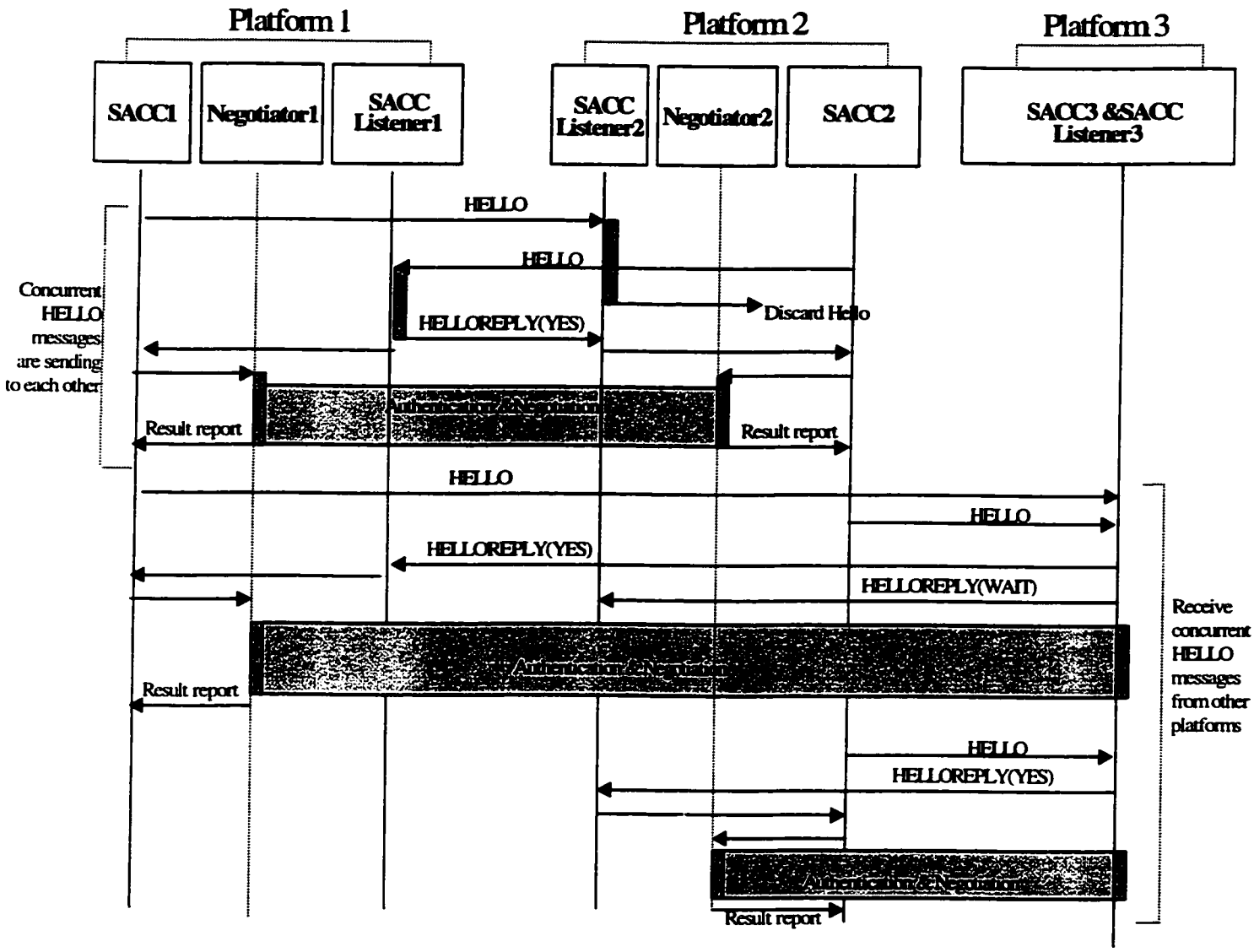

Figure 4.3.2 Example of Concurrent Platform Initiation

Note that there is no need for the clock synchronization between all platforms in order to provide the timestamp. Because it is almost impossible to keep all heterogeneous

platforms synchronized, the function of the timestamp is only a criterion to choose from two messages, enable one platform to accept and the other platform to discard. To do this we need to know the approximate transmission time of a message from one platform to another. Figure 4.3.3 shows that every platform calculates the same formula, uses its current time minus the summary of timestamp and the fixed time T. Since $T \cong T_t$, $(Ta+Tt)-(Tb+T) \cong (Ta-Tb)$; and $(Tb+Tt)-(Ta+T) \cong (Tb-Ta)$, therefore one platform will get a positive value and the other will get a negative value. The following example illustrates the basic idea.

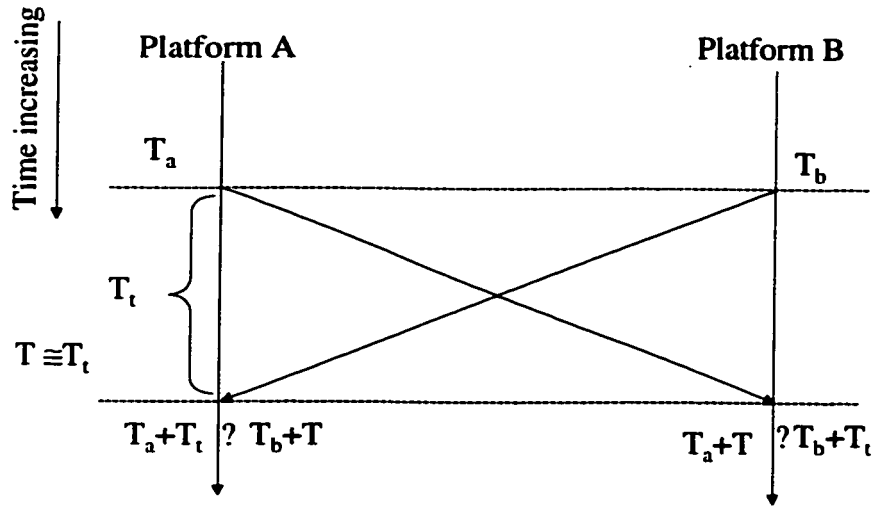

Figure 4.3.3 Time sequence of currency

Suppose that at one moment the system time in platform A is 9:30, whereas in platform B it is 9:40, and they both send the "HELLO" message to each other. If it costs

approximately 3 seconds to transmit a message between these two platforms, when platform B receives the "HELLO" message, its system time is 9:43, and the timestamp is 9:30. It will therefore compare the time 9:43 with the time 9:30 + 0:03, simply discarding this message and waiting for the response from platform A. However, in platform A, when the "HELLO" message arrives, the system time is 9:33 and the timestamp is 9:40, it will give up the effort to wait for the response from platform B, and instead sends a "HELLOREPLY" message to platform B immediately.

4.4 Tuple design

In order to be compliant with current FIPA-OS version, we have to encompass our special communication messages or encrypted messages in ordinary ACL messages that FIPA-OS supports. Therefore, on the sender's side, we put all those additional items we need into the "content" tag of an ordinary ACL message in format of a string. On the receiver's site, we decode this special string and restore it back to the items then extracting the values. For convenience, we designed a class called "Tuple" which is responsible for the encapsulation and de-capsulation of the content of a message. The main functions of the Tuple class are described as below:

```
public class Tuple extends Hashtable {
/* used by decode methods to keep track of a recursive tuple */
      private int bracketCounter = 0;
/* Create an empty tuple */
      public Tuple () {super();}
/* create a tuple from a given string, if the string doesn't follow the
format and can't be parsed correctly, an exception is thrown out */
      public Tuple (String aTupleString) throws
InstantiationException{}
```

```
/*    Add a new key-value pair to the tuple */
      public void add(String key, Object value)
      private boolean decode(String aTupleString)
      private boolean decode(StringTokenizer tokenizer)
/*convert a tuple to a string in the fixed format */
      public String toString ()
/*convert a tuple to a ACLstring in the fixed format */
      public String toACLString ()
```

With the help of Tuple, the "key", "value" pair in a ACL message can be translated into string and therefore transmitted in the "content" of the message. For example, one message might be as below:

```
(request
     :sender          sacc@iiop://agentworld.com:50/acc
     :receiver        sacc@iiop://agentland.com:50/acc
     :ontology        fipa-agent-management
     :language        SL1
     :protocol        fipa-request
     :content
          (:real_sender john@iiop://agentworld.com:50/acc
           :real_receiver peter@iiop://agentland.com:50/acc
           :type           message
           :plain_text  ("I'm going to visit you tomorrow
                     afternoon.    Please wait me at the
                     gate.")
           :security_level low
           :signature
                     ("1k32+q=1FgkFdjJeWr98o42=12]sAopDfGrD"
                     )
          ...))
```

After the processing of a tuple class, the real ACL message that is transmitted in the network is as follows:

```
(request
     :sender          sacc@iiop://agentworld.com:50/acc
     :receiver        sacc@iiop://agentland.com:50/acc
     :ontology        fipa-agent-management
     :language        SL1
     :protocol        fipa-request
```

```
     :content       "Tuple(:real_sender \"
john@iiop://agentworld.com:50/acc \" :real_receiver \"
peter@iiop://agentland.com:50/acc \" :type \" message \"
:cipher_text  \" W6o68VEL+0SnuWMNgihNkfZi6yqwW
sdoe0e90SLIE38skdfUalskdjflsGer+podoqE09Ew90hWoFD79U9v2R032
49+02\" :security_level \" low  :signature \"
lk32+q=lFgkFdjJeWr98o42=12]sAopDfGrD\" )"
)
```

4.5 Authentication and Negotiation Implementation Algorithm

The authentication and negotiation processes are implemented by the interactions of three components in the security layer: a SACC agent, a negotiator agent and a protocol object. An authentication protocol object is used in the authentication process and a negotiation protocol object is used in the negotiation process. Figure 4.5.1 illustrates how these three components interact and cooperate.

The process starts when one SACC agent sends a request message to the SACC on a remote platform. When the response is positive, the SACC generates a negotiator agent on the local platform, as well as a protocol object which defines the steps to be followed during the negotiation. On the remote site, after sending a confirmation message, the SACC also initiates a negotiator agent and a protocol object. Afterwards, the process is taken over by the negotiator agents on both sides. SACC agents are then free to perform other tasks such as preparing and forwarding the secure ACL message for other security channels. Usually, the protocol objects are in the form of a matched pair, one labeled "SENDER" and the other "RECEIVER".

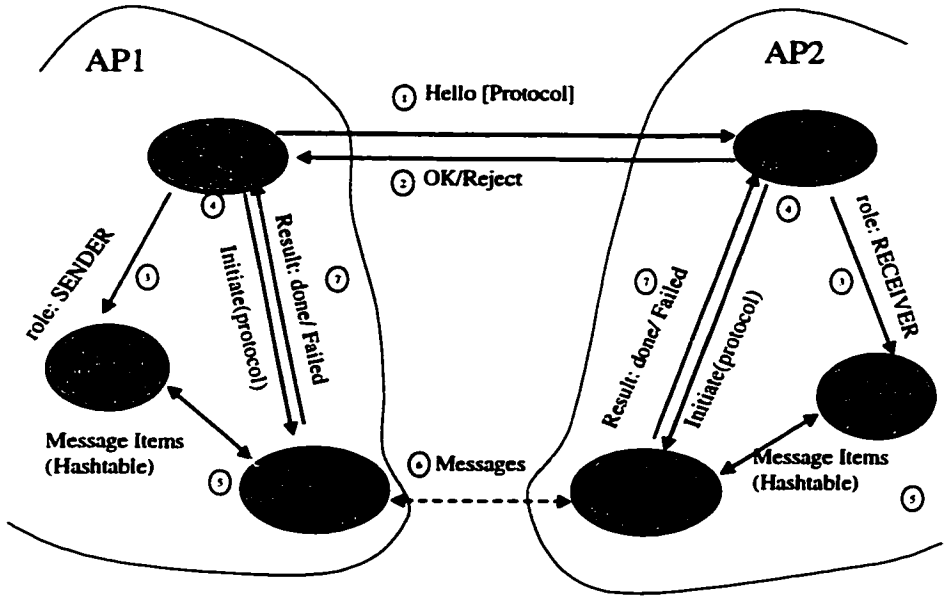

Figure 4.5.1 Cooperation of protocol object, negotiator agent and SACC agent

Once the process is finished, the negotiators on both sides will report the final results to the SACCs. If the result is positive, the negotiators also provide the necessary certificates, or the shared cryptographic options and secret keys. However, if an error occurs during either process, messages indicating the reason for the failure are reported to the SACC. For instance, the error TIME_OUT occurs when no reply is received in a limited time period. The error OUT_OF_ORDER occurs when the incoming message does not match the expected message. Other possible errors are all marked "WRONG_MESSAGE". These include the absence of specific content, unreadable messages or verification failure. (If the verification of the identity fails, then the authentication protocol fails).

Protocol objects are used as references in the communication between two negotiator agents, defining the appropriate actions based on the newly received message and the current states of the negotiators. As an abstract base class in implementation, a protocol object also provides uniform methods interface to negotiators while masking the complex detail of different protocol classes. Because we separate the authentication and negotiation process by implementing them into two protocol objects, it is more flexible for the upper application to choose and arrange according to their application-specific requirements. Each protocol object, whenever the relevant negotiator agent sends out a protocol message to its partner, or receives a correct reply message from peer negotiator agent, will upgrade its internal protocol state to keep up with the current change. Therefore, the two matched protocol objects can maintain the synchronization between each other. Besides, a protocol object can indicate the relevant negotiator agent on the outgoing message based on its internal protocol state.

The code segment of the negotiator:

The following code segment is the major processing logic of negotiator agent, which is created with an authentication request.

```
public void run(){
    try{
      while(!current_protocol.isFinished()){
       if ( !current_protocol.proceed()){

          System.out.println("The negotiation fails.");
          sendErrorResult("OUT_OF_ORDER");
          this.stop();
       }
      }

    }catch (Exception e){
       sendErrorResult("WRONG_MESSAGE");
```

```
        this.stop();
      }
      System.out.println("Negotiation success!");
      sendSuccessResult();
      this.stop();
    }
```

The sample code segment of authentication protocol (verifying the certificate):

```
public boolean proceed() {
  if (role == protocol.SENDER){
    switch (current_step){

    /* Sender's step 0 */
    case 0:
      hello(); /* send hello message */
      current_step++; /* */
      return true;

    /* Sender's step 1 */
    case 1:

       Tuple t = getNewlyArrivedMessage();
       if (t == null){
         System.out.println("ERROR: TIME_OUT");
         return false;
       }
       if ( !t.get("type").equals("AUTHENTICATION") ){
         System.out.println("ERROR: OUT_OF_ORDER");
         return false;
       }
       if ( !t.get("performative").equals("HELLOREPLY")){
         System.out.println("ERROR: OUT_OF_ORDER");
         return false;
       }
       if (t.get("Certificate") == null){
         System.out.println("ERROR: OUT_OF_ORDER");
         return false;
       }
       /* Decode the certificate and verify it*/
       EncodedCert = (String)t.get("Certificate");
       Cert = base64.decode(EncodedCert);
       cert = generateCertificate(Cert);
       if (!verifyCertificate(cert)){
         System.out.println("ERROR: WRONG_MESSAGE");
         return false;
       }

        /* send its own certificate and move to next step */
       sendCertificate();
       current_step++;
       return true;
```

```
case 2:
          ...

  }else if (role == protocol.RECEIVER){
    switch (current_step){
  case 0:
              ...
       case 1:
              ...
    }
      }
  return false;
}
```

4.6 Experiment Results

4.6.1 Samples of Experiment Results

Here are some samples of the screen snapshots when the system executes. Figure 4.6.1 shows the running state of one agent platform called "Telelearning" when it is initiated. The figure shows that the authentication and negotiation process with platform "Polaris" is successfully finished.

Figure 4.6.1 Platform Running Status

After the initiation of both platforms, an agent in platform "Polaris" called the "testagent" is trying to send a message (a piece of love verse of Shakespeare) to its peer testagent in platform "Telelearning". Below we give the screen snapshot of the running status of the testagent agent in sender platform— "Polaris".

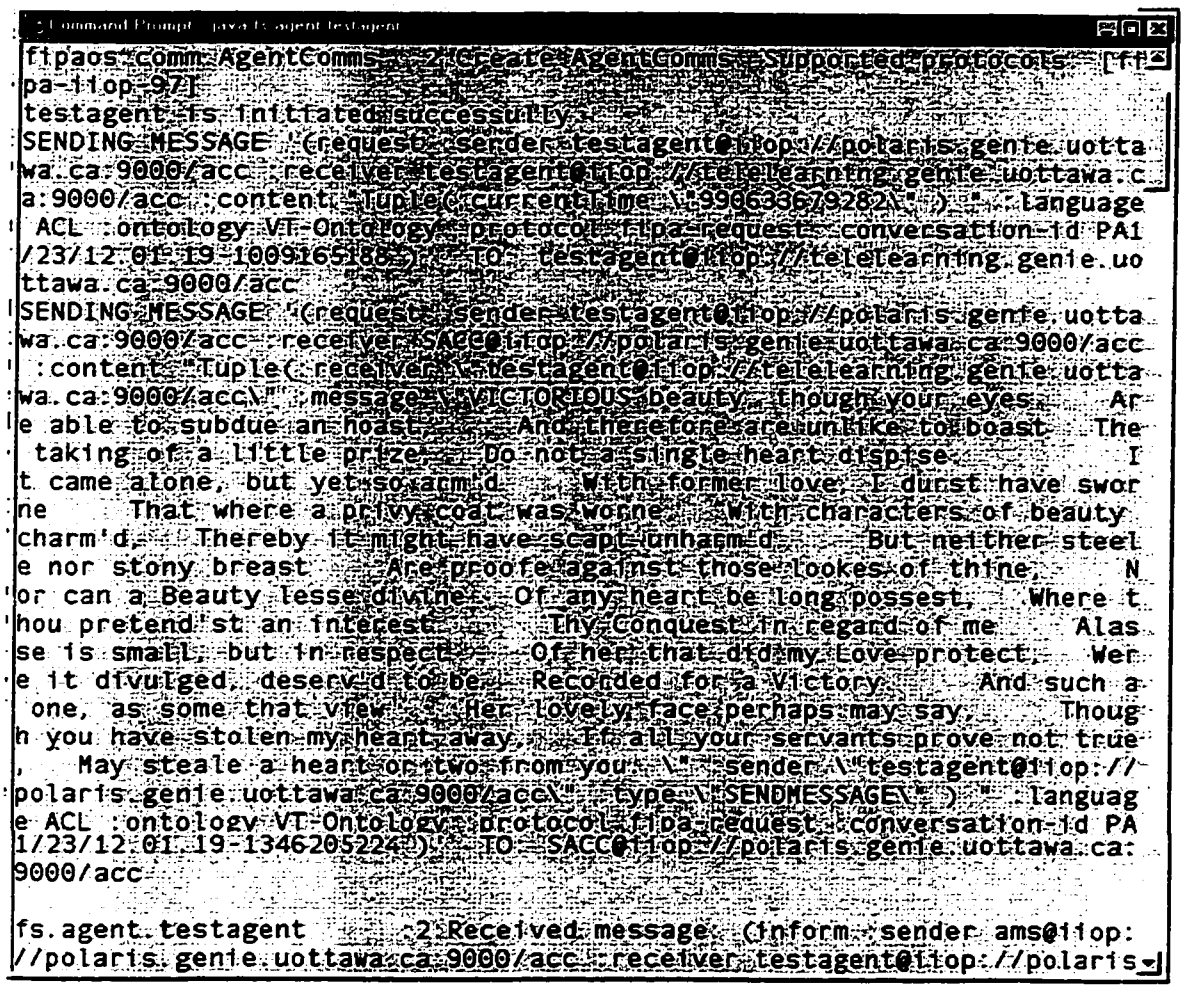

Figure 4.6.2 Running Status of testagent in the Sender Platform

As we explained in previous chapters, this message should be sent to the SACC of platform "Polaris" before being sent to the testagent in "Telelearning", where it can be encrypted and encoded. Figure 4.6.3 shows the window of the SACC in platform "Polaris". We can find from this figure that the encrypted and encoded message is being sent to the SACC of platform "Telelearning".

Figure 4.6.3 Running status of SACC in the Sender Platform

The running status of the receiver platform "Telelearning" is given in figure 4.6.4. The figure shows that the platform receives an encrypted message directing to the SACC of

current platform. When the SACC in "Telelearning" gets the encrypted message from platform "Polaris", it will be responsible for decrypting it and redirecting it to the testagent.

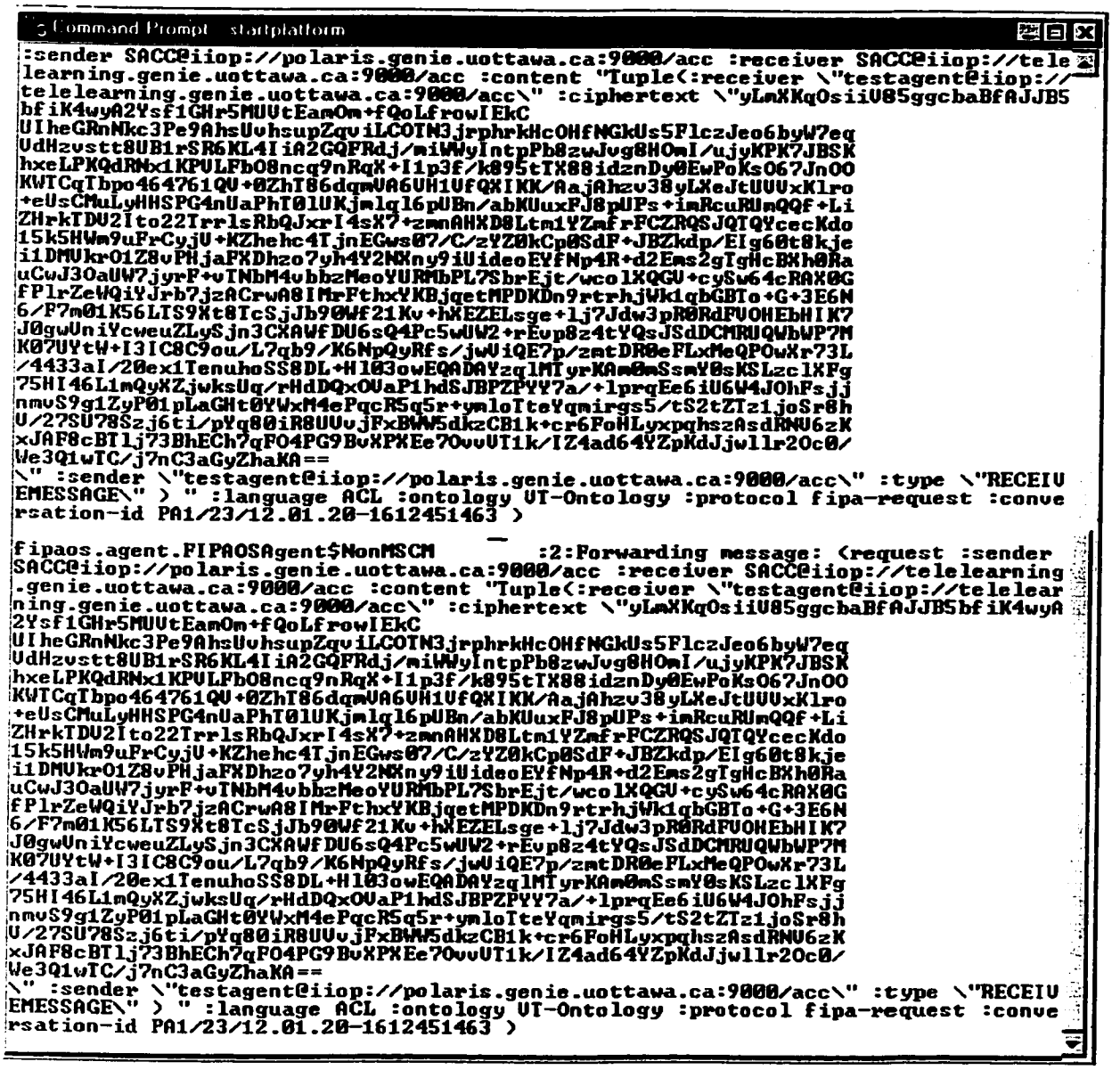

Figure 4.6.4 Running Status of the Receiver Platform

Finally, we give the snapshot screen of the receiver agent "testagent" in platform "Telelearning" in the figure 4.6.5. We can see that the message has already been decrypted, that is, the agent receives the original message that it can understand.

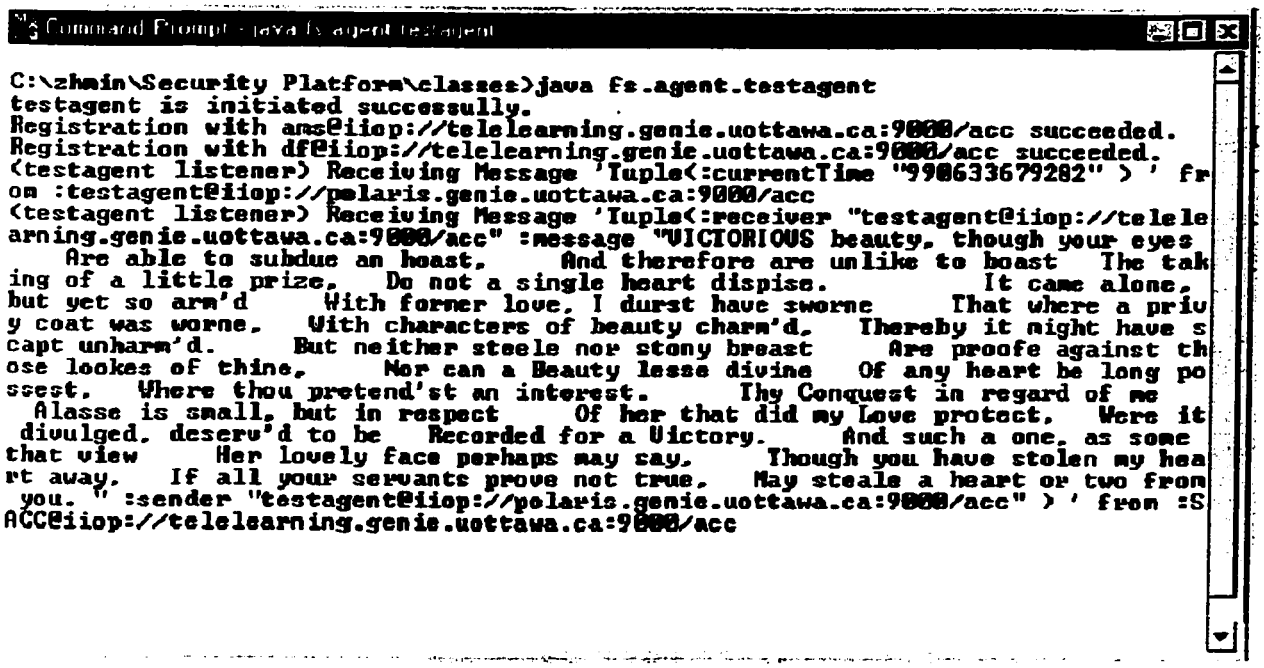

Figure 4.6.5 Running Status of the Receiver Agent

4.6.2 Results Evaluation

According to our implementation described in previous sections, we measured the overheads for the authentication protocol, negotiation protocol, and different level of de/encryption communication. We ran our tests under Windows NT and JDK1.2.

In the measurement of authentication protocol, the most time spend is on FIPA-OS communication, a corresponding small percentage on the verification of certificates (See

table 4.6.1). Besides, considerable time are spent on the encoding and decoding of certificates, which translate it from binary format into ASCII code that can be transmitted in the network.

Sub procedure	Time (ms)
Certificate Preparation (Encoding)	140
Certificate Preparation (Decoding)	80
Certificate Verification	123
Communication & others	4034
Total	4377

Table 4.6.1 Time Cost in Authentication Protocol

In the measurement of negotiation protocol, we found that the preemptive protocol is a slightly more efficient if it is successful (See table 4.6.2), which is more suitable when most platforms are homogeneous. In both preemptive protocol and reactive protocol, the most time spent is on the generation of the secret key, encrypting the secret key with the other's public key, and the decryption of the secret key. In reactive protocol, we sets a limit of three rounds. When several rounds are needed, it costs a little longer in communication negotiation. However, the time spent in finding a suitable option is considerably small in comparison with the following process of secret key preparation and exchange.

Protocol Type	Secret key Generation & Secret key De/Encryption	Communication & others	Rounds	Total Time (ms)
Preemptive protocol	26738	132	1	26870
Reactive protocol	27149	210	2	27359

Table 4.6.2 Time Cost in Negotiation Protocol

We measured the communication overhead with three different input messages. 1. An empty string message with a serialized length of 12 bytes. 2. A string object with a serialized length of 1k bytes. 3. A more complex object with a serialized length of 10k bytes. As expected, the communication time, including overhead introduced by enciphering and deciphering, is nearly linearly increased with the length of messages (see table 4.6.3). However, the overhead of communication is considerably limited, in comparision to the overhead in security preparation.

Messages	Time (ms)
Empty String (12 bytes)	491+(FIPA-OS message time)
Short Message (1k)	1272+(FIPA-OS message time)
Complex Message (10k)	9494+(FIPA-OS message time)

Table 4.6.3 Time Cost in Message Forwarding

CHAPTER 5

CONCLUSIONS

5.1 Summary

This thesis presents the design of a security-provided architecture which is compliant with FIPA specification, as well as its implementation on a specific agent platform called FIPA-OS. This architecture seeks to provide architectural solutions to security threats in FIPA standards. According to the attack model, we focus on protecting agents and its messages against attacks from network, and protecting a platform's resources or an agent's services against unauthorized access from other agents. We also present a hierarchical trust model based on a certificate authority — the trust basis of the whole system — in order to address other types of threats.

We describe a two-layer architecture of an agent platform, consisting of a basic agent platform and a security layer. This architecture provides the independence of security mechanisms and policies from specific agent platforms. In our design, the security layer in an agent platform ensures security by two types of security services, namely a secure agent communication service and a secure execution environment service.

To provide the secure communication service, platform authentication and platform negotiation protocols are introduced at the initiation of a platform's security layer. These protocols help two platforms to verify each other's identities and cryptographic capabilities, therefore setting up a secure communication channel between them. This secure channel enables the secure message to be forwarded in inter-platform communication.

We describe the setup of a secure communication network that consists of many point-to-point secure links. In this network, an agent platform may have no direct secure link to some platforms, but only those to its logically adjacent nodes. In the routing of a message, several secure paths may exist. Therefore an agent is able to choose an indirect path if the agent has full knowledge of all intermediate sites and indicate these sites explicitly in the message. We describe two possible scenarios in a platform's initiation, namely the sequential initiation and the concurrent initiation. Sequential initiation occurs when different platforms start at different time, and concurrent initiation indicates security layers of different platforms are started simultaneously.

To provide the secure execution environment service, we introduced the agent authentication and authorization mechanism. According to the trust model, local agents of an agent platform gained the trust of their home platform naturally. However mobile agents must have their identities authenticated before being allowed to register on the current platform. To be qualified, a mobile agent must come from a trustworthy platform and must have valid digital signature of a trustworthy user.

Some special resources of a platform or agent's services are individually protected in the security layer. Agents are permitted to access these resources or services only when they are authorized to do so by the policy server. As a result, their access rights are written in their permission credentials. An authenticator is designed to verify the validity of permission credential, which circulates in current platform as the proof of the agent's identity and its access rights to some special resources or services. The design and content of a permission credential ensures that it can prevent agent impersonation or get unauthorized access, therefore it is almost impossible for agents to use faked permission credentials or those belonging to someone else.

In short, the major contributions of this thesis can be concluded as follows: (1) a proposal of a two-layer architecture, which separates the security services from basic agent platform services and provides security services instead of mandatory security policies. (2) A new approach of the establishment of a secure communication network, which at the same time enables different platforms to have their choice of algorithms and security levels. (3) The design of the permission credential, which is the domestic proof for an agent's identity and is difficult to be abused by others.

5.2 Future Work

Since our proposal is towards an architectural solution for security concerns in FIPA specification, we are limited to provide a complete security schema because of the lack of

specific secure mechanisms which are related to implementation details of an agent platform. To be a more restricted system, each FIPA-compliant agent platform is allowed and required to develop its own security policies and mechanisms that meet their security requirements under this same architecture. For example, we defined the format of a permission certificate and the functions of an authenticator in our architecture. But it is up to the agent platform designers to decide how they are implemented, or if there should be additional policies related to it. Therefore, many other issues and areas must be studied before a complete secure agent platform can be fully developed, which follows this security architecture.

Further work to complete this architecture is much more challenging. One task is considering the possibility of malicious agents and platforms. In practice, it is reasonable to assume the owner of a mobile agent has a full knowledge of its workflow and behaviors. However, although certificates are effective proofs of a combination of a principle's name and a public key, it is difficult for a CA to judge the performance and personalities of a user or a platform and therefore assure the trustworthy of their activities by certificates. If this trust relationship between identity and its intention is broken up, an agent or a platform may be malicious, writing or sending malicious agents to other platforms even with a valid certificate. Many suggested research projects address the problem of malicious platforms. Some mechanisms focus on prevention, such as "limited blackbox security" [40] and "computing with encrypted functions" [41]. Other mechanisms focus on detection rather than protection, such as "cryptographic traces" in [42] and the "recording of path histories" in [43].

Furthermore, only the one-hop mobile agent and the stationary agent are considered in our system. We therefore adopt the mechanism of mobile code signing here. However, for multi-hop agents, the corresponding challenge in keeping the computation state of agents safely requires a new approach to prove an agent's integrity rather than code signing. In addition, the tracking of a mobile agent should also be considered in order to locate the mobile agent at any time.

Even more work is possible if we consider the complex situation when the revocation of a credential or a certificate occurs. Because in our system the CA issues certificates off-line, a platform may not get certificate update information in time, which may happens when a user or a platform changed its public-key or if it is deprived of the certificate before the expiry date. Certificate updating is not a new question, many works such as [47] [48] address this problem.

REFERENCE:

[1] M. Breugst, T. Magedanz: Mobile Agent—Enabling Technology for Active Intelligent Network Implementation, IEEE Network, pp. 53-60, May-June, 1998

[2] D. Kotz, R. Gray, S. Nog, et al.: Agent TCL: Targeting the Needs of Mobile Computers, IEEE Internet Computing, pp. 58-67, July-August, 1997

[3] P. Kotzanikolaou, G. Katsirelos, V. Chrissikopoulos: Mobile Agents for Secure Electronic Transactions, Recent Advances in Signal Processing and Communications, World Scientific Engineering Society, pp. 363-368, 1999

[4] A. Bieszczad, B. Pagurek, T. White: Mobile Agents For Network Management, IEEE Communication Surveys, v.1, n.1, pp. 2-9, 1998

[5] W. Theilmann, K. Rothermel: Domain Experts for Information Retrieval in the World Wide Web, Proceeding of 2nd International Workshop on Cooperative Informative Agents (CIA'98), Springer-Verlag, pp. 216-227, 1998

[6] D. Chess, B. Grosof, C. Harrison, D. Levine, and C. Parris: Itinerant agents for mobile computing, Technical Report OC 20010, IBM, March 1995

[7] R. S. Gray: Agent Tcl: A Flexible and Secure Mobile-Agent System. Proceedings of 1996 USENIX Tcl/Tk Workshop, pp. 9-23, San Diego, CA, 1996

[8] C. Baumer, M. Breugst, S. Choy, T. Magedanz: Grasshopper - A Universal Agent Platform Based on OMG MASIF and FIPA Standards, First International Workshop on Mobile Agents for Telecommunication Applications, pp. 1-18, Ottawa, 1999

[9] D. Wong, N. Paciorek, T. Walsh, et al: Concordia: An Infrastructure for Collaborating Mobile Agents, First International Workshop on Mobile Agents 97, LNCS 1219, pp. 86-98, Springer-Verlag, 1997

[10] ObjectSpace: Voyager Homepage, http:// objectspace.com/voyager

[11] Joint submission, MASIF Specification, OMG TC Document orbos/97-10-05

[12] Joint submission, FIPA Specification, FIPA homepage: http://www.fipa.org

[13] FIPA-OS: FIPA-OS Homapage, http://www.nortelnetworks.com/products/announcements/fipa

[14] F. Bellifemine, G. Rimassa, A. Poggi: JADE – A FIPA compliant Agent Framework, In proceedings of the 4th International Conference and Exhibition on the practical Application of Intelligent Agents and Multi-Agents, pp. 97-108, London, 1999

[15] H. Nwana, Ndumu D., Lee L., and Collis J.: ZEUS: A toolkit for Building Distributed Multieagent System, Applied Artificial Intelligence, v.13, n.1, pp. 129-185,1999

[16] K. Bach, R. Harnish.: Linguistic Communication and Speech Acts. M. I. T. Press, Cambridge, Massachusetts, 1979

[17] S. Poslad, M. Calisti: Towards Improved Trust and Security in FIPA Agent Platforms, the Autonomous Agents 2000 Workshop on Deception, Fraud and Trust in Agent Societies, Spain, June, 2000

[18] M. Laukkanen: Evaluation of FIPA-OS1.03, Helsinki, February, 2000. http://fipa-os.sourceforge.net/docs/papers/soneraevaluation.pdf

[19] D. K. Branstad, J. Gait, S. Katzke: Report on the Workshop on Cryptography in Support of Computer Security, NBSIR 77-1291, National Bureau of Standard, 1977

[20] R. L. Rivest: The RC5 Encryption Algorithm, Fast Software Encryption, LNCS1008, pp. 86-96, Springer-Verlag, 1995

[21] X. Lai and J. Massey: A proposal for a new block encryption standard, Advances in Cryptology —EUROCRYPT'90 Proceedings, Springer-Verlag, pp.389-404, 1991

[22] R. L. Rivest, A. Shamir, L. M. Adleman: On digital signatures and public key cryptosystems, MIT Laboratory for Computer Since, Technical Report, January, 1979

[23] R.L. Rivest: The MD4 Message Digest Algorithm, Advances in Cryptology — EUROCRYPT'90 Proceedings, pp. 303-311, Springer-Verlag, 1991

[24] National Institute of Standards and Technology, NIST FIPS PUB 180-1: Secure Hash Standard, April, 1995. http://csrc.nist.gov/fips/fip180-1.pdf.

[25] G. Tsudik: Message Authentication with One-Way Hash Functions, ACM computer Communications Review, v.22, n.5, pp.29-38, 1992

[26] R. M. Needham, M.D. Schroeder: Using Encryption for Authentication in Large Networks of Computers, Communications of the ACM, v.21, n.12, pp. 993-999, 1978

[27] J.Tardo, K. Alagappan, R. Pitkin: Public Key Based Authentication Using Internet Certificates, USENIX Security II Workshop Proceedings, pp. 121-123, 1990

[28] N. B. Bellcore, N. F. Innosoft: RFC 1521 MIME (Multipurpose Internet Mail Extensions), Sept, 1993

[29] I. Satoh: MobiDoc, A Framework for Building Mobile Compound Documents from Hierarchical Mobile Agents, Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents, ASA/MA 2000, Zurich, Switzerland, Sept, 2000

[30] A. Young: Implementation of JANET Authentication and Encryption Services, http://www.jisc.ac.uk/acn/authent/young.html

[31] M. Blaze, J. Feigenbaum et al.: Decentralized Trust Management, Proceedings 1996 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, pp. 164-173, May, Oakland, CA, 1996

[32] M. Blaze, J. Feigenbaum, J. Ioannidis, A. Keromytis: "The Role of Trust Management in Distributed Systems Security" in Secure Internet Programming: Security Issues for Mobile and Distributed Objects, LNCS1603, pp. 185-210, Springer-Verlag, Berlin, 1999

[33] D. Ferraiole, J. Cugini, D. Kuhn, Role Based Access Control: Features and Motivations, Proceedings of 11th Annual Computer Security Applications Conference, pp. 241-248, Dec, New Orleans, CA, 1995

[34] R. Sandhu, E. Coyne, H. Feinstein, C. Youman: Role-Based Access Control Models IEEE Computer, v.29, n.2, pp. 38-47, Feb, 1996

[35] K. Kato, K. Toumura, et al: Protected and Secure Mobile Object Computing in PLANET, Workshop Reader of the 10th European Conference on Object-Oriented Programming, pp. 319-326, 1997.

[36] H. Peine: Security Concepts and Implementation in the Ara Mobile Agent System, Proceedings of 7th IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 236-242, June 1998

[37] JavaSoft, "Java Security", 1998, http://www.javasoft.com/security

[38] G. Necula, P. Lee: Safe, Untrusted Agents Using Proof-Carrying Code, Mobile Agents and Security, LNCS1419, pp. 61-91, Springer-Verlag, 1998

[39] Website: http://www.cryptix.org/.

[40] H. Fritz: Time Limited Blackbox Security, Protecting Mobile Agent From Malicious Hosts, Mobile Agents and Security, LNCS1419, pp. 92-113, Springer-Verlag, 1998

[41] S. Tomas, T. Christian: Protecting Mobile Agent Against Malicious Hosts, Mobile Agents and Security, LNCS1419, p 44-60, Springer-Verlag, 1998

[42] V. Giovanni: Protecting Mobile Agents Through Tracing, Proceedings of the 3rd ECOOP Workshop on Mobile Object Systems, Finland, June, 1997 http://www.cs.ucsb.edu/~vigna/listpub.html

[43] V. Roth: Secure Recording of Itineraries Through Co-operating Agents, Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems, pp. 297-298, 1998

[44] D. D. Denning: A lattice Model of Secure Information Flow, Communications of the ACM, 19(5), May 1976

[45] D. E. Bell and L. J. LaPadula: Secure Computer Systems: Mathematical Foundations and Model. The Mitre Corporation, 1973

[46] M. M. Karnik, A. R. Tripathi: Security in the Ajanta Mobile Agent System, http://www.cs.umn.edu/Ajanta, 1999

[47] M. Naor, K. Nissim: Certificate Revocation and Certificate Update, the Proceedings of the 7th USENIX Security Symposium, San Antonio, Texas, pp. 26-29, January 1998

[48] P. Kocher: A Quick Introduction to Certificate Revocation Trees (CRTs). http://www.valicert.com/company/crt.html