TPM-based protection for mobile agents

Antonio Muñoz* and Antonio Maña

University of Malaga, ETSII, Campus de Teatinos, 29071 Malaga, Espana

ABSTRACT

Mobile agent is a promising paradigm for emerging ubiquitous computing and ambient intelligent scenarios. We believe that security is the most important issue for the widespread deployment of applications based on mobile agent technology. Indeed, community agrees that without the proper security mechanisms, use of mobile agent-based applications will be impossible. From our perspective, the security problem in mobile agents is the gathering of two subproblems; the problem of the agent protection and the problem of the host protection. This paper presents a hardware-based mechanism focused on solving the protection of the agent problem, which is a well known problem named the 'malicious host'. The solution presented in this paper bases its security in the trust and the security functionalities provided by the trusted platform module (TPM). Thus, migration process of mobile agents is protected when it actually takes place. A complete description of the secure migration can be found in the secure migration protocol section of this paper. Moreover, a validation of this protocol was performed by means of the AVISPA tool suite. Additionally, a first study about the use of an alternative protocol as the direct anonymous attestation protocol was done. Finally, the result of this work is the Secure Migration Library for agents (SecMiLiA), which is completely described in following sections.

KEYWORDS

Trusted Computing; mobile agents; cryptographic hardware

***Correspondence**

Antonio Muñoz, University of Malaga, ETSII, Campus de Teatinos, 29071 Malaga, Espana.
E-mail: amunoz@lcc.uma.es

1. INTRODUCTION

Over the years computer systems have evolved from centralized monolithic computing devices supporting static applications, into sophisticated environments that allow complex forms of distributed computing. Throughout this evolution limited forms of code mobility have existed: the earliest being remote job entry terminals used to submit programs to a central computer and the latest being Java applets downloaded from web servers into web browsers. A new phase of evolution is now under way that goes one step further, allowing complete mobility of cooperating applications among supporting platforms to form a large-scale, loosely coupled distributed system. It seems that the catalyst for this evolutionary path is the mobile software agent role.

Along this paper we define the concepts that determine the diverse security levels to understand the role of security in the transmission of private and critical information through an open environment like Internet: (i) Confidentiality is the property that ensures that only those that are properly authorized may be access the information. (ii) Integrity is the property that ensures that information cannot be altered. This modification could be an insertion, deletion or replacement of data. (iii) Authentication is the property that refers to identification. It is the link between the information and its sender. (iv) Non-repudiation is the property that prevents some of the parts to negate a previous commitment or action.

When we are dealing with agent-based systems, these properties are especially important, due to the autonomy and mobility of agents. Thus non-secure agent systems are senseless in open environment such as Internet. Especially whether it deals with critical data, because communications could be spied or even the identities of agents faked. Mobile agents are software entities that move code and data to remote hosts. Mobile agents can migrate from host to host performing actions autonomously on behalf of a user. The last host that executes the agent sends the processed results to the agent sender, i.e. the origin host as it is also called. The use of mobile agents saves bandwidth and permits off-line and autonomous execution in comparison to habitual distributed systems based on message passing. Consequently, mobile agents are especially useful to perform functions automatically in almost all electronic

services, like data mining, electronic commerce and network management. Despite their benefits, massive use of mobile agents is restricted by security issues.

Agent migration consists on a mechanism to continue the execution of an agent on another location [1]. This process includes the transport of agent code, execution state and data of the agent. In an agent system, the migration is initiated on behalf of the agent and not by the system. The main motivation for this migration is to move the computation to a data server or a partner to reduce network load by accessing a data server a partner by local communication. Migration is performed from a source agency where agent is running to a destination agency, which is the next stage in agent execution.

One key issue in agent technology is that an agent is thought to live in a society of agents; *multi-agent systems* (MASs). A *MAS* is composed of several agents collectively capable of reaching goals that are difficult to achieve by an individual agent of monolithic system. This represents a natural way of decentralization, where autonomous agents work as peers or in teams, with their own behaviour and control. However, in contrast to the Remote evaluation [2] and Code on demand [3] paradigms, mobile agents are active and choose to migrate between computers at any time during their execution. This situation makes mobile agent a powerful technology for implementing Ambient Intelligence environment systems and applications.

Software agents are a promising computing paradigm. Scientific community has devoted important efforts to this field [4]. Indeed, many current applications exist based on this technology. Despite of their benefits, a bottleneck exists for the widespread use of mobile agent technology. We aim to the lack of the appropriate security mechanisms for agent-based systems. Subsequently, the application of the current security techniques is not trivial for agent-based system developers, which are usually not security experts.

In Reference [4] two different solutions to provide agent protection are introduced. Firstly, a software-based solution built on the protected computing approach, based on the previous work of Maña [5]. And secondly, a solution that addresses the problem of *malicious hosts* is presented. The second approach consists on a hardware-based solution that takes advantage from the *Trusted Computing Group* (TCG) technology. Along this paper we describe in detail this solution based on cryptographic hardware.

The remainder of this paper is structured as follows. In Section 2, we describe the problem known as the '*malicious hosts*'. In Section 3, we review the previous work in mobile agent security. Section 4 introduces the role of the *Trusted Computing* technology in our solution. In Section 5, we detail the *Secure Migration Library for Agents* (SecMiLiA). Section 6 deals with the core of our approach, the *secure migration protocol*. In Section 7, we give an overview of the implementation of our library. Finally, Section 8 concludes and provides some ideas for ongoing research from host to host performing actions autonomously on behalf of a user.

2. THE PROBLEM OF MALICIOUS HOSTS

The fact that the runtime environment (the host) may attack the program (the agent) plays hardly a role in existing computer systems. Normally, the party that maintains the hosts also employs the program. Nevertheless, in the area of open mobile agents systems, an agent is operated in most cases by another party, the agent owner. This environment leads to a problem that is vital for the usage of mobile agents in open systems: the problem of malicious hosts. A malicious host can be defined in a general way as a party that is able to execute an agent that belongs to another party and tries to attack the agent in some way. The question of which action is considered to be an attack depends on the question which assurances an agent owner needs to use a mobile agent. We identify the following attacks:

Spying out data: The threat of a host reading the private data of an agent is very severe as it leaves no trace that could be detected. This is not necessarily true for the consequences of this knowledge, but they can occur a long time after the visit of the agent on the malicious host. This is a special problem for the data classes such as secret keys or electronic cash, where the simple knowledge of the data results in loss of privacy or money.

Spying out control flow: As soon as the host knows the entire code of the agent and its data, it can determine the next execution step at any time. Even if we could protect the used data somehow, it is rather difficult to protect the information about the actual control flow. This is a problem, because together with the knowledge of the code, a malicious host can deduce more information about the state of the agent.

Manipulation of code: If the host is able to read the code and if it has access to the code memory, it can normally modify the program of an agent. It could exploit either by altering the code permanently, thus implanting a virus, worm or trojan horse. It could also temporarily alter the behaviour of the agent on that particular host only. The advantage of the latter approach consists in the fact, that the host to which the agent migrates cannot detect a manipulation of the code since it is not modified.

Manipulation of data: If the host knows the physical location of the data in the memory and the semantics of the single data elements, it can modify data as well.

Manipulation of control flow: Despite of the fact that host is not granted to access to agent data, it can conduct the behaviour of the agent by manipulating the control flow.

Incorrect execution of code: Without changing the code or the flow of control, a host may also alter the way it executes the code of an agent, we can obtain the same effect as above.

Masquerade: It is the liability of a host that sends an agent to a receiver host to ensure the identity of the receiver. A third party may intercept or copy an agent transfer and start the agent by masking itself as the correct receiver host. A masquerade will probably be followed by other attacks like read attacks.

Denial of execution: As the agent is executed by the host, i.e. passive, the host can simply not execute the agent. This can be used as an attack i.e. in the case that a host knows about a time limited special offer of another host. The host simply can prevent the detection of this offer by the agent by delaying its execution until the offer expires.

Spying out interaction with other agents: The agent may buy the flowers remotely from a shop situated on another host. If the interaction between agent and the remote flower shop is not protected, the host of the agent is able to watch the buy interaction even in the case the host cannot watch the execution of the agent.

Manipulation of interaction with other agents: If the host can also manipulate the interaction of the agent it can act with the identity of the agent or mask itself as the partner of the agent.

Returning wrong results of system calls issued by the agent: in the case that the agent requests the name of the current location. Here the host could mask itself as the agent's home location by returning the corresponding address. The agent then thinks that it is at home and delivers the private information to the host i.e. a variable with the best price found to fly to Paris.

3. BACKGROUND

Once the *malicious hosts* problem is described, we provide a complete description in the background. Firstly, we give list of the most relevant works in the protection of agents and similarly in the host protection. Secondly, a description of the *JADE* security model is provided. Finally, we present the *Trusted Platform Module* (TPM) and its main features.

Several mechanisms for secure execution of agents have been proposed in the literature with the objective of providing protection for the execution of agents and their environments. Most of these mechanisms are designed to provide some type of protection or some specific security property. In this section, we focus on solutions that are specifically tailored or especially well-suited for agent scenarios. More extensive reviews of the state of the art in general issues of software protection can be found in Reference [5,6].

Some mechanisms are oriented to the protection of the agencies (host) against malicious agents. Among these, we found the *Software-Based fault isolation* [7] consisting on isolating application modules into distinct fault domains enforced by software, this technique is commonly referred as *SandBoxing* [8]. The idea behind the *Safe Code Interpretation* [9] is that commands considered harmful can be either made safe for or denied to an agent, the best known of the safe interpreters developed for agents is *Agent Tcl* [10]. A different technique for protecting an agent system is signing code or other objects with a digital signature by means of which the authenticity of an object can be confirmed. A clear example is the *Microsoft's Authenticode*, which is a form of code signing that enables *Java applets* to be signed, ensuring users that the software was has not been tampered with or modified and the identity of the author is verified.

Other technique is known as *state appraisal* [11] based on ensuring that an agent has not been somehow subverted due to alterations of its state information. *Appraisal functions* are used to determine what privileges to grant an agent, based on both on conditional factors and whether identified state invariants hold. An agent whose state violates an invariant can be granted no privileges, while an agent whose state fails to meet some conditional factors may be granted a restricted set of privileges. The basic idea behind the *Path-Histories* [12,13] is to keep an authenticable record of the prior platforms visited by an agent in such a way that a newly visited platform can determine whether to process the agent and what the resource constraints to apply. For this purpose, each agent platform adds a signed entry to the path to indicate its identity and the identity of the next platform to be visited. *Proof-carrying code* approach [14] forces to the code producer to formally prove that the program possesses safety properties, previously stipulated by the code consumer. It is important to mention the fact that this is a prevention technique. One of the most important problems of these techniques is the difficulty of identifying which operations (or sequences of them) can be permitted without compromising the local security policy.

Other mechanisms are oriented toward protecting agents against malicious agencies (hosts). This problem was full detailed in the previous section. Thus, this section briefly describes some partial approaches oriented to solve this problem. Among these approaches, there is the concept of *Partial Result Encapsulation*, which consist on the encapsulation of the results of an agent's actions, at each platform visited to be verified. A version of this technique is the presented by Yee as *Partial Result Authentication Codes* (*PRAC*) [15] consisting of cryptographic checksums formed using secret key cryptography. However this technique presents an important drawback when a malicious platform retains copies of the original keys or key generating functions of an agent.

An improvement is that rather than relying on the agent to encapsulate the information, each platform can be required to encapsulate partial results along the way [13]. However, Yee noted that forward integrity could also be achieved using a trusted third party that performs digital time-stamping. Thus, a timestamp [16] allows one to verify that the contents of a file or document existed, as such, at a particular point in the time. Also Yee raises the concern that the granularity of the timestamps may limit an agent's maximum rate of travel, since it must reside at one platform until the next time period. Another possible concern is the general availability of a trusted time-stamping infrastructure. A variation of this technique is the named *Path Histories* [17], which is a general scheme for allowing an agent's itinerary to be recorded and tracked by another cooperating agent and vice versa. Some drawbacks of this technique include the cost of setting up the authenticated channel and the inability of the peer to determine which of the two platforms is responsible if the agent is killed.

The *Itinerary Recording with Replication and Voting* approach is a technique for ensuring that a mobile agent ar-

rives safely at its destination [18]. The idea is that rather than a single copy of an agent performing a computation, multiple copies of the agent are used. Although a malicious platform may corrupt a few copies of the agent, enough replicates avoid the encounter to successfully complete the computation. Evidently, this approach is similar to *Path Histories*, but extended with fault tolerant capabilities. The technique seems appropriate for specialized applications where agents can be duplicated without problems, the task can be formulated as a multi-staged computation, and survivability is a major concern. One obvious drawback is the additional resources consumed by replicate agents. A detection-based technique is the execution tracing [19] for detecting unauthorized modifications of an agent through the faithful recording of the agent's behaviour during its execution on each agent platform. Each platform involved has to create and retain a non-repudiatable log or trace of the operations performed by the agent while resident there, and to submit a cryptographic hash of the trace upon conclusion as a trace summary or fingerprint. This approach has several drawbacks, the size and number of logs to be retained is the most obvious, and the fact that the detection process is triggered occasionally, based on suspicious results or other factors.

The *Environmental Key Generation* [20] describes a scheme for allowing an agent or take predefined actions when some environmental condition is satisfied. The main weakness of this approach is that a platform that completely controls the agent could simply modify the agent to print out the executable code upon receipt of the trigger, instead of executing it. Another drawback is that an agent platform typically limits the capability of an agent to execute code created dynamically, since it is considered an unsafe operation. The objective of Computing with Encrypted functions [21] is to determine a method whereby mobile code can safely compute cryptographic primitives. The approach is to have the agent platform execute a program embodying an enciphered function without being able to discern the original function, even though the idea is straightforward, the trick is to find the appropriate encryption schemes that can transform arbitrary functions as intended. This technique can be very powerful but does not prevent denial of service, replay, experimental extraction, and other forms of attack against the agent. Hohl [22] proposes the *Blackbox* technique. The strategy behind this technique is scrambling the code in such a way that no one is able to gain a complete understanding of its function, or to modify the resulting code without detection. However, the main drawback is that there is no known algorithm or approach for providing *Blackbox* protection. Several techniques can be applied to an agent to verify self-integrity and avoid that the code or the data of the agent is inadvertently manipulated.

Anti-tamper techniques, such as encryption, *checksumming*, anti-debugging, anti-emulation and some others [23,24] share the same goal, but they are also oriented toward the prevention of the analysis of the function that the agent implements. Additionally, some protection schemes are based on self-modifying code, and code obfuscation [25]. Finally there are techniques that create a two-way protection. Some of these are based on the aforementioned protected computing approach [5].

Along this paper we propose a hardware-based protection infrastructure that takes advantage of the recent advances in trusted hardware, especially of TCG technology to solve the problem of 'malicious hosts'. The basic idea behind the concept of *Trusted Computing* (*TC*) is the creation of a chain of trust between all elements in the computing system, starting from the most basic ones. Consequently, platform boot processes are modified to allow the TPM to measure each of the components in the system and securely store the results of the measurements in *Platform Configuration Registers* (PCRs) within the TPM. This mechanism is used to extend the root of trust to the different elements in the computing platform. Therefore, the chain of trust starts with the TPM, which analyses whether the BIOS of the computer is to be trusted and, in that case, passes control to it. The process is repeated for the master boot record, the OS loader, the OS, the hardware devices and finally the applications. In a Trusted Computing scenario a trusted application runs exclusively on top of trusted and pre-approved supporting software and hardware. Additionally the*TC technology* provides mechanisms for the measurement (obtaining a cryptographic hash) of the configuration of remote platforms by means of 'quote' functions. If this configuration is altered or modified, a new hash value must be generated and sent to the requester in a certificate. These certificates attest the current state of the remote platform, log or trace of the operations performed by the agent.

3.1. Jade security model (JADE-S)

We define the concepts that determine the diverse security levels to a better understanding of the role of security in the transmission of private and critical information through an open environment like Internet: (i) Confidentiality is the property that ensures that only those that are properly authorized may be access the information. (ii) Integrity of the property that ensures that information cannot be altered. This modification could be an insertion, deletion or replacement of data. (iii) Authentication is the property that refers to identification. It is the link between the information and its sender. (iv) Non-repudiation is the property that prevents some of the parts to negate a previous commitment or action.

When we are dealing with *MAS*, these properties are especially important, due to the autonomy and mobility of agents. A *MAS* without security support could not be used in an open environment such as Internet if it deals with critical data, because communications could be spied or even the identities of agents faked. *JADE-S* consists on a plug-in of *JADE* that allows to add some security characteristics in the development of MAS, so that they can start to be used in real environments. It is based on the Java security model and it provides the advantages of the following technologies:

– *JAAS* (*Java Authentication and Authorization Service*) allows establishing access permissions to perform certain operations on a set of predetermined classes, libraries or objects.
– *JCE* (*Java Cryptography Extension*) implements a set of cryptographic functions that allow the developer to deal with the creation and management of keys and to use encryption algorithms.
– *JSSE* (*Java Secure Socket Extension*) allows to exchange critical information through a network using a secure data transmission such as *SSL*.

Several considerations might be taken into account when dealing with *JADE* security. A *JADE* platform may be located in different hosts and have different containers.*JADE-S* structures the agent platform as a multi-user environment in which all components (agents, containers, etc) belong to authenticated (through a login and a password) users, who are authorized by the administrator of the system to perform certain privileged critical actions. Each platform contains a permissions file with a set of actions that each user is authorized to perform. Internally, each agent proves its identity by showing an Identity Certificate signed by the Certification Authority (proved in a transparent way to the agent when their registers in the system and provides the login and the password of its owner). Using these digitally signed certificates the platform may allow or deny certain actions to each agent.

We previously mentioned that each component of the platform belongs to an authenticated user. User who boots the platform also owns the *AMS* and *DF* agents and the main container. Then, when a user wants to join to the platform (through one of his/her agents), it has to provide his/her login and password. These data are checked with the passwords file contained in the platform, which is stored in a ciphered way, similar to *Unix* passwords. Password file is unique and is loaded with the main container. Each agent owned by this user will have an Identity Certificate that contains its name, its owner and the signature of the Certification Authority.

However, in a *JADE-S* platform the permissions to access resources are given to the different entities by following the mechanism defined by the new system provided by Java (*JAAS*) for user-based authentication. Thus, it is possible to assign permissions to parts of the code and to its executers, restricting the access to certain methods, classes or libraries depending on who wants to use them. An entity can only perform an action (send a message, move to another container) if the Java security manager allows it. The set of permissions associated to each identity is stored in the access rights file of the platform (which is also unique and is loaded when the platform is booted). Also Java provides a set of permissions (apart from those that may be defined by the user) on the basic elements of the language: AWTPermission, FilePermission, Socket Permission, etc. Moreover, *JADE-S* provides other permissions related to the behaviour of the agents: AgentPermission, ContainerPermission, etc. There is a list of related actions for every perm for every permission rule to be allowed or denied.

Concerning certification issues, the Certification Authority is the entity that signs the certificates of all the elements of the platform. To do that, it owns a couple of public/private keys so that, for each certificate, it creates an associated signature by ciphering it with its private key (which is secret). Then, when an entity has to be identified the signature may be decrypted using the Authority public key (which is publicly known) and we can check that the identity that the entity wanted to prove matches the one provided by the Authority. The secure platform JADE-S provides a Certification Authority within the main container. Each signed certificate is only valid within the platform in which it has been signed. JADE-S uses the *SSL* protocol (Secure Socket Layer) to provide a secure communication between agents located in different hosts or containers, which provides privacy and integrity for all the connections established in the platform. This is a way of being protected against network sniffers.

3.2. Introduction to trusted platform module

A *TPM* usually is implemented as a chip integrated into the hardware of a platform (such as a *PC*, a *laptop*, a *PDA*, a mobile phone). A TPM owns shielded locations (i.e. no other instance but the TPM itself can access the storage inside the TPM) and protected functionality (the functions computed inside the TPM cannot be tampered with). The TPM can be accessed directly via TPM commands or via higher layer application interfaces (the *TCG Software Stack*, *TSS*).

The TPM offers two main basic mechanisms: It can be used to prove the configuration of the platform it is integrated in and applications that are running on the platform, and it can protect data on the platform (such as *cryptographic keys*). These mechanisms can be performed by means of the *crypto co-processor*, hash and *HMAC algorithm*, key generator, etc, provided by TPM.

In order to prove a certain platform configuration, all parts that are engaged in the boot process of the platform (BIOS, master boot record, etc) are measured (i.e. some integrity measurement hash value is computed), and the final result of the accumulated hash values is stored inside the TPM in a so-called *PCR*. An entity that wants to verify that the platform is in a certain configuration requires the TPM to sign the content of the PCR using a so-called Attestation Identity Key (AIK), a key particularly generated for this purpose. The verifier checks the signature and compares the PCR values to some reference values. Equality of the values proves that the platform is in the desired state. Finally to verify the trustworthiness of an *AIK's signature*, the AIK has to be accompanied by a certificate issued by a trusted *Certification Authority*, a so-called *Privacy CA (P-CA)*. Note that an *AIK* does *not* prove the identity of the TPM owner.

Keys generated and used by the TPM have different properties: Some (so-called *non-migratable keys*) cannot be used outside the TPM that generated them; some (like *AIKs*) can only be used for specific functions. We highlight the fact that keys can be tied to *PCR* values (by specifying *PCR* number and value in the key's public data). This has the effect that such a key will only be used by the TPM if the platform (or some application) configuration is in a certain state (i.e. if the *PCRs* the key is tied to contains a specific value). In order to prove the properties of a particular key, for example that a certain key is tied to specific PCR values, the TPM can be used to generate a certificate for this key by signing the key properties using an AIK. For requesting a TPM to use a key (i.e. for decryption), the key's authorization value has to be presented to the TPM. This together with the fact that the TPM specification requires a TPM to prevent *dictionary attacks* provides the property that only those entities that know the key's authorization value can use that key.

Non-migratable keys are especially useful for preventing unauthorized access to some data present in a platform. Binding such a key to specific *PCR* values and using it to encrypt data to be protected achieves two properties: The data cannot be decrypted on any other platform (because the key is *non-migratable*), and the data can only be decrypted when the specified *PCR*contains the specified value (i.e. when the platform is in a specific secure configuration and is not manipulated).

4. THE ROLE OF TRUSTED COMPUTING IN THE SECURE MIGRATION

Sander and Tschuding [21] asked the question: 'Can a program actively protect itself against its execution environment that tries to divert the intended execution towards a malicious goal?' By means of after a little thought this seems to be a problem impossible to solve because it leads to an infinite recourse. The assessment routine that would detect wrong execution of code or tampering of data and that would try to counter them would also be subject to diversion. For mobile code applications, more specifically for mobile software agents, which are designed to run on potentially arbitrary computers, this problem is of primordial importance. Without strong guarantees on computation integrity and privacy, mobile programs would always remain vulnerable to hijacking and brainwashing.

Pearson [26] defines a related notion, namely that of a trusted platform as follows: 'A Trusted Platform is a computing platform that has a trusted component, probably in the form of built-in hardware, which it uses to create a foundation of trust for software processes'. The concept of trusted computing is in surprisingly recent, although some of key ideas around have been around much longer. Trusted computing refers to the addition of hardware functionality to a computer system to enable entities with which the computer interacts to have some level of trust in what the system is doing.

Agent migration consists on a mechanism to continue the execution of an agent on another location [1]. This process includes the transport of agent code, execution state and data of the agent. The migration in an agent-based system is initiated on behalf of the agent and not by the system. The main motivation for this migration is to move the computation to a data server or a communication partner to reduce network load by accessing a data server a communication partner by local communication. Then migration is done from a source agency where agent is running to a destination agency. Migration can be performed by two different ways. Moving is the process in which the agent is removed from the source agency when is copied in the destination agency. Cloning consists on the agent is copied to the destination agency. From this moment on, the two copies of the agent coexist executing in different places. In the remainder of this paper and at least stated explicitly we will use the term migration to refer to both cloning and moving of agents.

Our approach is addressed on achieve a secure migration process. For this reason we propose a hardware-based mechanism to provide security to agent systems. The TPM provides mechanisms, such as cryptographic algorithms, secure key storage and remote attestation that provides important tools to achieve a high level of security. Remote attestation allows changes to the user's computer to be detected by authorized parties. It works by having the hardware generate a certificate stating what software is currently running. The computer can then present this certificate to a remote party to show that its software has not been tampered with. This functionality is usually combined with public-key encryption so that the information sent can only be read by the programs that presented and requested the attestation, and not by an eavesdropper, such as the computer owner.

Unfortunately, the TPM technology is very complex and so are the procedures and action sequences to use it. The access to the device and the use of TPM functionality is not an easy task, especially for average software developers. Therefore, we have developed a library that provides access to the TPM from software agents. The main advantage for this approach is that developers of agent systems do not need to become security experts, and can access the security mechanisms without taking care of low level details of the TPM technology. Hardware-based solutions can be built on the basis of different devices. We have selected the TPM for our solution because it provides all necessary features. In particular it provides cryptography capabilities, and secure storage, and intimate relation to the platform hardware, and remote attestation, etc.

Remote attestation is perhaps the most important feature for us; it allows the computer to recognize any unauthorized/authorized changes to software. If certain requirements are met then the computer allows the system to continue its operations. This same technology that TPM utilizes can also be applied in agent-bases systems to scan

machines for changes to their environments made by any entity before the system boots up and accesses the network. Additional reasons are the support of a wide range of industries and the availability of computers equipped with TPM. In summary, this element is the cornerstone of our approach. Indeed the security of our system settle in this device as previously described.

The use of TPM in these systems is as follows. Each agency takes measures of system parameters while booting to determine its security, such as BIOS, keys modules from Operating System, active processes and services in the system. Through these parameters an estimation of the secure state of the agency can be done. Values taken are stored in a secure way inside the trusted device, preventing unauthorized access and modification. Agencies have the ability to report previously stored configuration values to other agencies to prove their security. Our system takes advantage of the remote attestation procedure provided by the TPM to verify the security of the agencies. In this way, agents can verify the security of the destination agency before migration.

5. THE SECURE MIGRATION PROTOCOL

In this section, we describe the main protocol involved in the secure migration. However, firstly we analyse different attestation protocols, as well as secure migration protocols, studying their benefits, so that we can build our own secure migration protocol. We use the migration concept both to agent cloning and agent moving, because for protocol there is no difference between them, henceforth we use migration uniquely.

A first approach of a secure migration protocol is in Reference [5]. This protocol provides some important ideas to take into account during the design process of the final protocol, we highlight that the agent trusts in its platform to check the migration security. Besides, we highlight the necessity of using of TPM technology to obtain and report configuration measures through a secure channel. This protocol describes the secure migration to a destination agency following these steps:

(1) Agent ag1 requests to the source agency S for migration to the destination agency D according to a set of requirements (Dreq).
(2) S in collaboration with TPMs (TPM in source agency computer) establish a communication channel with TPMd (TPM in destination agency computer) to obtain the destination agency configuration (Dconf).
(3) Destination agency configuration is analyses to check whether destination requirements (Dreq) are fulfilled.
(4) If these requirements are fulfilled then destination agency D is granted to ag1 migration,

This protocol contributes with some important ideas to consider in the task of design the new version of the protocol, thus the agent trusts in the platform to check the security in the migration. This represents a key issue since it is necessary a root of trust to extend the trust boundaries. Another relevant concept is the requirement of a hardware element such as TPM to obtain and report configuration values securely. Nevertheless, the key ideas provided by this protocol settle on; the protocol shows how an agent from the agency requests to TPM the signed values from *PCRs*. Besides this protocol describes how the agent obtains platform credentials. These credentials, together with PCRs signed values, allow calculate if destination configuration is secure.

Other protocol that provides interesting ideas to take into account when we develop a secure migration system is detailed in Reference [23]. This protocol, which is included and full described in *TGC Specification Architecture Overview* [27], presents some interesting ideas to take into account in our task. A further description of this protocol is out of the scope of this paper, a more detailed version can be found in Reference [27]. The Integrity Report Protocol consists on 'Roots of Trust' for Report has two main objectives, provide secure location to integrity report storage and attest the authenticity of stored values according to the identity of secure platforms. *PCRs* can be implemented in volatile or non-volatile memory and might be tampered against software attacks, as well as tamper-proof resistant against physic attacks. Integrity reports are digitally signed using an *Attestation Identity Key* (*AIK*) to authenticate *PCR* values. Signature includes a nonce to avoid repetition-based attacks. TPM provides a internal key names *Endorsement Key* (*EK*). This is used to establish the TPM owner and issue *AIK credentials*. *TPM owner* is able to generate a *Storage Root Key* (*SRK*) to wrap other keys from TPM. This mechanism facilitates the external and secure storage keys of TPM. *Security report* can be used to determine the configuration of a platform in a concrete instant.

We notice the fact that this protocol is independent of transport mechanism and data delivering. One objective of the attestation is allow that an entity determines whether a TPM signed a message, and even to determine which TPM did it. Nevertheless, for this is necessary a *CA* that issues *AIK* credentials used to attest the platform trustworthy. These credentials will substitute *Endorsement Key* (*EK*) public part, which is not recommended for public distribution. TPM must prove that *AIK's CA* is exclusively used for this, thus *CA* issues encrypted credentials with *EK*. This guarantees that only the TPM that generated that key can decrypt it. On the other side, *CA* must keep the private information related with the TPM which signed. Once the protocol to obtain the platform configuration information is analysed lets study the feedback to build our protocol. This protocol shows how an agent from the agency requests to the *TPM* the signed values of *PCRs* values. The agent obtains *platform credentials*. These credentials together with PCR values signed allow determine whether the destination agency is secure. The usage of these credentials is very in-

teresting since credentials certify that PCR signed belongs to the agency to attest carrying out the *remote attestation* process.

Finally, we describe the complete protocol for the secure migration, which is based on the usage of a *Certification Authority* (*CA*).The Algorithm 1 describes the steps of this protocol.

Algorithm 1 Secure migration protocol.

Ensure: Source Agency –> S
Ensure: Destination Agency –> D
Ensure: Agent –> a
Ensure: Source TPM –>STPM
Ensure: Destination TPM –> DTPM
1: a requests to S (migration to D).
2: S sends to D attestation request.
3: D accepts the request for attestation and sends a nonce (this value is composed by random bits used to avoid repetition attacks) and indexes of PCRs values that needs.
4: S requests to STPM PCR values requested by D together with the nonce all signed.
5: STPM returns requested data.
6: S obtains AIK credentials from its credentials repository.
7: S requests D for PCRs values requested and nonce all signed. Then it sends AIK credentials, which contains the public key corresponding with the private key used to sign data. Additionally, it sends a nonce and the indexes of PCRs that wants to know.
8: D validates the authenticity of received key verifying the credentials by means of the CA public key which was used to generate those credentials.
9: D verifies the PCRs values signature and the nonce received using the AIK public key.
10: D verifies that PCRs values received belong to the set of accepted values and then the agent configuration is valid.
11: D requests to Destination TPM the PCR values requested by D together with the signed nonce.
12: DTPM returns requested data.
13: D obtains AIK credentials from its credentials repository.
14: D sends to S PCR values requested and the nonce signed. Also it sends AIK credentials, which contains the public key corresponding to the private key used to encrypt the data.
15: S validates the authenticity of received key verifying the credentials by means of CA public key that generated those credentials.
16: S verifies the PCR values signature and the nonce received using the AIK public key.
17: S verifies that PCR values received belong to the set of accepted values and then the D configuration is secure. From this point trustworthy between S and D exists.
18: Then S allows to the ‘a’ migrate to D.

Then we analyse the protocol depicted in the Algorithm 1. More relevant ideas from this protocol are the use of an AIK to sign the PCR values, and the use of a CA that validates the AIK, and the use of configurations to compare received results from remote agency. We designed a new protocol based on the study of these three protocols, in such a way that we took advantage of the appeals provided by each of them. Our protocol has some characteristics. The agency provides to the agent the capacity of secure migration. Also the agency uses a TPM that provides configuration values stored in PCRs. TPM signs PCRs values using a specific AIK for destination agency, in such a way that data receiver knows securely TPM identity which signed. A Certification Authority generates needed credentials to correctly verify the AIK identity. Together with signed PCRs values the agency provides AIK credentials in such a way that the requester can correctly verify that data comes from agency TPM. Following we define the 18 steps protocol, used to perform secure migration.

Our protocol fulfils the five main characteristics we previously mentioned. Then we have a clear idea of the different components of the system as well as the interaction between them to provide the security in the migration.

5.1. An alternative based on the direct anonymous attestation protocol

Another alternative for the development of the secure migration is based on the *Direct Anonymous Attestation protocol* (*DAA*). In contrast to the previous protocol here, instead of using a *CA* a issuer certificates authority is used for *DAA keys*. The Algorithm 2 depicts the protocol.

Both protocols allow to verifier entity to trust that *AIK* used to sign configuration values belongs to the entity that generated the signature. Besides to trust that this entity has a TPM installed and working in trusted mode used to generate that *AIK*. On the other hand, *CA-based protocol* presents several disadvantages: (i) A certificate for each *AIK* is necessary to be created, this becomes to *CA* in the bottleneck. (ii) The verification entity and *CA* should be confabulated to violate the security of the system.

However *DAA protocol* does not present such disadvantages but *DAA protocol* needs an additional protocol to verification entity can determine that verified entity poses an *AIK certificate*. SecMiLiA implements the *CA-based protocol*, since we use TPM4java library and this is not provided to implement*Anonymous Direct Attestation protocol*.

5.2. Infrastructure for the secure migration protocol

Some key concepts might be showed to provide to the reader a full understanding of the work presented. As aforementioned, a TPM is physical to prevent forgery, tamper resistant to prevent counterfeiting, and is provided cryptographic

Algorithm 2 Direct anonymous attestation protocol.

Ensure: Source Agency –> S
Ensure: Destination Agency –> D
Ensure: Agent –> a
Ensure: Source TPM –>STPM
Ensure: Destination TPM –> DTPM

1: S request to source STPM configuration data signed.
2: STPM returns requested data.
3: S sends to D configuration data signed together with AIK public key used to sign and a cryptographic proof that poses: (i) An AIK certificate generated using a DAA key. (ii) A certificated for DAA key generated for a DAA certificate issuer.
4: D verifies cryptographic proof and thus AIK validity.
5: D verifies received signed data using AIK public key.
6: D verifies that received configuration values belong to the set of acceptable values and therefore source agency configuration S is secure.
7: D requests to destination TPM configuration signed values.
8: DTPM returns requested values.
9: S sends to D configuration data signed together with the AIK public key used in the signature, as well as a cryptographic proof that poses: (i) An AIK certificate that poses using a DAA key. (ii) A certificate for DAA key generated by a DAA certificate issuer.
10: S verifies the cryptographic proof and therefore AIK validity.
11: S verifies received data signed using AIK public key.
12: S verifies received configuration values belongs to the set of accepted values and therefore source agency configuration is secure. Henceforth a mutual trusted relationship exists between source S and destination agencies S.

functions to support authenticity, integrity, confidentiality, guard against replay attacks, digital signatures, and in if required the use of certificates.

Attestation identities prove that all they correspond to a Trusted Platform and a specific identity always identifies the same platform. Then each identity is created on individual trusted platform, with attestation from a PKI Certification Authority (CA). Each identity has randomly created asymmetric cryptographic key and an arbitrary textual string used as an identifier for the attestation identities (chosen by the platform owner). To obtain the attestation from the CA, the platform´s owner sends the CA information that proves that the identity was created by a genuine Trusted Platform. This process uses signed certificates from the manufacturer of the platform and uses a secret installed in the TPM. That secret is known only to the Trusted Platform and only used under control of the owner of the platform. In particular, it is not divulged to arbitrary third parties, unlike the cryptographic attestation values.

5.3. Formal verification of the protocol

Once the secure migration protocol is described our next target is the validation of this protocol. Security protocols aim to provide security guarantees between parties communicating over insecure networks. These protocols are at the core of security-sensitive ICT systems applied in a variety of industrial relevant domains (i.e. health-care, e-commerce, and e-government). The proper functioning of security protocols is thus a crucial factor for the public acceptance of ICT systems as a security protocol failure may have critical consequences on the ICT system's end-user (i.e. patients, business organizations, citizens).

This is the reason why many tool-supported approaches to security validation and verification have been applied on security protocols in the last decade discovering many uncovered flaws and subtle vulnerabilities. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications [28] has been quite remarkable in this respect by successfully analysing more than 50 protocols of relevance for standardization bodies and for the e-business application domain.

AVISPA is an automatic push-button formal validation tool for Internet security protocols, which is a shared-cost RTD (FET open) project, funded by the European Commission under the Information Society Technologies Programme operating within the Fifth Framework Programme. It encompasses all security protocols in the first five OSI layers for more than twenty security services and mechanisms. Furthermore this tool covers (that is verifiable by it) more than 85 of IETF security specifications. AVISPA library available on-line has in it verified with code about hundred problems derived from more than two dozen security protocols. AVISPA uses a High Level Protocol Specification Language (HLPSL) to feed a protocol in it; HLPSL is an extremely expressive and intuitive language to model a protocol for AVISPA. The operational semantic of this tool is based on the work of Lamport on Temporal logic of Actions. Communication using HLPSL is always synchronous. Once a protocol is fed in AVISPA and modelled in HLPSL, it is translated into Intermediate Format (IF). IF is an intermediate step where re-write rules are applied in order to further process a given protocol by back-end analyser tools. A protocol, written in IF, is executed over a finite number of iterations, or entirely if no loop is involved. Eventually, either an attack is found, or the protocol is considered safe over the given number of sessions. System behaviour in HLPSL is modelled as a 'state'. Each state has variables which are responsible for the state transitions; that is, when variables change, a state takes a new form. The communicating entities are called 'roles' which own variables. These variables can be local or global. Apart from initiator and receiver, environment and session of protocol execution are also roles in HLPSL. Roles can be basic or composed depending on if they are constituent of one agent or more. Each honest participant or principal has one role. It can be parallel, sequential or composite. All communication between roles and the intruder are synchronous. Communication channels are also

represented by the variables carrying different properties of a particular environment.

The language used in AVISPA is very expressive allowing great flexibility to express fine details. This makes it a bit more complex than Hermes to convert a protocol into HLPSL. Further, defining implementation environment of the protocol and user-defined intrusion model may increase the complexity. Results in AVISPA are detailed and explicitly given with reachable number of states. Therefore regarding result interpretation, AVISPA requires no expertise or skills in mathematics contrary to other tools like HERMES [29] where a great deal of experience is at least necessary to get meaningful conclusions.

Of the four available AVISPA Back-Ends we chose the OFMC Model, which is the unique that uses fresh values to generate nonce's. However, this alternative requires a limit value for the search. The results of our research are the following:

Algorithm 3 AVISPA results.

```
SUMMARY
SAFE
DETAILS
BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
/avispa/avispa-1.1/testsuite/results/protocolo_2.txt.if
GOAL
as_specified
BACKEND
OFMC
COMMENTS
STATISTICS
parseTime: 0.00s
searchTime: 18.40s
visitedNodes: 4 nodes
depth: 400 plies
environment()
```

Therefore we can observe the feedback of running our model with the AVISPA tool. These results show that the summary of the protocol validation is safe. Also some statistics are shown among them depth line indicates 400 plies, but this process has been performed for 25, 50, 75, 100, 150, 200, 250, 300, 350 and 400 of depth values with similar results. In the verification of the Secure Migration Protocol with AVISPA, we have checked the authentication on the following variables *pcra*(*pcrib*), *pcrb*(*pcria*), *aikab* and *aikba*, because these are the values exchanged among the agents A and B.

6. SECMILIA: SECURE MIGRATION LIBRARY FOR AGENTS

This section introduces the *SecMiLiA*. It aims to serve as a platform for the agent migration on top of the JADE platform. This library makes use of the inherent security features provided by the TPM to provide a secure environment. In this way, the migrations are supported by a trusted platform.

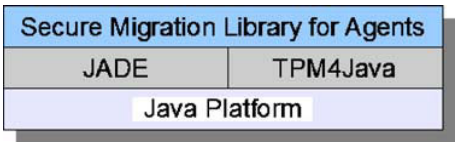

Figure 1. Stack packages.

Figure 1 shows how *SecMiLiA* is the application of the Secure Migration Protocol in a real agent platform, in this case we implemented on *JADE*. We enforce the security of the migration process in the TPM. To achieve a full development of Java we make use of the functionalities of the *TPM4Java* library. *TPM4Java* facilitates the access to the TPM functionalities, which is completely developed in *Java*. Nevertheless, the current version of this library cannot be used directly for our purposes. For this reason, we have extended TPM4Java, as described in Section 7.3.1.

We highlight as one of the main objectives in the design of this SecMiLiA the provision of a secure environment in which agents can migrate securely. Secondly, we focused on achieving a easily usable library. Obviously, agent systems developers have not a huge expertise in security. Additionally, we concentrated in achieving a versatile library easily adaptable to be used in different cryptographic devices (i.e. *smartcards*), as well as to get a generic and flexible interface library. All these functionalities allow users to take advantage of this library to solve a wide range of problems.

We previously aimed that the most important functionality provided by the library is the *secure migration mechanism*. This secure mechanism is based on the *remote attestation* of the destination agency, before the migration actually takes place, the idea is to warrant that agents are always executed in a secure environment. Additionally, we provide a secure environment for the execution of agents, in such a way that malicious agents are not able to modify the host agency.

The library was designed according to the following set of requirements. Our main objective, which is directly related with the functional requirements, was to provide a mechanism for secure agent migration in JADE platform. The goal was to make possible to extend the trusted computing base for agents to the next destination agency before migrating to it. It is important to mention that each agency must provide local functionality to perform the secure migration mechanism, which requires the presence of the TPM for remote attestation. Similarly each agency must provide the functionality to allow to other agencies to take remote integrity measures to determine whether its configuration is secure. The library includes a protocol that allows two agencies to exchange information about their configurations to support the previous processes. This exchange establishes the

bases of the mutual trust between agencies. Concerning the non-functional requirements, the library must seamlessly integrate in the JADE platform, in such a way that its use does not imply modifications in the platform. Additionally, the operation of the library must be transparent to the user. We decided to create an additional abstraction layer that provides generic services in order to facilitate the adaptation of the library to new devices. In this way, the generic services remain unchanged. One final requirement has to support both types of migration (*moving and cloning*). Concerning the architecture, TPM plays the role of trusted element of each agency, able to certify to other agencies its trustworthiness.

7. SECMILIA IMPLEMENTATION MAIN ISSUES

The main objective of the work presented in this paper is two-folded: (i) to solve the problem of 'malicious hosts' and (ii) to provide a transparent solution to integrate security mechanisms in agent systems. The first of these objectives is described in next sections, while the second is following addressed.

To reach the second proposed objective implies on providing mechanisms that are simple to use for agent-based system developers, and a library easy to embed in other security devices such as smartcards. Besides, to provide a library with such a generic and flexible interface allowing users to solve some of security problems, despite of the lack of expertise in security. Naturally, SecMiLiA is easily extensible to be included new functionalities in. As shown in Figure 1, SecMiLiA is based on the one hand on the JADE platform and on the other hand on the TPM4Java library. TPM4Java provides access to the TPM functions from the Java programming language. However, a straightforward application of TPM4Java in agent-based systems is not possible. Thus, the TPM4Java library was enhanced to provide new functionalities for agent migration.

We aimed that the most important functionality provided by the library is the secure migration mechanism. This mechanism is based on the remote attestation of the destination agency, before migration actually takes place, in order to guarantee that agents are always executed in a secure environment. Additionally we provide an environment for secure agent execution, in such a way that malicious agents are not able to manipulate the host agency. In the library design process we prepared a list of requirements that library might fulfils. Indeed, the library is developed fulfilling those requirements. Next the list of requirements is presented:

- Functional requirements; we considered that our main objective is to provide a mechanism for secure agent migration in JADE platform. The goal is to make possible for agents to extend their trusted computing base to the next destination agency before the migration. Therefore every agency must provide local access to secure migration functionality. Besides, every agency must provide the mechanisms to allow other agencies to take remote integrity measures. To calculate whether the configuration values measured match with the trusted values and to determine its security. Once we presented that our system is based on the secure migration concept. Next step consists on the design of a protocol that allows two agencies to exchange the information about their configurations. This exchange establishes the basis of the mutual trust between agencies.
- Non-functional requirements; SecMiLiA might be integrated in the JADE platform, in such a way that its use does not imply modifications in the platform. Additionally, the operation of the library must be transparent to the user. In order to facilitate the adaptation of the library to new devices we decided to create an additional abstraction layer that provides generic services. In this way the generic services remain unchanged. One final requirement has to support both types of migration (moving and cloning). Figure 2 shows a class diagram concerning the architecture, where each agency can host different agents and is supported by TPM functionalities. TPM plays the role of trusted element of each agency, able to certify to other agencies its trustworthiness.

In following sections, we describe the main design and development issues in the SecMiLiA creation process. For this purpose we study the architecture of this in details as well as we show the components and their related functionalities. Main use case consists on a user that uses SecMiLiA

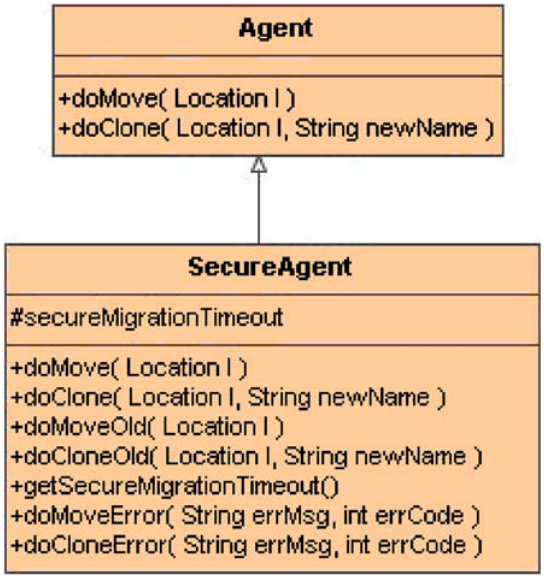

Figure 2. SecureAgent class.

to develop a regular, but secure MAS. Traditionally in these types of systems, the user defines the set of agents that compound the system. Concretely JADE defines an agent by means of a class that inherits from Agent class. Throughout this new agent created is provided of the basic behaviour of an agent. Then user defines the specific behaviour of this agent. Among the basic functionality of a JADE Agent we found the compatibility with inter-containers migration. JADE security model is briefly described in Section 2.1.1. Any secure migration is provided by JADE Agent class. Thus, we created a new class that inherits from this class and redefines migration methods to perform them securely.

Concerning the main migration methods we notice; doMove(Location1); to Move the agent from a source container to a destination one. doClone(Location1, String newName); to clone the agent in container1 using newName as the name. Inter-platform migration methods intra-platform migration methods are shared but the most relevant difference between them is the dynamic type of param1.

Two main services are provided by SecMiLiA that use AgentMobility Service to perform a secure inter-platform migration in the same platform, and SecureInterPlatformMobility service that use InterPlatformMobility service to perform the secure intra-platform migration.

7.1. Design and development

In this section we outline the most relevant issues in the design and development of the SecMiLiA. Therefore we briefly describe some services provided by the library and some considerations about the involved technology.

7.2. SecMiLiA: SecureAgentMobility service

Along this section we deeply describe the most important services provided by *SecMiLiA*. *SecureAgentMobility* service wraps the secure migration functionality of *SecMiLiA*. Figure 3 depicts a class diagram of the service, as well as the most relevant methods. Concretely, *Helper class* provides two important methods *secureMove* that allows secure agents moving securely to destination and *secureClone* that allows to secure agents to be cloned securely in destination containers.

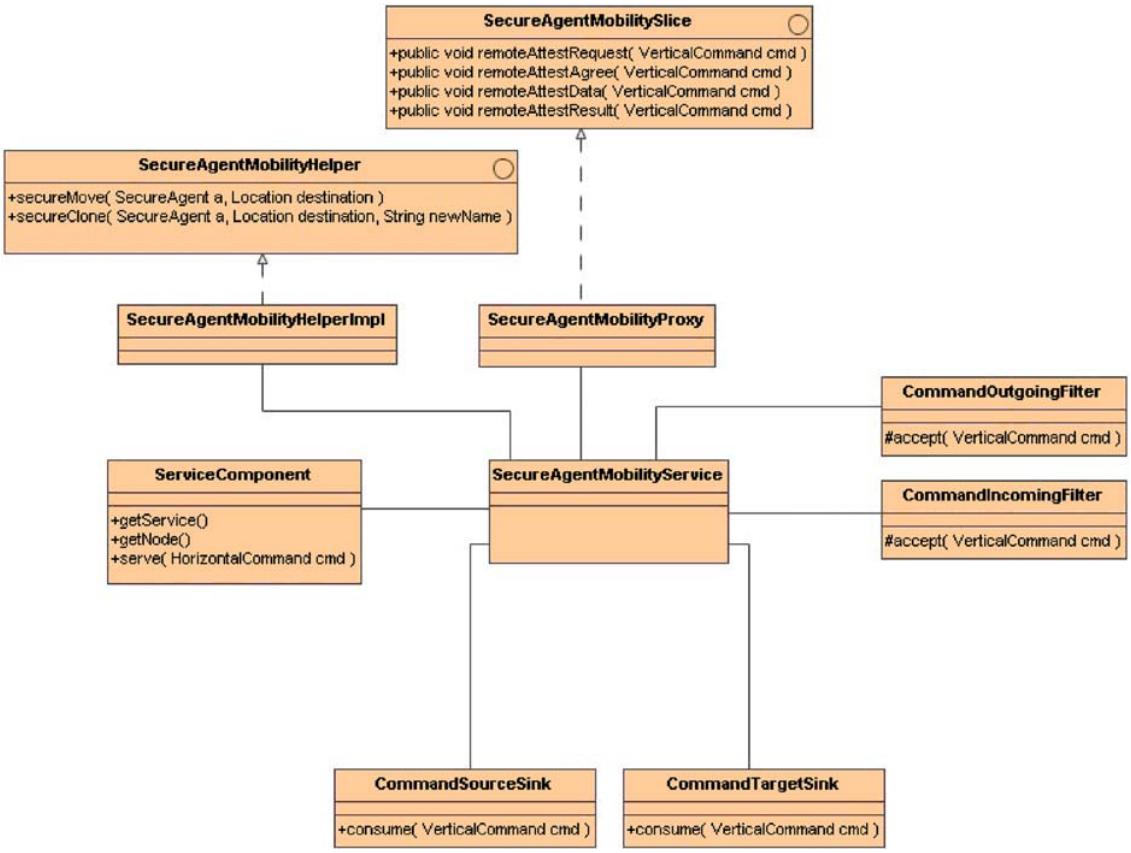

Figure 3. SecureAgentMobility service class diagram.

Algorithm 4 presents a scenario where an agent request for a service to move to a container (c2). This scenario is composed by the following stages:

Algorithm 4 The agent SA is moved to the container C2.

1: SA agent requests for S1 service to move to C2.
2: Service sends a remote attestation request to S2 service.
3: S2 service accepts the request.
4: S1 service sends requested information to S2.
5: S2 service responses to S1 sending the attestation result.
6: S1 service starts agent migration to C2 container.

This protocol considers the service as a unique entity. However, only the most relevant components are sketched in order to clarify. Other important issue is that the service invokes *doMoveOld* method to start the migration. This functionality is similar to *doMove* from *Agent class*, moving agent to destination. Previously to this migration the service checks the trustworthy in the destination agency. This provides a similar behaviour of *doMove* method from *SecureAgent* and *doMove* method from *Agent class*. Content of these messages is encapsulated using *AttestRequest_Interface* and *AttestData_Interface* classes *AttestRequest_Interface* class provides access to data from a request attestation message, and *AttestData_Interface* provides access to attestation information from a concrete container. Both classes encapsulate information from attestation protocol messages. Containers complete needed data using set methods in each message, in such a way that the other container continues the attestation procedure. In order to destination container continues the attestation procedure source container completes the needed data by means of set method. Next Figure 4 depicts *SecureAgentMobility* services uses *AttestTool_Implement* class to complete data messages.

AttestTool_Implement class handles attestation protocol messages, that is, it generates the messages in their sources and verifies them in destination. *AttestTool_Implement* class is provided by access to system TPM by means of *TPM_Interface* as well as to a *CA* throughout *CA_Interface* interface. This fact allows using both entity functionalities to complete messages. Also, *AttestTool_Implement* class can access to system configuration through *AttestConfig_Interface* interface. *AttestTool_Implement* class manages TPM access in such a way that attestation protocol can be performed. *AttestTool_Implement* class behaviour is similar to *Key Cache Manager* managing TPM keys. *AttestTool_Implement* class implements *AttestTool_Interface* interface where secure migration process states codes are defined. These error codes permits to agents determine the happened when *doMoveError* and *doCloneError* methods are called. It is necessary the use of a TPM and a *CA* in order to generate and verify the attestation messages content. Moreover TPM provides the functionalities to generate attestation data that is *AIK* generation, data signature, nonce values generation, etc. *CA* provides credentials generation. Credentials are used to certify *AIKs* objectives in the rest of containers trust in *AIK* signed data.

A relevant aspect in the use of *AIK* in the protocol is to sign attestation data. TPM generates *AIK* and this must be certified by a *CA*. Protocol 5 details how this key is generated and certified.

Several interfaces interact in this process.

- *AIKRequestData_Interface*: Contains data to allow TPM generates *AIK* as well as creates credentials generation request.
- *AIKRequestData_Interface*: Contains data to allow *CA* generate credentials to *AIK*.
- *AIKResponse_Interface*: Contains data to allow TPM obtain *CA* generated credentials.

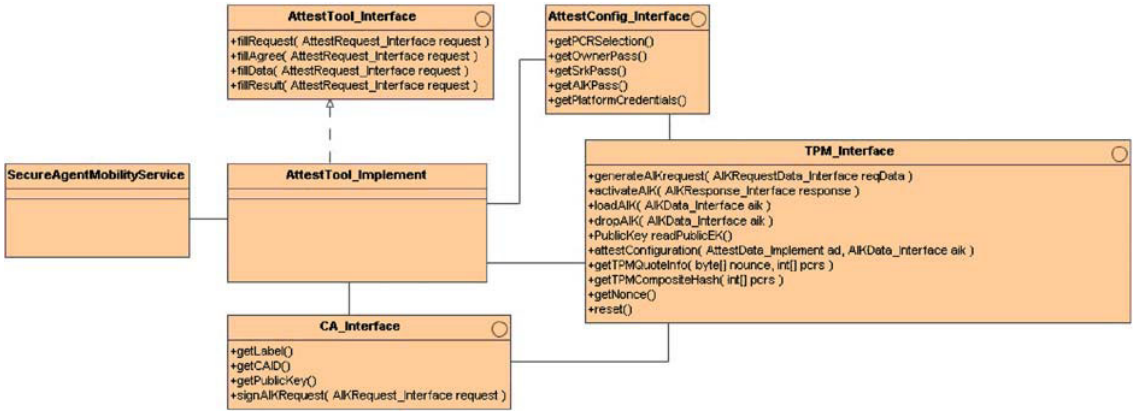

Figure 4. Main classes of the service.

Algorithm 5 Key generation and certification.

1: *AttestTool_Implement* requests for generate an *AIK* key and a credentials request to TPM.
2: TPM generates AIK key as well as the request and these are delivered to *AttestTool*. Request is encrypted in such ways that only CA is able to have in clear.
3: *AttestTool_Implement* sends a *CA* request.
4: *CA* decrypts the request and generates corresponding credentials which are delivered to *AttestTool_Implement* in such a way that only TPM have in clear.
5: *AttestTool_Implement* sends *CA* response to TPM.
6: TPM decrypts the request and sends key data to *AttestTool_Implement*.

At this point we briefly described more relevant classes. *AIKRequestData_Interface* interface provides *getIdentityLabel() method*, this returns *AIK* identity label. Once *AIK* reaches corresponding destination, service can generate configuration attestation data as well as sign them. Finally another relevant component of the solution, these are credential. *CA* generates *AIK credentials* defined by *AIKCredentials_Interface interface*. Thus we finish *SecureAgentMobility service* analysis, and then we analyse *SecureInterPlatformMobilityService service* design.

7.3. Technology

Along this section we briefly described some of the underlying technology of secure migration library of agents. This section is organized as follows: a subsection describing the TPM4Java library and a subsection that describes some extensions to TPM4Java especially requested in our solution.

7.3.1. Use of *TPM4Java* library.

TPM4Java consists on java library developed in the Darmstadt Technique University. *TPM4Java* provides a new friendly layer to access to TPM functionality. Briefly *TPM4Java* architecture is deployed on three main layers.

The architecture of TPM4Java is composed of three layers: (i) a high level layer that provides access to common TPM functionality (this is defined in *TssHighLevel* layer). (ii) A low level layer that provides access to almost full TPM functionality (this is defined in *TssLowLevel* layer). And (iii) a *backend* used by the library that sends commands to TPM. High level layer is developed in such a way that uses low level layer, then all tasks that can be done with high level layer can be performed by means of lower levels calls. Accessing high level layer user must code:

$$TssHighLevel \text{ highLevel} = TssFactory.getHighLevel();$$

TPM4Java provides two main functional backends and one experimental:

- *Linux backend*. Meaning this backend, *TPM4Java* provides access to TPM by this file*('/dev/tpm')*. This fact implies that user is allowed to read and write on that file. In the case that uses a different file is mandatory to specify by this linux command: '*Dtss.tpm=linux./dev/myTPM*'.
- *Window backend*, a TPM driver provided by the manufacturer is provided for any windows-based system. This file is usually named tpmddl.dll and can be found at *windows/system32* folder. *TPM4Java* provides a file named *javaddl.dll* accessible by the application that uses *TPM4Java*.
- *Csharp backend*. This is experimental backend. This solution includes a demon needed to run the application. To use the demon is mandatory for the application specify in commands line next message:'*Dtss.tpm=cs.localhost:12321*'. The application establishes a communication channel with the demon running in localhost. Therefore localhost can be updated by hostname where demon is running.

Finally the most relevant classes used in the development of our library are: (i) The *TssLowLevel interface* provides access to high level layer. (ii) The *TssHighLevel interface* provides access to low level layer. (iii) The *PcrSelection.class* provides a type of data to manage the information of a set of PCR records. (iv) The *SimplePrivacyCA* class provides an implementation of basic functionality of a *CA*. (v) The *TCPAIdentityRequest* creates an *AIK* a class of this type is generated with needed data by *CA* generates the needed information to active the *AIK*. (vi) The *TCPAIdentityCredential class* contains the information generated by *CA* to activate the *AIK*. Finally, (vii) the *TPMKeyWrapper class* contains the information of a key, with its public part and its private one. Private part is encrypted in such a way that is in clear only in TPM.

7.3.2. Improvements to *TPM4Java*.

While developing Services Provided by Library some classes and interfaces from *TPM4Java* library have been modified. Concretely the modified classes are:

- *TssLowLevel interface*. *TPM_OwnerReadPubek* method is added because this is not included in *TPM4Java* but *TCG 1.2 specification* includes that.
- *TssHighLevel interface*. *GenerateAIK* method is modified to returns a *TPMKeyWrapper* class corresponding to generated *AIK* instead of key handler in TPM. Therefore *AIK* data are needed to store it in the hard disc.
- *TssCoreService class* implements *TssLowLevel* interface. *TPM_OwnerReadPubek* method has been added.
- *TssHighLevelImpl class* implements TssHighLevel interface. *GenerateAIK* method has been modified in

such a way that a *TPMKeyWrapper* class is returned.
– *PrivacyCA class* is added, this is an altered version of *SimplePrivacyCA* class provided by *TPM4Java*.

8. CONCLUSIONS AND AREAS FOR FUTURE RESEARCH

In this paper, we have presented a new approach for the agent-based systems protection based on specific hardware. To validates our approach we provide a proof of concept by means of the ' SecMiLiA'. Along this paper, we deeply describe SecMiLiA library and all protocols involved. Especially we discuss the 'secure migration protocol' as the basis to our library. Moreover, we provide the methodology to validate this protocol by means of *AVISPA* validation tool.

Concerning the ongoing work, an important issue is the implementation of the Anonymous Direct Attestation protocol. We mentioned that our library is based on the attestation protocol based on the Certification Authority protocol described in Section 3.1. Our solution is based on this protocol due to the fact that TPM4Java only implements this. Certification Authority protocol provides important advantages as previous sections described but this entails some disadvantages. Among these disadvantages we highlight the needed to generate a certificate for each used key in every attestation, which implies that many requests might be performed to the CA and this provokes a bottleneck for the system. Besides, we found that whether verification authority and CA act together the security of the attestation protocol can be violated. That is, the use of the anonymous direct attestation protocol entails some advantages. Let us see through an example, certify issuer entity and verifier entity cannot collaborate to violate the system security. Therefore, a unique entity can be verifier and issuer of certificates. Last but not least of the found advantages of this approach is that certificates only need to be issued once which solves the aforementioned bottleneck. These advantages prove that the anonymous attestation protocol is an interesting option for attesting.

Possible future lines of research are the improvement of the keys management system of the library. Our library uses RSA keys for attestation protocol that must be loaded in TPM. However the size for key storage in the TPM is very limited, then it must be carefully managed to avoid arisen space problems.

Our library handles the keys in such a way that only one key is loaded in TPM simultaneously, therefore keys are loaded when will use and downloaded when used. This procedure is not very efficient due to the continuous loading and downloading of keys. For this problem we propose the possibility of downloading the same key that we will use in next step. Of course this entails more issues. A different improvement in the key management lies on caching these keys. Then, several keys can be loaded simultaneously in TPM and that these can be change when more available space is needed. This solution entails to develop a key replace policy to determine which key delete from TPM to load a new key in the cache, but this task is out of the scope of this paper.

Another future work is to extend the library with new functionalities to secure migration services in order to provide concurrency. Secure migration services implemented in the library provide secure migration to a remote container. However they have capacity to handle a unique request simultaneously, then migration requests arriving while migration is performed are refused. This happens due to TPM management of the problem together with the aforementioned key management problem. A possible improvement of the library is to provide of secure migration service with the capability handle simultaneously several requests.

ACKNOWLEDGMENT

Work partially supported by E.U. through projects SERENITY (IST-027587) and OKKAM (IST- 215032) and DESEOS project funded by the Regional Government of andalusia.

REFERENCES

1. General Magic, Inc. The Telescript Language Reference. Online available at http://www.genmagic.com-Telescript/ TDE/TDEDOCS_HTML/telescript.html, 1996
2. Stamos JW, Gifford DK. Remote Evaluation. *ACM Transactions on Programming Languages and Systems (TOPLAS) archive* 1990; **4**(4): 537–564. ISSN:0164-0925
3. Brooks RR. Mobile code paradigms and security issues. *Internet Computing, IEEE* 2004 **8**(3): 54-59.
4. Maña A, Muñoz A, Serrano D. Towards Secure Agent Computing for Ubiquitous Computing and Ambient Intelligence. In *Fourth International Conference*, Ubiquitous Intelligence andComputing, Hong Kong (China) 2007; LNCS.
5. Maña, A. Software Protection based on Smartcards. *PhD Thesis*. University of Mlaga. 2003.
6. Hachez G. A Comparative Study of Software Protection Tools Suited for E-Commerce with Contributions to Software Watermarking and Smart Cards. *PhD Thesis*, Universite Catholique de Louvain, 2003.http://www.dice.ucl.ac.be/hachez/thesis_gael_hachez.pdf
7. Wahbe R, Lucco S, Anderson T. Efficent Software-Based Isolation. In *Proceedings of the Fourteenth ACM Symposium on Operating Systmes Principles*. ACM SIGOPS Operating Systems Review, 1993; 203–216.
8. Gosling J, Joy B, Steele G. The Java Language Specification. Addison-Wesley, 1996.

9. Ousterhout JK. Scripting: Higher-Level Programming for the 21st Century. *IEEE Computer* 1998; 23–30.
10. Gray RS. Agent Tcl: A Flexible and Secure Mobile-Agent System. In *Proceedings of the Fourth Annual Tcl/Tk workshop (TCL 96)*, 1996; 9–23.
11. Farmer W, Guttman J, Swamp V. Security for Mobile Agents: Authentication and State Appraisal. In *Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS'96)*, September 1996; 118–130.
12. Ordille JJ. When agents Roam, Who can You Trust? In *Proceedings of the First Conference on Emerging Technologies and Applications in Communications*, Portland, Oregon, May 1996.
13. Chess D, Grosof B, Harrison C, Levine D, Parris C, Tsudik G. Itinerant Agents for Mobile Computing. *IEEE Personal Communications*, 1995; **2**(5): 34–49.
14. Necula G. Proof-CaiTying Code. In *Proceedings of 24th Annual Symposium on Principles of Programming Languages*, 1997.
15. Yee BS. A Sanctuary for Mobile Agents, *Technical Report CS97-537*, University of California in San Diego, April 28, 1997. Online available at http://www-cse.ucsd.edu/users/bsy/index.html
16. Haber S, Stornetta S. How to Time-Stamp a Digital Document, *Journal of Cryptology* 1991; **3**: 99–111.
17. Roth V. Secure Recording of Itineraries Through Cooperating Agents. In *Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations*, INRIA, France, 1998; 147–154.
18. Scheider FB. Towards Fault-Tolerant and Secure Agency. In *Proceedings 11th International Workshop on Distributed Algorithms*, Saarbucken, Germany, September 1997.
19. Vigna G. Secure Recording of Itineraries Through Cooperating Agents. In *Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations*, INRIA, France 1998; 147–154.
20. Riordan J, Scheneider B. Environmental Key Generation Towards Clueless Agents. *Mobile Agents and Security. Lecture Notes in Computer Science 1419 Springer*, Vigna G. (ed.). 1998. ISBN 3-540-64792-9.
21. Sander T, Tschudin C. Protecting Mobile Agents Against Malicious Hosts. In *Mobile Agents and Security*, *Vol. of 1419 in Lecture Notes in Computer Science*, Vigna G. (ed.). Springer 1998, ISBN 3-540-64792-9, 1998.
22. Hohl F. Time Limited Blackbox Security: Protecting Mobile Agents From MaliciousHosts. In *Mobile Agents and Security*, *Vol. of 1419 in Lecture Notes in Computer Science* Vigna G. (ed.), Springer-Verlag, 1998; 92–113
23. Trusted Computing Group: TCG Specifications. 2005. Online available at https://www.trustedcomputinggroup.org/specs/
24. Stem JP, Hachez G, Koeune F, Quisquater JJ. Robust Object Watermarking: Application to Code. In *Information Hiding '99*, *vol 1768 of Lectures Notes in Computer Science (LNCS)*, Pfitzmann A (ed.), pages 368-378, Springer-Verlag: Dresden, Germany, 2000. http://citeseer.nj.nec.com/stem00robust.html
25. Collberg C, Thomborson C. Watermarking, Tamper-Proofing, and Obfuscation—Tools for Software Protection. *Technical Report 170*, University of Auckland, 2000.
26. Pearson S. How can you trust the computer in front of you? *Technical Report*, Trusted E-Services Laboratory, HP Laboratories Bristol. HPL-2002-222, November 5th, 2002.
27. Trusted Computing Group. TCG Specification Architecture Overview, Revision 1.4 (2007), https://www.trustedcomputinggroup.org/groups/TCG14 ArchitectureOverview.pdf
28. Armando A, Basin D, Bouallagui M, Chevalier Y, Compagna L, Modersheim S, Rusinowitch M, Turuani M, Vigano L, Vigneron L. The AVISS Security Protocol Analysis Tool. In *Computer-Aided Verification CAV'02*, *LNCS 2404*, Brinksma E, Larsen KG (eds.) Springer-Verlag, Heidelberg, 2002; 349–354. URL of the AVISS and AVISPA projects: www.avispa-proiect.org
29. Hussain M, Seret D. A Comparative study of Security Protocols Validation Tools: HERMES vs. AVISPA. In *Proceedings of International Conference on Advanced-Communication Technology (ICACT)* IEEE Communication Society, 2006.