International Journal of Computer Science and Applications,

Vol. 6, No. 5, pp 30 - 58, 2009

PROTECTING AGENTS FROM MALICIOUS HOSTS USING TPM

Antonio Muñoz, Antonio Maña and Daniel Serrano*

University University Malaga†
famunoz,amg,serranog@lcc.uma.es

Software agents represent a promising computing paradigm. They are an elegant technology to solve problems that can not be easily solved in other way. The Scientific Community has proved that the use of the software agents approach simplifies the solution of difierent type of traditional computing problems. A proof of this is that several important applications exist based on this technology. It is clear that software agents have so many benefits, but unfortunately, a lack of the appropriate security mechanisms for systems based on them represents a barrier for the widespread use of this paradigm in the industry. The application of the current security mechanisms is not trivial for agent based system developers, since they usually are not security experts and consequently do not count on the appropriate expertise. This paper presents a new protection infrastructure solving the problem known as malicious host in mobile agent systems. This protection infrastructure implements a secure protocol to migrate agents from host to host relying in hardware elements, particularly it is based on the recent advances in Trusted Platforms Modules (TPM) computing. In order to provide an easy way of using the proposed infrastructure we provide it by means of an extension to the Java Agent Development framework (JADE). Finally, the migrating protocol presented has been validated using the Automated Validation of Internet Security Protocols and Applications (AVISPA) framework, this validation is presented in this paper.

Keywords: TPM; Agent Protection; Hardware Protection.

1. Introduction

According to a well-accepted de_nition, a software agent is ”a piece of software that acts on behalf of a user or other program in a relationship of agency”. The idea of ”acting on behalf of” implies the authority to decide which action (if any) is appropriate. Among the most important characteristics of agents we can highlight the following: Autonomy (agents behave independently according to their state), proactivity (agents are able to initiate actions without external commands), reactivity (agents react in timely fashion to direct environment stimulus), situatedness (ability to settle in an environment that might contain other agents), directness (agents live in a society of other collaborative or competitive groups of agents), and social ability (agents can interact with other agents, possibly motivated by collaborative

* E.T.S.I.Informatica, Campus Teatinos 29071, Malaga (SPAIN)
† Campus Teatinos, Malaga, 29071,Spain

problem solving). The latter characteristic means that agents are supposed to live in a society with other agents; each of these societies is called a multi-agent system (MAS). Multi-agent systems are composed of several agents collectively capable of reaching goals that are difficult to achieve by an individual agent of monolithic system.

Besides, a MAS represents a natural way of decentralization, where there are autonomous agents working as peers, or in teams, with their own behaviour and control. Mobile agents combine features from mobile code and software agents paradigms. However, in contrast to the Remote evaluation [42] and Code on demand [4] paradigms, mobile agents are active and choose to migrate between computers at any time during their execution. This makes mobile agent a powerful technology for implementing Ambient Intelligence environment systems and applications.

Agent migration is a mechanism that allows an agent running on a host to continue its execution on another location [13]. This process includes the transport of agent code, execution state and data of the agent. In an agent system, the migration is initiated by a request from the agent and not by the system. The main motivation for this migration is to access the services available in a given host, while reducing network load by accessing that host by local communication. Then migration is performed from a source agency where agent is running to a destination agency, which is the next location in the agent execution plan.

Along this paper we discuss the importance of security in mobile agent systems. In order to understand the role of security in the transmission of private and critical information through an open environment like Internet, it is common to use the following concepts that determine the diverse security levels: (i) Confidentiality is the property that ensures that only those that are properly authorised may be access the information. In the case of agents, the code; the execution state and the agent data (frequently called payload) may be considered confidential; (ii) Integrity is the property that ensures that information cannot be altered (referring to insertion, deletion or replacement of data). For agents, integrity of the code, execution state and data is usually required; (iii) Authentication is the property that refers to establishing the identity of an entity, but can also refer to ensuring the origin of some information. It provides a link between the information and its originator. Authentication applies to agent systems by requiring that agents are authenticated by the hosts or vice-versa, and by requiring that agent results are recognized as produced by the agent; (iv) Non-repudiation is the property that prevents some of the parts to negate a previous commitment or action. This is not so frequent for agent systems, and moreover, it is technically very difficult to provide.

In summary, 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 not conceivable in open environments such as Internet.

However, the focus of this paper is to provide a solution to the problem of the existence of 'malicious hosts'. A malicious host can be defined as an actor able to

execute an agent that belongs to another actor with the goal of attacking that agent in some way. The question of what action is considered an attack depends on the assurances agent owners need for their agents.

In order to achieve a protection level that is comparable to the one of the agents that run on trusted hosts, we must first identify and then prevent several attacks such as spying out code, manipulation of code, manipulation of the interaction with other agents, etc. For instance, in the case of manipulation of code, if the host is able to read the code and has access to the code memory, it can easily modify the program of an agent. The attack could be based on altering the code permanently, thus implanting a virus, worm or trojan horse, or it could also temporarily alter the behaviour of the agent on that particular host only. The advantage of the latter approach is that the host to which the agent migrates next will not detect any manipulation of the code since the copy it has received is not modified. Finally, spying out code is an attack based on the analysis of the code of the agent (which is usually available to the host). Although this access can be limited to the next instruction at a single point of time, this does not solve the problem since after several executions of the agent the hosts will have almost all of the code.

Along this paper we describe how to build a solution for the secure migration of agents, in order to solve the problem of malicious hosts. Software-only solutions for protection of software running on untrusted hosts has been proved to be impossible. For this purpose we propose a hardware based solution. Even if we do not aim at a complete protection, the main reason for using some hardware support is based on the fact that the degree of confidence in software-only security solutions depends on several factors that are frequently uncontrollable. Therefore this paper shows how to integrate a hardware based security solution in an agent standard development framework; in particular, JADE.

The remainder of this paper is organised as follows. In section 2, we briefly review previous work in the area of software protection and mobile agent security as well as providing a background on the areas of agent systems and trusted computing. In section 3, we present the Java Agent Development Framework security. Section 4 shows how the trusted computing technology can benefit the protection of the agents. In section 5, we briefly describe the secure migration protocol and its implementation as a library for agent systems. Section 6 presents the protocol validation using AVISPA. In section 7, we introduce the time-of-check time-of-use problem and provide a solution for it. We conclude and present some future work in section 8.

2. Related Work

In this section we provide a complete study of the background of the security in mobile agents. Firstly, we give list of the most relevant works in the protection of agents. Secondly we overview the current approaches to address the host protection. Additionally, we give a description of the *JADE* security model. Finally, we present

the *Trusted Platform Module(TPM)* with the most relevant features of this device and we introduce the importance of this hardware in the secure agent migration.

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 [27; 17].

Some protection mechanisms are oriented to the protection of the environments (host) against malicious agents. Among these, we found the *Software-Based fault isolation* [48] consisting on isolating application modules into distinct fault domains enforced by software, this technique is commonly refered as *SandBoxing* [14]. The idea behind the *Safe Code Interpretation* [34] 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* [15].

A different technique for protecting an agent system is signing code or other objects with a digital signature by means of which the authenticity and integrity 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.

Another technique known as *state appraisal* [11] is 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* [33; 5] is to mantain 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. The approach of the *Proof-carrying code* [32] 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 hosts. This problem was presented in the previous section, but in this section we briefly describe some partial approaches oriented to solve this problem. Among these ap-

proaches we find 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)* [51] 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 [5]. However, Yee noted that forward integrity could also be achieved using a trusted third party that performs digital time-stamping. Thus, a timestamp [18] 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 concern is the general availability of a trusted time-stamping infrastructure. A variation of this technique is the one named *Path Histories* [37], 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.

Itinerary Recording with Replication and Voting is a technique for ensuring that a mobile agent arrives safely at its destination [39]. 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 [47] 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 concussions as a trace summary of 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* [36] describes a scheme for allowing an agent of 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 [38] 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 encryption 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 [19] proposes the *Blackbox* technique. The strategy behind this technique is scramble 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 [44; 43] 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 [8]. Finally there are techniques that create a two-way protection. Some of these are based on the aforementioned protected computing approach [27].

Along this paper we propose a hardware based protection infrastructure that takes advantage of the recent advances in trusted hardware, and in particular of the specifications of the *Trusted Computing Group* (TCG), 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 (PCR)* 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.

3. Java Agent Development Framework

We are focused on providing solution to the secure agent execution problem. For this purpose, we designed a protection scheme, briefly described in this paper, a further description can be found in [30; 28]. However, we are concerned about agent-based systems developers whose security expertise is poor in most of cases. For this reason, the technology provided in this work is completely integrated in the JADE [21] (Java Agent Development Framework) framework. Prior to describe the core of our solution, we outline the poor security mechanisms provided by JADE showing the urgent necessity for the appropriate security mechanisms.

A JADE definition is given by a software Framework fully implemented in Java language. It simplifies the implementation of multi-agent systems through a middleware that complies with the FIPA specifications [12] and through a set of graphical tools that supports the debugging and deployment phases.

3.0.1. *Jade Security Model (JADE-S)*

JADE-S consists on a plug-in of *JADE* that allows to add some security characteristics in the development of *Multi-Agent Systems (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)* [22] allows to establish access permissions to perform certain operations on a set of predetermined classes, libraries or objects.
- *JCE (Java Cryptography Extension)* [24] 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)* [25] allows to exchange critical information through a network using a secure data transmission such as *Secure Socket Layer (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. In order to introduce security in such an open and distributed environment, *JADE-S* structures the agent platform as a multi-user environment, in which all components (agents, containers, etc.) belong to authenticated users. These are authorised by the administrator of the system to perform certain privileged critical actions. Then, in each platform there is a permissions file that contains the set of actions that each user is authorised 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.

As aforementioned, each component of the platform belongs to an authenticated user. Thus, 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 encrypted, following a scheme similar to the one used for Unix passwords. The 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. For each permission there is a list of related actions that may 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 encryption it with its private key (which is secret). Then, when the identity of an entity has to be checked, the signature may be unencrypted with the public key of the *Authority* (which is publicly known). Then 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.

In order to support secure communication between agents located in different hosts or containers, *JADE-S* uses the *SSL protocol* that provides privacy and integrity for all the connections established in the platform. This is a way of being protected against network sniffers.

After a first glance on the security mechanisms provided by *JADE*, we infer that these are not related with neither solving the problem of the malicious hosts at all nor solving the issues related with the malicious agent execution. Along next sections, we present a solution for the malicious hosts problem which has been fully integrated in the *JADE* framework to facilitate its usage for agent based systems developers.

4. The role of Trusted Computing in the Agent Protection

This section shows the importance of the Trusted Computing [46] technology in our approach for agent protection. Concretely, we take advantage of the recent advances in the Trusted Platform Module (TPM). 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 can not be tampered with). The TPM can be accessed directly via TPM commands or via higher layer application interfaces (the Trusted 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). For realizing these mechanisms, the TPM contains a crypto co-processor, a hash and an HMAC algorithm, a key generator, etc.

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 Platform Configuration Register (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, in order 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) can not be used outside the TPM that generated it. Some (like AIKs) can only be used for specific functions. Particularly interesting is 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 PCR the key is tied to contains a specific value). In order to prove the properties of a particular key, for example to prove 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 (e.g. 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 only entities that know the key's authorization value, can use the key.

Non-migratable keys are especially useful for preventing unauthorized access to some data stored on the platform. Binding such a key to specific PCR values and using it to encrypt data to be protected achieves two properties: The data can not 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).

Trusted platform modules (TPMs) can provide a variety of security functionalities. However, the TPM specification is highly complex and the deployment of TPM-based solutions is a difficult and delicate task. In this paper we propose the use of a hardware based protection mechanism that takes advantage of the recent advances in trusted hardware, especially of Trusted Computing Group (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 the 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. Previously we argued that is essential to integrate strong security mechanisms in agent systems world to achieve a reasonable security level to support real world applications.

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. Unfortunately, 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 take advantage of the TPM functionalities from software agent platforms. The main advantage of our 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.

Despite of the hardware-based solutions can be built on the foundations of different devices, we use the TPM in our solution because of it provides a number of relevant features. The most relevant appealing cryptography capabilities are the secure storage, the intimate relation to the platform hardware and the remote at-

testation. Our solution is based on the remote server attestation that uses TCG [44] technology, specifically the Trusted Platform Module(TPM) 'quote' function. This function creates a signature of the current platform software state. This state is reported through a log of software events, such as calling a higher software layer, starting a service, or reading a configuration file. These events are recorded as 'measurements', which are cryptographically protected by extending them into PCRs. Signing the PCRs effectively signs the event log.

This same technology that TPM utilizes can be applied in agent-based systems to scan machines for changes to their environments made by any entity before the system boots up and accesses the network. Additional reasons for the use of TC technology in our scheme are the support of a wide range of industries and the availability of computers equipped with TPM.

5. SecMiLiA: Secure Migration Library for Agents

SecMiLiA stands for Secure Migration Library for Agents. This section introduces the SecMiLiA, which consists on an extension to the JADE platform using the TPM4Java library. The TPM4Java library provides access to the Trusted Platform Module (TPM) functions. We have extended TPM4Java, as described in [30; 28] to be easily integrated in JADE. In order to provide a secure environment, SecMiLiA use the recent advances in Trusted Computing technology, as previous sections described. The design of SecMiLia has been driven by the following challenges:

- (i) The main objective consists on providing a secure environment where agents can execute and migrate in a secure way.
- (ii) The integration of TPM technology with JADE.
- (iii) The provision of mechanisms easy to use for programmers.
- (iv) The adaptability of the library, concretely we have designed the library with the idea in mind of change the hardware supporting it in the future. For instance, we are working on integrate smartcards as security hardware devices.
- (v) The flexibility and extendibility of the library interface.

Only achieving all these challenges, users will profit of SecMiLiA to solve a wide range of problems.

The most important function provided by the library is the secure migration mechanism. This mechanism is based on the remote attestation of the destination agency. Prior to the migration process, the library establish a secure environment for agent execution. This environment guarantees that malicious agents are not able to modify the host agency. The migration process has been integrated in the JADE platform by extending the Trusted Computing Base for agents from the initial agency to the destination agency before the migration takes place. One of the key points of this approach is that each agency must be TPM supported, so it must be possible their remote attestation. Similarly, each agency must provide

the functionality to allow to other agencies to take remote integrity measures to deter mine whether its configuration is secure. This second functionality is also supported by the TPM. The proposed library is based on a protocol that allows the two agencies involved in the migration to exchange information about their configurations. This exchange establishes the bases of the mutual trust between agencies.

5.1. *The Foundations: Secure Migration Protocol*

The first approach of a secure migration protocol can be found in[27]. This protocol provides the basis of protocol included in our approach. To start, we consider the fact that the agent trusts in its current agency to check the security of the migration process. The necessity of the TPM is justified since the TPM provides support to obtaining and reporting the configuration of an agency in a secure manner. Additionally, the protocol presented in [44] provides some keys taken into account to develop our protocol. In this case, this protocol shows how an agent from an agency uses a supporting TPM to produce a signature with the PCRs values and how to handle credentials. These credentials together with the signature are used to determine whether the destination configuration is secure. Finally, we have taken ideas from the protocol presented in [44]. The most relevant ideas from this third protocol are;

- (i) the use of an Attestation Identity Key *AIK* to produce the signature with the *PCR* values.
- (ii) the use of a Certification Authority *CA* that validates the *AIK*.
- (iii) the use of configurations to match the results from the remote agency.

Using ideas took from these three protocols, we have designed new protocol. The protocol presents a number of features, but the most important one is that the agency provides to the agent secure migration capacity. Besides, the agency uses a TPM that provides configuration values stored in *PCRs*. *TPM* is used to produce a signature with the PCRs values using a concrete AIK for each destination agency, in such a way that data receiver knows the identity of the TPM used to produce the signature. A Certification Authority generates the credentials to verify the AIK identity. Together with the signature the agency provides the AIK related credentials in such a way that the requester can correctly verify that the origin of the data is correct. The secure migration protocol is based on the CA based protocol described. However, the secure migration protocol implemented is adapted to the agent migration scenario, where an agent requests migration to a concrete destination agency. Thus, we detail every step carried out in the implementation of the protocol in *SecMiLiA*. A further description of this protocol is found in *SecMiLiA*. The process is described by an agent that requests to the source agency S the migration to destination agency D. The source agency S sends to destination agency a request for query. The destination agency accepts

this query and sends a nonce (random bits used to avoid replay attacks) as well as PCR values to know *(NONCEd, PCRId)*. The TPM Quote function is used to produce *(getConfig(NONCEs,PCRIs))*. This consists on the source agency requests to source TPM takes measures of the *PCR* together with the nonce all signed. The signature produced is *SIG(PCRs(PCRId), NONCEd,AIKsd)*. Then, *TPM* is used to produce *SIG(PCRs(PCRId), NONCEd,AIKsd)*.The source agency obtains *AIK* credentials from his credentials repository *AIKsd*, and sends to the Destination agency the signature produces. S sends AIK credentials, these contains the public key related to the private key used to sign these data. Besides S sends a nonce and *PCR* values to know *(SIG(PCDd(PCRId), NONCEs), AIKds, PCRd(PCRIs), CREDds)*. Therefore the Destination Agency validates the authenticity of the received key verifying the credentials by means of CA public key that generated those credentials *(verify(CREDsd, PCRs))*. Destination agency verifies the signature of PCR values and nonce received using the AIK public key *(verify(SIG(PCRs, PCRId), NONCEd,AIKsd))*. Also, the Destination agency verifies that *PCR* values *(check(PCRs,TrustSetd))* received belongs to the set of acceptable values and therefore source agency configuration is secure. A similar process is performed in the other way in order to perform a mutual attestation. Henceforth a mutual trustworthy exists between source and destination agencies. And hopefully the Source agency sends to agent the confirmation to migrate to the destination agency *(migrateto(DestinationAgency))*.

The secure migration protocol is based on the *Certification Authority (CA)* based protocol described. However, the secure migration protocol implemented is adapted to the agent migration scenario, where an agent requests migration to a concrete destination agency. Thus, we detail every step carried out in the implementation of the protocol in *SecMiLiA* in the Appendix. A further description of this protocol is found in [30; 28].

6. Protocol Verification with AVISPA

There are several ways in which a security protocol can be verified such as theorem proving, deductive methods, programming logic, modal logic and model checking [7]. Each of these methods has its own advantages and disadvantages. While logic-based and theorem proving methods can verify a security protocol fulfills the goals defined in its formal specification, they cannot generate attacks on that protocols. Model checking methods, on the other hand, will search for states of the system in which some properties are violated. Using model checking it can only be said that an attack is not possible for the specified system which states the number of principals, number of concurrent protocol runs, principal roles and so on. However, to validate our protocol this is enough. Thus, to formally analyze the proposed protocol, we use the AVISPA[2] protocol prover. This toolsuite provides a special language for describing security protocols and specifying their intended security properties.

AVISPA is an automatic push-button formal validation tool for Internet security

protocols, developed in a project sponsored by the European Union. 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.

When we deal with AVISPA we code in the 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 AVISPA operational semantic is based on the work of Lamport [26] on Temporal logic of Actions. 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 as well 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 as well 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.

The most relevant appeal in the use of AVISPA is that results in AVISPA are detailed and explicitly given with reachable number of states. Therefore, the interpretation of the results no requires a great expertise or skills in mathematics. On the contrary other tools like HERMES[20] require 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. Following, we show a complete description of the HLPSL code of the verification of the protocol, as well as an explanation of the results obtained from this verification. We refer to AVISPA website for more details.

6.1. *AVISPA coding for the validation of the secure migration protocol*

As mentioned, AVISPA uses HLPSL as specification language. Thus, 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 [20] where a great deal of experience is at least necessary to get meaningful conclusions.

In the HLPSL model of our protocol we define several roles; agent (modeling the agent itself), agency_S (modeling the source agency), agency_D (modeling the destination agency), TPM_agency_S (TPM in the source platform), TPM_agency_D (TPM in the destination platform) and environment.

Next code shows the behaviour of the agent, which can be in three different states, namely "0" to start migration, "1" migrating and "2" pass the control to other role (agency_S) .Thus the agent is modelled as follows, in the sate named "0" is waiting until reveive the *starting call*. When the execution is in state "1" is waiting until receive the migration call and pass the control to the agency_S role.

```
role agent(S,Ag:agent,SND,RCV:channel(dy)) played_by Ag
...
init
State := 0
transition
1.State = 0 /\ RCV(start) =|> SND(migrate.S) /\ State' := 1
2.State = 1 /\ RCV(migrate.Ag) =|> State' := 2
end role
```

The next role models the Source Agency in the Secure Migration Protocol above descibed by agency_S. This role is modeled

```
role agency_S(S,D,Ag,TPM_S:agent,SND,
RCV:channel(dy)) played_by S
...
init
TrustsetA := {pcrb(pcria)} /\ State := 0
```

```
transition
1.State = 0 /\ RCV(migrate.A) =|> SND(remoteattestrequest.D) /\ State' := 1
2.State = 1 /\ RCV(remoteattestagree.Nonceb'.Pcrib'.S) =|>SND(getconfig.Nonceb'.Pcrib'.TPM_S) /\ State' := 2
3.State = 2 /\ RCV(pcra(Pcrib).Nonceb.{hashf(pcra(Pcrib).Nonceb)}_inv(AIKab').AIKab'.TPM_S)=|>
Noncea':=new() /\SND(pcra(Pcrib).Nonceb.{hashf(pcra(Pcrib).Nonceb)}_inv(AIKab').
AIKab'.getCredentials(AIKab').Noncea'.pcria.D)/\State':=3
4.State = 3 /\RCV(Value'.Noncea.{hashf(Value'.Noncea)}_inv(AIKba').
AIKba'.getCredentials(AIKba').D) /\in(Value',trustseta) =|> SND(migrate.Ag)
end role
```

The Destination Agency of the protocol is modeled by means of the agency_D role, which behaviour is modeled by three different states.

```
role agency_D(S,D,TPM_D:agent,SND,RCV:channel(dy)) played_by D
...
init
TrustsetD := {pcra(pcrib)} /\ State := 0
transition
1.State = 0 /\ RCV(remoteattendrequest.D) =|> Nonceb':=
new() /\ SND(remoteattestagree.Nonceb'. pcrib.S) /\ State' := 1
2.State = 1 /\ RCV(Value'.Nonceb.{hashf(Value')}_inv(AIKab').AIKab'.getCredentials(AIKab').
Noncea'.Pcria'.D)/\in(Value',TrustsetD) =|> SND(getconfig.Noncea'.Pcria'.TPM_D) /\ State':=2
3.State = 2 /\ RCV(pcrb(Pcria).Noncea.{hashf(pcrb(Pcria).Noncea)}_inv(AIKba').AIKba'.TPM_D) =|>
SND(pcrb(Pcria).Noncea.{hashf(pcrb(Pcria).Noncea)}_inv(AIKba).AIKba'.getCredentials(AIKba').S)
end role
```

In the previous section, we detailed the secure migration protocol. There we described that each agency is TPM provided to implement the protocol described in section 5.1 order to model our solution based on the fact that both platforms are TPM provided we define these roles.

```
role tpm_Agent_D(S,TPM_S:agent,SND,RCV:channel(dy)) played_by TPM_S
...
init
State := 0
transition
1.State = 0 /\ RCV(getconfig.Nonceb'.Pcrib') =|> SND(pcra(Pcrib').Nonceb'.{hashf(pcra(Pcrib').Nonceb')}
_inv(aikab).aikab.S) /\ State' := 1
end role
```

Following the model of destination agency TPM role.

```
role tpm_Agent_S(S,TPM_S:agent,
SND,RCV:channel(dy)) played_by TPM_S
...
init
State := 0
transition
1.State = 0 /\ RCV(getconfig.Noncea'.Pcria') =|> SND(pcrb(Pcria').Noncea'.{hashf(pcrb(Pcria').Noncea')}
_inv(aikba).aikba.D) /\ State' := 1
end role
```

The next role describes the agent session modeled by the composition of the roles above explained.

```
role session(S,D,TPM_S,TPM_D,Ag:agent)
def=
local SND1,SND2,SND3,SND4,SND5,RCV1,RCV2,RCV3,RCV4,RCV5:channel(dy)
composition
agent(S,Ag,RCV1,SND1)/\ aagencyA(S,D,Ag,TPM_S,SND2,RCV3)/\ agency_D(S,D,TPM_S,SND3,RCV3)
/\ tpm_Agent_S(S,TPM_S,SND4,RCV4) /\ tpm_Agent_D(D,TPM_D,SND5,RCV5)
end role
```

Finally, next role is coded to describe the full process. We named environment to this role because of it is the main role that uses the rest of roles previously described. We aim the intruder knowledge in which we explicitely put all the information that an intruder could use.

```
role environment()
def=
const
s,d,tpm_s,tpm_d,ag:agent
intruder_knowledge = {s,d,tpm_s,tpm_d,ag,hashf}
composition
session(s,d,tpm_s,tpm_d,ag)
end role
environment()
```

6.2. *Results interpretation*

The execution of the protocol as described above produces the following results with different inputs. To perform this validation we described a set of goals. These goals are given by the integrity of several exchanged values. Concretely, it is essential to have integrity in the values of the PCRs and in the attestation keys. Thus, we check the following values pcra(pcrib), pcrb(pcria), aikab and aikba.

It is important to mention the fact we use the Dolev-Yao intruder model [10] to model the attacker and the environment. In this intruder model the attacker has full control over all messages that are sent over the network. Therefore the attacker can analyze , intercept, or modify messages to any party.

```
...
GOAL
as_specified
BACKEND
OFMC
COMMENTS
STATISTICS
parseTime: 0.00s
searchTime: 3.23s
visitedNodes: 4 nodes
depth: 2000 plies
environment()
```

These results show how the summary of the protocol validation is safe. Concretely, some statistics are shown referred to a checking of 2000 plies of depth. However, this process has been performed for depths of 250, 300, 400, 500, 1000 and 2000 with similar results. We aim that the ATTACK TRACE section is completely empty. This fact means that no attacks were found for this protocol. Evidently, it does not means that no exists any against this protocol.

7. CA-based protocol against a Direct Anonymous Attestation protocol

In this section we present an alternative to the protocol above described based on the use of a Certification Authority. Thus, we describe the direct anonymous attestation protocol in detail and we compare both protocols.

7.1. *Direct Anonymous Attestation (DAA)*

Direct Anonymous Attestation (DAA) is a cryptographic protocol which enables the remote authentication of a trusted platform whilst preserving the user's privacy. The protocol has been adopted by the Trusted Computing Group (TCG) in the latest version of its Trusted Platform Module (TPM) specification[44] as a result of privacy concerns. The DAA protocol is based on three entities and two different steps. The entities are the TPM platform, the DAA Issuer and the DAA verifier. The issuer is charged to verify the TPM platform during the Join step and to issue DAA credential to the platform. The platform uses the DAA credential with the verifier during the Sign step. Through a zero-knowledge (In cryptography, a zero-knowledge proof or zero-knowledge protocol is an interactive method for one party to prove to another that a (usually mathematical) statement is true, without revealing anything other than the veracity of the statement.) proof the verifier can verify the credential without attempting to violate the platform's privacy. DAA also supports anonymity revocation: at the time a TPM proves that it is certified, a third (passive) party can be designed who will later on be able to identify the TPM in case of misuse.

The Direct Anonymous Attestation protocol (DAA) represents an alternative to the use of Certification Authorities for determining whether a TPM signed a message and which TPM did it is. In contrast to the previous protocol described in Section 5.1. Here instead of using a CA, an issuer certificate authority is used for DAA keys.

Algorithm 1 Direct Anonymous Attestation protocol

1: Source Agency (S) sends getConfig() to the Source TPM (tpms)
2: tpms produces SIG(PCRs,AIKs),PCRs and sends it to S
3: S sends the remoteAttest(SIG(PCRs,AIKs),PCRs,AIKs,PROOFs) to the Destination Agency (D)
4: D verify(PROOFs)
5: D verify(SIG(PCRs,AIKs)
6: D check(PCRs,TrustSetD)
7: D sends getConfig() to the Destination TPM (tpmd)
8: tpmd sends SIG(PCRid,AIKd),PCRid to D
9: D sends to D attestResults(SIG(PCDd,AIKd),PCDd,AIKd,PROOFd)
10: S verify(PROOFd)
11: S verify(SIG(PCDd),AIKd)
12: S check(PCDd,TrustSetS)

This protocol is following described; S requests to source TPM signed configuration data using 'getConfig()'. Source TPM produces the signature 'SIG(PCRs,AIKs),PCRs' and sends it to S. Then the source agency S sends to destination agency D configuration data signed together with AIK public key and a cryptographic proof 'remoteAttest(SIG(PCRs,AIKs),PCRs,AIKs,PROOFs)'. Then an AIK certificate is generated using a DAA key 'AIKs', as well as a certificate for DAA key is generated for a DAA certificate issuer 'PROOFs'. Thus, D verifies the cryptographic proof and the AIK validity with 'verify(PROOFs)'. D ver-

ifies received signed data using AIK public key 'verify (SIG(PCRs,AIKs))' and verifies that received configuration values belong to the set of acceptable values 'check(PCRs,TrustSetD)', and therefore source agency configuration S is secure. The process is the same for the opposite that is the D requests and so on.

7.2. *CA-based protocols vs DAA protocol*

We study the comparison between the DAA protocol described in this section and the protocol based on the use of an external Certification Authority presented in previous sections. Both protocols allow to the verifier entity to trust that AIK used to sign configuration values belongs to the entity that generated the signature. Additionally the AIK signature proves that there is a TPM installed. However, CA based protocol presents several disadvantages :(i)A bottleneck is that is necessary to create a certificate for each AIK. And (ii) the verification entity and CA should be confabulated to violate the security of the system.

DAA protocol does not present such disadvantages but DAA protocol needs an additional protocol to the entity verification to determine that the verified entity has an AIK certificate. For implementing the secure migration library for agents we decided to use the TPM4java library, since this does not provide the DAA functionality, we implemented a CA based protocol. From these protocols, except Anonymous Direct Attestation protocol, we design a new protocol gathering all advantages of them. Following all features of our protocol are described:

(1) Agency provides to agent the capability to migrate securely.
(2) TPM provides to the agency the PCR configuration values.
(3) TPM signs PCR values using a concrete AIK for each destination agency in such a way that data receiver knows securely the TPM identity used to sign.
(4) A CA generates credentials to verify correctly the AIK identity.
(5) Agency provides AIK credentials together with PCR values signed to allow the requester verify that data comes from a trusted agency, that is, from the TPM.

Among the main advantages of the protocol based on the use of a CA, we highlight the control of agent identities in the system. Nevertheless, it is needed the installation of a CA in the system. Indeed, the management and issue of a lot of keys and certificates imply an overload in the communication channel. The main disadvantage of using a Certification Authority based library is that to generate a certificate for each key used for the attestation, we need to make a request to the CA, and this may cause the system to slow down as many requests are performed. We also found out that the security of the attestation protocol can be violated if the verification authority and the CA act together. We have to consider the point that the use of these technologies is to provide security to agent technology, but we use the agent technology for reduce the communications between different parts of the same system, in the opposite to the initial appeal of the mobile agent computing.

DAA based protocol presents several advantages. Evidently, the advantages of

the use of the anonymous direct attestation protocol are among others that, as the certificates issuer entity and the verifier entity cannot act together to violate the system security, then one only entity can play both roles, as verifier and issuer of certificates. Also, the certificates only need to be issued once, so we solve the slow down problem mentioned above. With this we see that anonymous attestation protocol is an interesting option to take into account as attestation method. We have not to install a CA with the keys and certificates related. Besides it is the same philosophy of mobile agents. However, the use of a direct anonymous attestation scheme implies the anonymity of the identity of the platforms. Thus this problem requires a further study due to this fact implies a decreased level in the security of the system.

8. Time-of-Check Time-of-Use problem

The solution presented in previous sections only enables the state of the destination Agency to be verified until a concrete moment. Obviously, we can warrant that the attestation agency is in the same state but there are no guarantees that the destination Agency stays in the attested state after it was verified but prior to the agent arrival.

The next example illustrates the magnitude of this problem. Let us suppose that there is an attacker who owns a trusted agency, which is TPM-provided, and another manipulated agency. We assume that the attacker trusted agency is the next destiny in the agent plan. The attack is based on the fact that after the verification performed of the destiny any more verification is done. Therefore a non trusted agency could replace the trusted agency. In this case the agent might be in the domain of the attacker and this might be manipulated.

Henceforth a malicious user can change the trusted agency by a tampered one. This new agency could manipulate the agent obtaining some kind of benefit. In an attempt to solve this problem, we propose an approach based on the self protection of the agent code. The foundations of this solution are based on the encryption of part of the agent code with a key generated in destination TPM.

We make use of the *sealed storage* capability of the TPM. At an abstract level, the TPM presents a simple interface for binding data to the current platform configuration (as defined by the PCRs) via the *seal* operation. The Seal command takes a set of PCR indices as input, and encrypts the data provided using its Storage Root Key (Kroot), a key that never leaves the TPM. It outputs the resulting text encrypted C, along with an integrity-protected list of the indices provided and the values of the corresponding PCRs at the time Seal was invoked.

Nevertheless this is not a regular encryption key, this key is bound to a concrete set of PCR values, in such a way that if these values do not match with the PCR values the execution of the agent is not granted. The TPM does not decrypt the encrypted part of the agent code, thus since that the key is needed in destination agency D to decrypt the agent, prior to be executed if the status change it is not

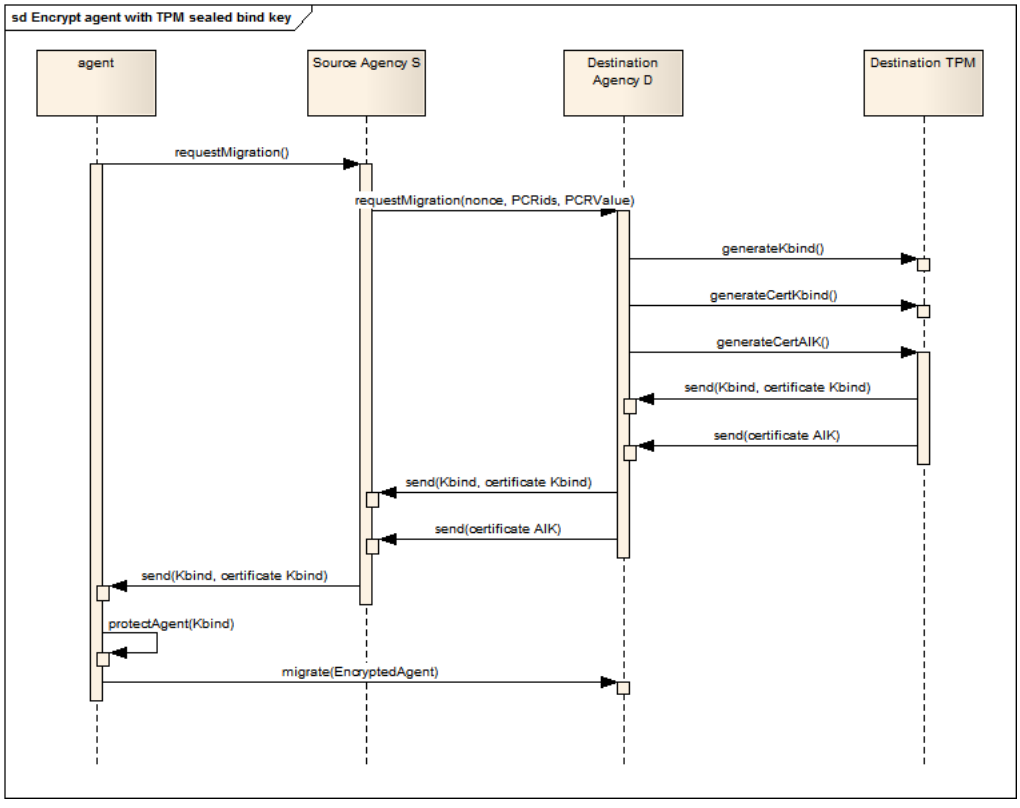

Fig. 1. Sealed-Bind Key based Protocol

possible. Figure1 depicts every step of the process.

A detailed description of the process is the following: The agent *a* requests for migration from the *source agency (S)* to the *destination agency (D)*. Evidently both agencies are TPM provided. Let us assume that *D* requests the *AIK certificate.* The protocol is described as; agent plans to migrate to *D*, the agent sends *requestMigration()* to the *S* and this sends this message to the *D*. *S* sends a randomly value nonce together with the set of *PCRs* and their values. *D* uses the destination TPM to generate a *sealed-binding key* (*generate (KbindPCR)*) and the related certificate to assure the use of the key *(generate(certificate Kbind)*). This is the definition of the "time-of-check time-of use" problem. Afterwards all data are sent to the source agency. These data are provided to the agent to encrypt part of its own code to solve posed problem. When the agent actually migrates part of the agent is encrypted with Kbind, the only way that this agent can be executed is by means of decrypting the encrypted code using the related Kbind. If the status of the selected PCRs changed it is produced by the execution of non allowed software then the new Kbind generated is different. Therefore the destination agency cannot execute

the agent. Obviously this agency cannot modify or alter the agent code to obtain benefits due to its encrypted.

(1) a request migration to S.
(2) S request migration to D.
(3) S sends (nonce, PCRids,ValuePCR) to D.
(4) D calls to the TPM_CreateWrapKey() function to produce the KBindPCR.
(5) D generate certificate KBindPCR
(6) T is used to generate the KBindPCR and sends(KBindPCR) to D.
(7) KBindPCR is sent to D.
(8) D sendsCertificate KBindPCR to S.
(9) D sendsCertificate AIK to S.

This mechanism takes advantage of the sealed bind key generation functionality of the TPM. This operation consists on binding messages to a concrete status of the platform. In such a way the message can only be decrypted, while decryption platform keeps the status. This operation links a message with a set of PCR values and an asymmetric and non-migratable key. By means of this mechanism, we achieve that the destination agency only can have the agent in clear if the TPM uses the related encryption key, and thus the PCRs values are the expected.

Obviously, the most important improvement of this protocol consists on the avoiding of the problem of the problem known as the *time-of check time-of-use*. This solution obligates to both platforms to keep the same status in all the process. Obviously this open an interesting discussion about the restriction of this solution, but it is further described in the future work section.

8.1. *AVISPA coding for the validation of sealed-bind key protocol*

In previous sections we showed the HLPSL model of the protocol based on CA. Here we present the HLPSL model of the protocol based on the sealed-bind key. The HLPSL code is composed of these roles; agent, agency_S, agency_D, TPM_agency_S , TPM_agency_D and the environment role. The agent role is exactly the same that in the previous protocol. Following we describe the HLPSL code for the model of the Agencies. We notice the Kbind key, which is the sealed bind key generated in the TPM.

```
role agency_S(S,D,Ag,TPM_S:agent,SND,RCV:channel(dy)) played_by S
...
init
TrustsetA := {pcrb(pcria)} /\ State := 0
transition
1.State = 0 /\ RCV(migrate.A) =|> SND(remoteattestrequest.D) /\ State' := 1
2.State = 1 /\ RCV(remoteattestagree.Nonceb'.Pcrib'.S) =|>
SND(getconfig.Nonceb'.Pcrib'.TPM_S) /\ State' := 2
3.State = 2 /\ RCV(pcra(Pcrib).Nonceb.{hashf(pcra(Pcrib).Nonceb)}_inv(AIKab').AIKab'.TPM_S)=|>
Noncea':=new() /\SND(pcra(Pcrib).Nonceb.{hashf(pcra(Pcrib).Nonceb)}_inv(AIKab').
AIKab'.getCredentials(AIKab').Noncea'.pcria.D)/\State':=3
4.State = 3 /\RCV(Value'.Noncea.{hashf(Value'.Noncea)}_inv(AIKba').
AIKba'.getCredentials(AIKba').Kbind'.getCredentials(Kbind').D) /\in(Value',trustseta) =|> SND(migrate.Ag)
end role
```

```
role agency_D(S,D,TPM_D:agent,SND,RCV:channel(dy)) played_by D
...
init
TrustsetD := {pcra(pcrib)} /\ State := 0
transition
1.State = 0 /\ RCV(remoteattendrequest.D) =|> Nonceb':=
new() /\ SND(remoteattestagree.Nonceb'. pcrib.S) /\ State' := 1
2.State = 1 /\ RCV(Value'.Nonceb.{hashf(Value')}_inv(AIKab').AIKab'.getCredentials(AIKab').
Noncea'.Pcria'.D)/\in(Value',TrustsetD) =|> SND(getconfig.Noncea'.Pcria'.Kbind'.TPM_D) /\ State':=2
3.State = 2 /\ RCV(pcrb(Pcria).Noncea.{hashf(pcrb(Pcria).Kbind'.Noncea)}_inv(AIKba').AIKba'.
TPM_D) =|> SND(pcrb(Pcria).Noncea.{hashf(pcrb(Pcria).Kbind'.Noncea)}_inv(AIKba).AIKba'
.getCredentials(AIKba').Kbind'.getCredentials(Kbind).S)
end role
```

Similarly that the two previous roles, in the destination TPM role we have the Kbind key generated by the destination TPM.

```
role tpm_Agent_S(S,TPM_S:agent,
SND,RCV:channel(dy)) played_by TPM_S
...
init
State := 0
transition
1.State = 0 /\ RCV(getconfig.Noncea'.Pcria') =|> SND(pcra(Pcrib').Nonceb'.
{hashf(pcra(Pcrib').Kbind'.Nonceb')}_inv(aikab).aikab.Kbind.D)
 /\ State' := 1
end role
```

The next role model sessions of the agent.

```
role session(S,D,TPM_S,TPM_D,Ag:agent)
def=
local SND1,SND2,SND3,SND4,SND5,RCV1,RCV2,RCV3,RCV4,RCV5:channel(dy)
composition
agent(S,Ag,RCV1,SND1)/\ aagencyA(S,D,Ag,TPM_S,SND2,RCV3)
/\ agency_D(S,D,TPM_S,SND3,RCV3)/\ tpm_Agent_S(S,TPM_S,SND4,RCV4)
/\ tpm_Agent_D(D,TPM_D,SND5,RCV5)
end role
```

Finally, next role is coded to describe the full process. We named environment to this role because of it is the main role that uses the rest of roles previously described.

```
role environment()
def=
const
s,d,tpm_s,tpm_d,ag:agent
intruder_knowledge = {s,d,tpm_s,tpm_d,ag,hashf}
composition
session(s,d,tpm_s,tpm_d,ag)
end role
environment()
```

The results of the execution of this code are very similar to the produced with the previous protocol. We aim the importance of the empty ATTACK TRACE section, which means that any attack was found for this protocol with the intruder knowledge input.

9. Conclusions and Future Work

In this paper we have presented a new protection infrastructure solving the problem known as malicious host in mobile agent systems. This protection infrastructure is used to support the secure migration of agents by means of a hardware-based a secure protocol. In particular, the protocol uses functionality provided by Trusted Platforms Modules (TPM). We have also described the implementation of the proposed infrastructure, which is materialized an extension to the Java Agent DEvelopment framework (JADE). Finally, we have presented the validation of the protocol using the Automated Validation of Internet Security Protocols and Applications (AVISPA) framework. With regards to our future work, we are working on adding to the secure migration services the capacity to attend concurrent requests. The actual implementation of this in the library provides secure migration to a remote container, but it is able to handle only one request at the same time. While this request is attended, further requests are refused until the migration is not ended. This happens because of details with the way the TPM manages the keys. The solution we are developing will allow the library to handle concurrent secure migration requests. Several fields are open for future work for research. An interesting point is the study of the concurrent capacity for the secure migration services to attend concurrent requests. The current implementation of this in the library provides secure migration to a remote container, but it is able to handle only one request at the same time. After this request is attend, further requests are refused until the migration is not ended. This happens because of the TPM management of this problem and also because of the keys management problem mentioned before. To improve this, the library could provide the ability to handle concurrent secure migration requests. And to do so, it is necessary to develop the aforementioned cache of keys to provide the multiple keys storage capability to the TPM. We find several difficulties related to key management in this improvement. The system could be in a state attending several requests using different keys, so it has to manage the keys to provide their availability when needed. And we can also find the situation where any key can be replaced as all the keys are being used, in this case new requests might wait until its key is loaded.

Finally, a further study of different approaches to implement a semantic attestation can be done. The main motivation for this is the huge number of possible trusted configurations than can exist, and this number is even bigger if the sequence in the software installed is a concern. Thus, instead of store remotely all the possibilities the system should attest every software independently. Nevertheless, this implies that some collaborative attack can be performed.

Let us assume that we work in highly a restricted operation environment. Thus all software executed in the computer is monitored by the operating system. To avoid the case that another agency with illicit purposes is executed, we forbid the simultaneous execution of any other software except the trusted agency and the operating system. To do that our operating system takes part of the chain of trust,

in such a way that a PCRExtend operation is done to take measures of PCRs to validate in the next execution. Thus the OS is part of the chain of trust of the TPM.

Indeed, this represents a hard restriction in the environment. Especially, in versatile systems like agent based ones this is not desirable. For that reason we propose different ways to improve the flexibility keeping the security degree. Therefore we decide to list all trusted software that can execute simultaneously in our system. Of course, this raises new problems. More relevant problem consists on different combinations of software exists, especially if the sequence matters, since sequence is marked in PCRs when are measures by the operating system. Nevertheless, this mechanism allows posing QUOTE values validation that fulfils with the chain of trust.

However this solution is restricted as well, because of the software to use is hardly limited, this is not desirable. A possible improvement is based on avoiding that two agencies execute simultaneously. We only have to measure the set of PCRs modified when we run a non trusted agency. Even a further approach is based on taking measures of PCRs values and find out which are the values for trusted agencies and non-trusted agencies respectively.

An important limitation of TPM-based protections, including sealed storage, is that the TPM only provides a guarantee at software load time. Thus, if a legitimate application is loaded and then modified in memory, the TPM will not be aware of the change. In other words, the TPM does not protect against an attacker who exploits a flaw in currently executing software. Also, as noted above, the legitimate software must be modified to extend measurements into the correct PCRs, and secrets must be sealed under the correct PCRs.

10. Appendix

In previous sections we described the foundations of the SecMiLiA, the Secure Migration Protocol. The most relevant ideas from this protocol are the use of an AIK to produce the signature of the PCR values, 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 to migrate by a secure way. 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.

The next protocol describe every step of this protocol.

Algorithm 2 Secure migration protocol

1: Agent Ag requests to his agency the migration to Agency B.
2: Source Agency S sends to Destination Agency D a request for attestation.
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 the TPM installed in this platform TPMS PCR values requested by D together with the nonce all signed.
5: TPMS returns requested data.
6: S obtains AIK credentials from its credentials repository.
7: S requests D 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 belongs to the set of accepted values and them the agent configuration is valid.
11: D requests to the TPM installed in the destination platform TPMD the PCR values requested by the D together with the nonce signed.
12: TPMD 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 belongs to the set of accepted values and then the D configuration is secure. From this point trustworthy between S and D exist.
18: Then S allows to the agent Ag the migration to D.

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. This work has been developed in the GISUM research group from the Languages and Computer Science Department in the University of Malaga, Spain.

References

Alechina, N., Alechina, R., Habner, J., Jago, M., Logan, B., *"Belief revision for AgentSpeak agents"*. In the Proc Of Autonomous Agents and Multi Agents Systems 2006. Hakodate, Japan. ISBN:1-59593-303-4, p1288 - 1290.

"AVISPA Project". Online available at *http://www.avispa-project.org*. the AVISPA project homepage.

Bennet S. Yee,. *"A Sanctuary for Mobile Agents"*. Secure Internet Programming 1999.

Brooks, R.R. *"Mobile code paradigms and security issues"*. Internet Computing, IEEE. Vol. 8, Issue 3. 2004 pp:54-59.

Chess, D., Grosof, B., Harrison, C., Levine, D., Parris, C., and Tsudik, G. *"Itinerant Agents for Mobile Computing"*. IEEE Personal Communications, vol.2, no.5, October 1995, pp 34-49.

Clements, P., Papaioannou, T., Edwards, J., *"Aglets: Enabling the Virtual Enterprise"*. In the Proc. Of Managing Enterprises Stakeholders, Engineering, Logistics and Achievement (ME-SELA'97) ISBN 1 86058 066 1, p425.

Coffey, T., Dojen, R. and Flanagan, T., *"On Different Approaches to Establish the Security of Cryptographic Protocols"*, Proceedings of SAM'03 (Conference on Security and Management), Col. II, Las Vegas, USA, 23-26 June 2003, Pages 637-643, ISBN

1-932415-18-1. Ed. H.R.Arabnia and Y. Mun.

Collberg, C., Thomborson, C. *"Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection"*. University of Auckland Technical Report 170. 2000.

"Cougaar project". Online available at *http://cougaar.org/*.

Dolev, D., Yao, A. *"On the security of public key protocols"*. In proceedings of the IEEE 22nd Annual Symposium on Foundations of Computer Science, pp 350-357. 1981.

Farmer, W., Guttman, J., and Swarup, V. *"Security for Mobile Agents: Authentication and State Appraisal"*. Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS '96), September 1996, pp 118-130.

"FIPA:Foundation for Intelligent Physical Agents". Available on line at: http://www.fipa.org/.

General Magic, Inc. "The Telescript Language Reference", online available at *http://www.genmagic.com-Telescript.*

Gosling, J., Joy, B., and Steele, G.. *"The Java Language Specification"*. Addison-Wesley, 1996.

Gray, R. *"Agent Tcl: A Flexible and Secure Mobile-Agent System"*. Proceedings of the Fourth Annual Tcl/Tk workshop (TCL 96), pp 9-23, July 1996.

Gunter C. A., Homeier P., Nettles S. *"Infrastructure for Proof-Referencing Code"*. In Proceedings, Workshop on Foundations of Secure Mobile Code, March 1997.

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.

Haber, S., and Stornetta, S. *"How to Time-Stamp a Digital Document"*, Journal of Cryptology, vol. 3, pp. 99-111, 1991.

Hohl, F. *"Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts"*. G. Vigna (Ed.), Mobile Agents and Security, pp 92-113. Springer-Verlag, LNCS No 1419, 1998.

Hussain, M., Seret, D., *"A Comparative study of Security Protocols Validation Tools: HERMES vs. AVISPA"*. Published in Proceecings of International Conference on Advanced Communication Technology (ICACT) 2006. IEEE Communication Society.

"JADE". Online available at *http://jade.tilab.com/*.

S. Microsystems, *"JavaTM Authentication and Authorization Service (JAAS)"* vol. 2001, http://java.sun.com/products/jaas/.

"JAVACT project". Online available at *http://www.irit.fr/recherches/ISPR/IAM/JavAct.html.*

"AIK Java Crytology," 2001, http://jcewww.iaik.at/.

"Java Secure Socket Extension (JSSE) Reference Guide", http:// java.sun.com/ j2se/1.4.2/ docs/ guide/ security/jsse/JSSERefGuide.html.

Lamport, L. "Pasword authentication with insecure communication". In Communications ACM, vol.24, pp 770-720, NY, USA, 1981. ACM Press.

Maña, A. *"Software Protection based on smartcards"*. PhD Thesis. University of Mlaga. 2003.

Maña, A., Muñoz, A. *"A Hardware based Infrastructure for Agent Protection"*. In proceedings of the 3rd Symposium of Ubiquitous Computing and Ambient Intelligence, in University of Salamanca, Spain, 22-24th October, 2008. Advances in Soft Computing, Springer Verlag , Vol. 51 Corchado, Juan M.; Tapia, Dante; Bravo, Jose (Eds.) 2009, XII, 354 p. 131 illus., SoftcoverISBN: 978-3-540-85866-9.

Maña, A., Muñoz, A., Serrano, D. " Towards Secure Agent Computing for Ubiquitous Computing and Ambient Intelligence". In proceedings of the 4th UIC. Published by

Springer-Verlag. LNCS, ISSN 0302-9743.

Muñoz, A., Maña, A., Serrano, D. *"SecMiLiA: An Approach in the Agent Protection"*. In the proceedings of the International Conference on Availability, Reliability and Security, ARES 2009, published by IEEE Computer Society Press.

Mouratidis H, Kolp M, Faulkner S, Giorgini P,. "A Secure Architectural Description Language for Agent Systems". AAMAS'05 July 25-29, Utrecht, Netherlands.

Necula G. *"Proof-Carrying Code"*. In Proceedings of 24th Annual Symposium on Principles of Programming Languages. 1997.

Ordille, J. *"When agents Roam, Who can You Trust?"*. Proceedings of the First Conference on Emerging Technologies and Applications in Communications, Portland, Oregon, May 1996.

Ousterhout, J. *"Scripting: Higher-Level Programming for the 21st Century"*. IEEE Computer, March 1998, pp 23-30.

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.

Riordan, J., and Scheneider, B. *"Environmental Key Generation Towards Clueless Agents"*. G.Vigna (Ed). Mobile Agents and Security, Springer-Verlag, LNCS No 1419, 1998.

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

Sander, T., and Tschudin, C. *"Protecting Mobile Agents Against Malicious Hosts"*, in G. Vigna (Ed.), Mobile Agents and Security. Springer-Verlag, LNCS No. S225 711, August 1983.

Scheider, B. *"Towards Fault-Tolerant and Secure Agency"*. Proceedings 11th International Workshop on Distributed Algorithms, Saarbucken, Germany, September 1997.

Schwuttke, U. M., and Quan, A. G. 1993. *"Enhancing Performance of Cooperating Agents in Real-Time Diagnostic Systems"*. In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93), 332337. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Shepherdson, D., ''*The JACK Usage Report"*. In the Proc Of. Autonomous Agents and Multi Agents Systems 2003 (AAMAS 03).

Stamos, J., and Gifford, D. *"Remote Evaluation"*. ACM Transactions on Programming Languages and Systems (TOPLAS) archive. Vol, Issue 4. 1990. pp 537 - 564.ISSN:0164-0925.

Stern, J. P., Hachez, G., Koeune, F., Quisquater, J. J. *"Robust Object Watermarking: Application to Code"*. In Proceedings of Info Hiding '99, Springer-Verlag. LNCS 1768, pp. 368-378, 1999. http://www.dice.ucl.ac.be/crypto/publications/1999/codemark.pdf.

Trusted Computing Group. *"TCG TPM Specification Version 1.2"*. Parts I-III, 2005.

Trusted Computing Group. *"TCG Specification Architecture Overview, Revision 1.4"* (2007), https://www.trustedcomputinggroup.org/groups/TCG 1 4 Architecture Overview.pdf

"TPM Main - Part 1 Design Principles, Version 1.2, Revision 103." TCG Technical Specification, July 2007.

Vigna, G. *"Secure Recording of Itineraries Through Cooperating Agents"*. Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations, pp 147-154, INRIA, France 1998.

Wahbe, R., Lucco, S., Anderson, T. *"Efficent Software-Based Isolation"*. Proceedings of the Fourteenth ACM Symposium on Operating Systmes Principles. ACM SIGOPS Operating Systems Review, pp 203-216. Dec 1993.

Wang, H., and Wang, C. 1997. *"Intelligent Agents in the Nuclear Industry"*. IEEE Computer 30(11): 2834.

Wooldrigde, M. *"Agent-based Software Engineering"*. In IEE Proceedings on Software Engineering, 144(1), pages 26-37, February 1997.

Yee, B. *"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*.