Introducing Trusted Third Parties to the Mobile Agent Paradigm

Uwe G. Wilhelm[1], Sebastian Staamann[1], and Levente Buttyán[2]

[1] Laboratoire de Systèmes d'Exploitation
[2] Institut pour les Communications informatiques et leurs Applications
Ecole Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland

{Uwe.Wilhelm, Sebastian.Staamann, Levente.Buttyan}@epfl.ch

Abstract. The mobile agent paradigm gains ever more acceptance for the creation of distributed applications, particularly in the domain of electronic commerce. In such applications, a mobile agent roams the global Internet in search of services for its owner. One of the problems with this approach is that malicious service providers on the agent's itinerary can access confidential information contained in the agent or tamper with the agent.
In this article we identify trust as a major issue in this context and propose a pessimistic approach to trust that tries to prevent malicious behaviour rather than correcting it. The approach relies on a trusted and tamper-resistant hardware device that provides the mobile agent with the means to protect itself. Finally, we show that the approach is not limited to protecting the mobile agents of a user but can also be extended to protect the mobile agents of a trusted third party in order to take full advantage of the mobile agent paradigm.

1 Introduction

New approaches for distributed computing based on mobile agent technology, such as *Aglets*, *Telescript*, or *Voyager* become ever more pervasive and are considered as innovative new ideas to structure distributed applications. A particularly interesting and perhaps economically important class of applications, to which mobile agents seem well adapted, is electronic commerce.

A typical use of mobile agents in the domain of electronic commerce includes the scenario, in which an agent roams the global Internet in search of some service for a user (the owner of the agent). Such a service can have many different forms, for instance, the provision of a physical good, the execution of a search for an information item, or the notification of the occurrence of some event. The agent is configured by the user with all the relevant information about the desired service, the constraints that define under which conditions an offer from a service provider is acceptable, and a list of some potential providers of the service. It will then migrate to the sites of these service providers in order to locate the best offer for the service sought by the user and finalize the transaction with the chosen service provider.

Since an agent is vulnerable when it is executing on the execution platform of a service provider, it is necessary that the user obtains some guarantees concerning the protection of his agents. Consider a mobile agent that holds data for one or several payment methods, which it needs to finalize a purchase. These payment data should not be available to any principal other than the one that actually provides the service and, thus, is entitled to receive the payment. A malicious service provider might try to obtain the data of the payment method without providing the service or might otherwise tamper with the agent in order to trick it into accepting the malicious provider's offer (e.g., by removing some information about a better offer from the memory of the agent). The usual approach that is taken to provide a user with certain guarantees concerning the protection of his agents, is to assume that the service providers are *trusted* principals [13] or to create a mechanism that enables the user to detect which of the providers on the itinerary have misbehaved [21].

The notion of trust has long been recognized as being of paramount importance for the development of secure systems [6, 10, 28]. For instance, any conceivable system for authenticating users needs trusted functionality that holds the necessary authentication information (see e.g., [18, 26]). Yet, the meaning that is associated with trust or the notion of a trusted principal is hardly ever clearly defined in these approaches and the reader is left with his intuition.

In this article we address the question of how trust in a certain principal can be motivated based on technical reasoning and present a pessimistic approach to trust that tries to prevent malicious behaviour rather than correcting it after it has occurred. The approach relies on a trusted and tamper-resistant hardware device that can be used to enforce a policy. If this policy is properly chosen, an agent can take advantage of it in order to protect itself as well as the information it contains from possibly malicious service providers.

The mobile agent paradigm usually identifies two interacting principals: the owner of the agent who configures it and the executor of the agent, which may be identical to the service provider. However, many protocols for security related problems, especially those that are concerned with non-repudiation, require an additional third party that often has to be trusted by the other parties. This principal is called a *trusted third party* (TTP). Due to this trust requirement, the functionality of a TTP must be realized in a trustworthy environment, which is usually only available at the site of the TTP. Hence, the principals in the mobile agent paradigm have to interact with the TTP server using the classical client/server mechanisms. The approach described here, which provides protection for the mobile agents of regular users, can also be applied to mobile agents that are owned by a TTP. This allows us to take full advantage of the mobile agent paradigm and removes the need for remote messaging between the interacting principals.

In the following Section 2, we introduce our model for mobile agents and point out the problems related to trust within this model. Then, in Section 3, we discuss the notion of trust and define its relation to policy, which enables us to better assess the possible motivations for trust. In Section 4, we introduce

a trusted and tamper-resistant hardware device and a protocol, which allow us to define certain guarantees for the execution of agents. In Section 5, we show how this approach can be used to protect the agents of a regular user as well as those of a TTP. In Section 6, we discuss why we consider this to be an adequate way to approach the problem and what effects this has on the notion of open systems. Finally, Section 7 concludes the paper with a summary of the main contributions.

2 The mobile agent paradigm

The mobile agent paradigm has been identified by many authors as a promising and innovative approach to structure problems in distributed computing [3, 4, 7, 8, 11, 23]. However, it is still under lively discussion and it has been shown in [9] that there is no single compelling reason to favour the mobile agent paradigm over classic client/server approaches. On the other hand, the same authors point out that the mobile agent paradigm provides interesting solutions to many real-life problems, for instance in the context of:

- *mobile users*, where an agent is sent out from a mobile computer in order to accomplish a well-defined task on behalf of the user while he is disconnected from the communication network. Once the user reconnects, the agent returns and reports the result of the task or the problems it encountered.
- *high-bandwidth interactions*, where an agent is sent to a database server that holds a large amount of unstructured data to search for some specific information for the user.
- *customizable services*, where a service provider can offer a small number of simple operations on its service interface that can be combined by an agent to obtain the desired functionality.
- *resident agents*, which are stationary agents that take residence at some service provider and handle simple routine actions for their owner (e.g., communication management for a mobile user, where the agent decides how to handle an incoming communication request).

2.1 Basic definitions

In this article, we do not focus on the underlying technology that is used to implement the paradigm, but we only require a simple model of agents for our discussion. Therefore, we identify the following major abstractions that we associate with mobile agents. A mobile agent consists of code, data, and its current execution state, which can be marshalled by the *agent owner* in a transport format and subsequently sent to the *agent executor*. The agent can be confidentiality and integrity protected during transit to prevent outside attacks through the use of cryptographic mechanisms. These mechanisms can also provide data origin authentication for the marshalled agent. The agent executor will then eventually unmarshall the agent and instantiate it on a special environment

located at the agent executor, which is called *agent platform* (AP). Here, the mobile agent can interact with services of the local AP as well as other agents located at this AP and continue to accomplish the task it was given by its owner.

The literature on agents (e.g., [3,13]) distinguishes between two different approaches to support agent mobility: *weak* mobility and *strong* mobility. The former does not automatically support the transfer of the current execution state of the agent. Thus, if an agent is supposed to visit more than a single AP (which is often referred to as multi-hop agent), the current execution state has to be explicitly encoded in the agent's data before it can migrate to another AP. The latter approach automatically supports the transfer of the current execution state and allows an agent to continue its execution exactly where it left off before initiating the migration. A mobile agent can thus easily visit as many APs as it deems necessary to accomplish the desired task. In both approaches, the result of the agent's remote execution can be sent directly to the agent owner in the form of a message or kept in the execution state of the agent and extracted by the agent owner when the agent returns.

The reason for only providing weak instead of strong mobility is that the execution environment in the AP and the agent transport format can be much simpler since the AP does not have to provide the current execution state (which is, for instance, not available from the Java virtual machine) and the transport format does not have to encode it. Also, if an agent visits only a single AP, which is supported by weak mobility, the trust model becomes much simpler. Any damage incurred by the agent to the AP (and thus to the agent executor) or by the AP to the agent (and thus to the agent owner) can easily be attributed to the other entity. If more than two principals are involved, the problem of accountability becomes much more difficult [13]. Each principal can defer any damage to actions by one of the other principals on the agent's itinerary.

In this article we will concentrate on the protection of a mobile agent on a particular AP. In our solution, a multi-hop agent that visits several sites only represents multiple instances of the same problem. Our main concern is how to protect the agent and especially the data it contains from undue manipulation by or undesired disclosure to the agent executor, which is mainly a question of trust in the agent executor.

2.2 The need for trust

There are many examples where an agent might need confidential information that should not be disclosed to the service provider, even though the agent needs the information to accomplish its task:

- a shopping agent in an electronic-commerce system might hold data that could give a bargaining advantage to the service provider if it were known to him (e.g., a maximum price that the service user is willing to pay or the lowest QoS that he is willing to accept before inquiring at another service provider).

- another agent for electronic commerce might hold a private key with which it can sign messages on behalf of its owner, for instance, to confirm an order placed by the agent. This key has to be kept secret to prevent that the service provider can also sign messages on behalf of the agent owner.
- an agent in a personalized information system might contain private data from the customization by its owner that is needed to find appropriate information. An agent searching for movies that are likely to interest its owner, might contain some very personal information about the user's special interests, which the agent executor cannot infer from simply observing the agent's choice.
- finally, an agent that merely searches for some particular financial information (e.g., stock quotes) might, depending on the owner of the agent, convey some very sensitive information (the mere request already conveys the interest in the information).

2.3 Threats to a mobile agent

In a conventional mobile agent system, when the agent owner sends a mobile agent to an agent executor in order to use some service, the agent owner loses all control over the code and data of the agent. The agent executor can:

- reverse engineer the agent's code,
- analyze the agent's data,
- arbitrarily change the agent's code and data, or, if none of these direct attacks is feasible,
- experiment with the agent (e.g., by feeding it with arbitrary data and resetting it to its initial state, in order to observe the agent's reactions).

This constellation puts the agent executor in a much stronger position than the agent owner. The agent owner simply has to trust the agent executor not to use the methods described above to illicitly obtain confidential information from the agent that it has to carry in order to use the service. There is no way for the agent owner to control or even know about the behaviour of the agent executor.

The reason for the imbalance between agent executor and agent owner in the mobile agent model as compared to the principals in the client/server model is that in the former approach, the agent owner has no guarantees whatsoever concerning the execution of its agent. In the client/server approach, the client relies on many guarantees that are so basic that one hardly ever thinks of them. Nevertheless, these guarantees allow the implementation of certain types of behaviour in the client part of the distributed application that can not be implemented in conventional agent systems (e.g., code will be executed at most once, code will be executed correctly, or the code can rely on a reasonably accurate time service). This is due to the fact that the client implementation is under the physical control of the service user, who can can observe what is happening in the system and notice any irregularities. Thus, he is able to react accordingly, for instance, to interrupt an ongoing transaction or to log any irregularities at the

client side so that they can be provided as evidence in the case of a dispute with some server. This is opposed to the mobile agent paradigm, where logged data can easily be deleted by the agent executor.

We intend to create an environment for mobile agents that allows them to base their execution on assumptions similar to the client/server approach, so that it becomes possible for a mobile agent to better protect itself from a malicious agent executor.

3 The notion of trust

We already mentioned the importance of trust for security in distributed systems and pointed out the lack of a clear definition of what is meant by the terms trust or trusted principal. In the following, we present our analysis of possible trust relations between different principals.

A reason for the lack of a clear definition of trust could be that trust is more a social than a technical issue and consequently quite difficult to tackle entirely in a technical approach. The major problem stems from the fact that the notion of trust mixes the goals of a principal with its behaviour to achieve these goals. In order to trust some principal, it is usually necessary to concur with or at least to approve of its goals (which are not always clearly stated) and to believe that it will behave accordingly. In our definition of trust, we will try to clearly separate these two issues by identifying a policy that is consistent with the goals of the principal. This policy is a set of rules that constrains the behaviour of the principal for all conceivable situations. It has to be written down and made available to all other principals that interact with the issuer of the policy. Then, we define *trust in another principal* as the belief that it will adhere to its published policy.

The question of whether a certain principal can be trusted now consists of (a) checking its published policy in order to decide if it is acceptable and (b) to establish a motivation for the belief that it will adhere to its published policy. The former is quite difficult but can be supported by a formal specification of the security policy (similar to the approach in [15]). The latter, however, is a problem that is quite difficult to formalize. Depending on how the belief in the adherence to the published policy is motivated, we have identified two fundamentally different approaches to the problem of trust:

- the optimistic approach and
- the pessimistic approach.

In the optimistic approach, we give an entity the benefit of the doubt, assume that it will behave properly, and try to punish any violation of the published policy afterwards. In the pessimistic approach we try to prevent any violation of the published policy in advance by effectively constraining the possible actions of a principal to those conforming to the policy. Both of these approaches have advantages and disadvantages.

3.1 The optimistic approach

This approach is easy to implement, since it does not require any special measures to make trusted interaction possible. This is probably the reason why it is the basis for most business conducted today. On the other hand, it requires some reliable mechanism to discover a policy violation after it has occurred. If such a mechanism does not exist, then the approach degenerates to *blind trust*, which indicates that there is no particular motivation to believe that a principal will adhere to its published policy other than its own assertion. Blind trust is obviously a very weak foundation for trust and not recommended for any important or financially valuable transaction. It is therefore important to make the probability that a policy violation is discovered as high as possible by improving controls and establishing checkpoints.

Once a policy violation is discovered and if it can further irrefutably be attributed to one of the participants in the corresponding transaction, this principal should be punished according to the appropriate laws and the damage caused by the policy violation. The primary goal of this punishment is to deter potential violators from committing a policy violation in the first place.

Depending on how this punishment is enacted, we identify the following two motivations for the belief that an entity will adhere to its published policy:

- trust based on (a good) reputation and
- trust based on explicit punishment.

Trust based on reputation stems from the fact that the principal in question is well known and has very little to gain through a violation of its own policy but a lot to lose in case a policy violation is discovered. This loss is supposed to transpire from the lost revenue due to customers taking their business to another provider. Reputation is an asset that is expensive to build up and that is invaluable for any company. Thus, we assume that a principal would not risk to lose its good reputation for a small gain and will consequently rather adhere to its policy.

Trust based on explicit punishment means that we do not trust the principal, but rather the underlying legal framework to ensure the principal's proper behaviour. Here, we explicitly introduce a similar tradeoff as in trust based on reputation by imposing disciplinary actions such as fines or imprisonment, depending on the severity of the offence. The short term gain that might be achieved through a policy violation is supposed to be negated by appropriate punishment.

Obviously, there are many problems with this approach, such as the enforcement of laws, which is usually expensive, quite slow, and sometimes very complex (in particular if the laws of different countries are applicable as can be expected for transactions on the Internet). The difficulty of very different perceptions of punishment, where a person who has not much to lose might readily risk some years of imprisonment for the possibility of a relatively large gain. Another problem in the optimistic approach stems from the fact that many abuses of confidential information are not necessarily conducted for the purposes of the

company that holds this information, but rather by malicious insiders of such a company, who do it for strictly personal reasons or financial benefits [20, 25]. Such abuses are even more difficult to discover (there are less people involved) and to punish (it has to be decided if only the employee for malicious behaviour, only the company for negligence, or both have to be pursued).

The problem to reliably discover a policy violation could be resolved by requiring a high degree of transparency. However, this is difficult to achieve and it is quite likely that even trustworthy principals with a good reputation might not be eager to accept it in order to protect internal business processes. We therefore assume that complete transparency is not a very useful tool for supervision. A better approach would be to designate specialized appraisal companies that execute frequent in-depth controls of the conduct of companies.

Finally, by definition, the optimistic approach cannot prevent malicious behaviour, but it only tries to compensate for it after it has been discovered. For many situations in real-life, where a violation might have an irreparable effect or where a proper functioning of the system is absolutely essential, this guarantee might not be strong enough. A more detailed discussion of the notion of trust described above, can be found in the paper of Swarup and Fábrega [19].

We would like to remark that most of the described problems are also present in our every-day life and therefore quite well understood. However, the question stands if we can do better than that.

3.2 The pessimistic approach

The pessimistic approach removes all these disadvantages by simply preventing any violation of the published policy. This would clearly be the best foundation for trust since we can solely rely on a principal's policy to verify that its behaviour will be acceptable. The behaviour of the principal becomes completely transparent as far as it is constrained by its policy without the need to actually supervise any particular action. If the policy prescribes an action for some event and if the policy is enforced then it is guaranteed that the action will take place. Unfortunately, this policy enforcement can not be realized in its full generality, but is limited to those policies (or rules of a policy) that can effectively be enforced with some uncircumventable mechanism. For non-enforceable policies, we still have to rely on optimistic approaches to trust.

There is no simple way to conceive such an uncircumventable enforcement mechanism. Until recently it was considered impossible without relying on some piece of trusted and tamper-resistant hardware [4]. However, in [16] Sander and Tschudin describe a new approach that might eventually be capable to provide some protection for an agent without relying on such hardware. Unfortunately, in its current form the approach does not allow to create agents that encode arbitrary programs, but it only supports polynomial and rational functions. Thus, the only viable way that can be implemented with current knowledge has to rely on trusted and tamper-resistant hardware. In the following section, we will describe such a piece of hardware and the requirements that have to be met so that

it can be used to enforce certain rules of a policy. This hardware is comparable to the *Secure Coprocessor* described by Yee in [27].

4 Tamper-resistant hardware and the CryPO protocol

We will first present the execution environment that we rely on and then describe the protocol that uses it. Figure 1 gives an overview of the principals in the system.

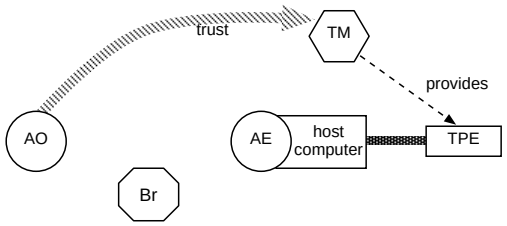

Fig. 1. Overview of the Principals in the CryPO protocol

A manufacturer (TM) produces the execution environment(TPE), which can be bought by any agent executor (AE). An agent owner (AO) has to trust the manufacturer to design and produce its execution environments properly (see Section 6). The broker (Br) is basically a directory service to locate other principals and to obtain their credentials.

4.1 Notation

The described approach relies on public key cryptography [5] (such as RSA [14]). A detailed description of cryptography and the corresponding notations is not within the scope of this presentation. For information on this topic see e.g., [12, 17]. The notation we will use is as follows.

A principal P has a pair (or several pairs[1]) of keys (K_P, K_P^{-1}) where K_P is P's public key and K_P^{-1} its private key. Given these keys and the corresponding algorithm, it is possible to encrypt a message m using the receiver P's public key K_P, denoted $\{m\}_{K_P}$, so that only P can decrypt it with its private key. A signed message, including a digital signature on the message m, generated by P using its private key K_P^{-1} and verifiable by anybody using the respective public key K_P, is denoted $\{m\}_{S_P}$.

In the following we assume the usage of optimization schemes such as encrypting a large message with a symmetric session key, which in turn is encrypted

[1] It is advisable to have at least two pairs of keys, one for encryption/decryption and one for digital signatures.

using public key cryptography and prepended to the message as well as the use of hash algorithms to reduce the computational complexity of signing. However, for ease of presentation, we will not make this explicit.

4.2 The processing environment

As we have noted above, there is no way to enforce any particular behaviour from another principal without a piece of trusted and tamper-resistant hardware. The concept of tamper-resistance usually applies to a well-defined module, sometimes called black-box, that executes a given task. The outside environment cannot interfere with the task of this module other than through a restricted interface that is under the complete control of the tamper-resistant module. We will call this device *trusted processing environment* (TPE). The TPE (see Figure 2) provides a complete agent platform that cannot be inspected or tampered with. Any agent residing on the TPE is thus protected by the TPE both from disclosure and manipulation.

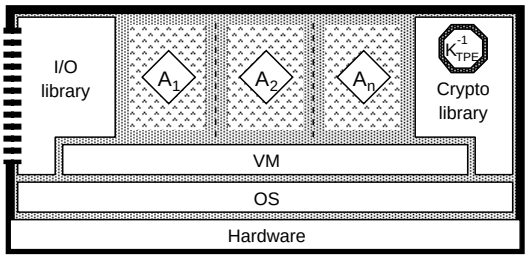

Fig. 2. The trusted processing environment (TPE)

The TPE is a complete computer that consists of a CPU, RAM, ROM, and non-volatile storage (e.g. hard-disk or flash RAM). It runs a virtual machine (VM) that provides the platform for the execution of agents and guarantees the correct execution of the agent's code according to the definition of the used language (e.g., Java byte-code). Below the VM is the operating system that provides the external interface to the TPE and controls the VM (e.g., protection of agents from each other). Furthermore, the TPE contains a private key K_{TPE}^{-1} that is known to no principal other than the TPE – also the physical owner of the TPE does not know the private key. This can, for instance, be achieved by generating the private key on the TPE[2]. Using this approach, the private key is never available outside of the TPE and, thus, protected by the operating system and the tamper-resistance of the TPE. The secrecy of the private key is a crucial requirement for the usage of the TPE to enforce a particular behaviour.

[2] Other, more sophisticated approaches to create the pair of keys could be envisaged, which could also incorporate key recovery mechanisms (e.g., escrowed key shares).

The TPE is connected to a host computer that is under the control of the TPE's owner. This host computer can access the TPE exclusively through a well defined interface that allows, for instance, the following operations on the TPE:

- upload, migrate, or remove agents;
- facilitate interactions between host and agent or between agents on the TPE;
- verify certain properties of the TPE (such as which agents are currently executing).

Due to its implementation as a tamper-resistant module and the restricted access via the I/O interface, it is impossible to directly access the information that is contained in the TPE. This property is ensured by the TPE manufacturer (TM), which also provides the agent executor (AE) with a certificate (signed by TM). The certificate contains information about the TPE, such as its manufacturer, its type, the guarantees provided, and its public key. The agent owner (AO) has to trust the TM (see Section 6) that the TPE actually does provide the protection that is claimed in the certificate.

4.3 CryPO protocol

The CryPO (cryptographically protected objects[3]) protocol transfers agents exclusively in encrypted form over the network to a TPE, using the TPE's public key. Therefore, it is impossible for anyone who does not know the private key to obtain the code or data of such a protected agent.

The protocol is divided into two distinct phases. The first phase consists of an initialization, which has to be executed once before the execution of the second phase of the protocol. This second phase is concerned with the usage of the TPE and the actual transfer of the agent. The protocol is based on the interactions given in Figures 3 and 4.

Initialization In the initialization phase, the participants exchange the required key information:

- we assume that the AO holds an authentic copy of the TM's certification key K_{TM}.
- the TM sends the certificate $Cert_{TPE} = \{K_{TPE}, ...\}_{S_{TM}}$ to the AE.
- the AE registers its reference Ref_{AE}[4] with one or several brokers.

[3] We had originally chosen the term object since it is more general than the term agent.

[4] A reference to an AE consists of its name, its physical address in the network, its policy, and the certificate $Cert_{TPE}$ for its TPE. The broker can also verify that the AE actually controls the corresponding TPE by executing a challenge-response protocol with the TPE via the AE.

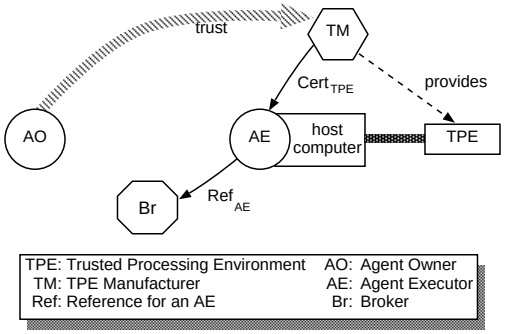

Fig. 3. Initialization of the CryPO protocol

Usage After the participants have finished the initialization, they can execute the usage part of the CryPO protocol:

- the AO queries the broker for the reference to the AE with which it wants to interact (or it already holds this reference from a previous interaction).
- the AO verifies the policy of the AE whether it is acceptable as well as the certificate $Cert_{TPE}$ to check the manufacturer and the type of the TPE, in order to decide if it satisfies the security requirements of the AO. If any of these checks fail, the AO will abort the protocol.
- the AO sends the agent encrypted with the public key of the TPE, $\{A\}_{K_{TPE}}$, to the AE.
- the AE cannot decrypt $\{A\}_{K_{TPE}}$ nor can it do anything other than upload the agent to its TPE.
- the TPE decrypts $\{A\}_{K_{TPE}}$ using its private key K_{TPE}^{-1} and obtains the executable agent A, which it will eventually start. The agent can then interact with the local environment of the AE or with other agents on the TPE.
- the agent can, after it has finished its task, migrate back to its owner ($\{A\}_{K_{AO}}$) or to another AE to which it holds a reference.

The obvious problem of protecting the TPE from malicious agents is independent of the described approach and has to be tackled with additional mechanisms, such as code signing and sandboxing. The problem of protecting the TPE from tampered agents can easily be solved by concatenating the agent with a hash of the entire agent $h(A)$, including its execution state, before encrypting it $\{A, h(A)\}_{K_{TPE}}$. The TPE simply has to verify the correct hash before starting the agent.

4.4 Notes on feasibility

The actual construction of a tamper-resistant module in the real world is difficult; nevertheless, there are many applications that rely on them (e.g., payphones,

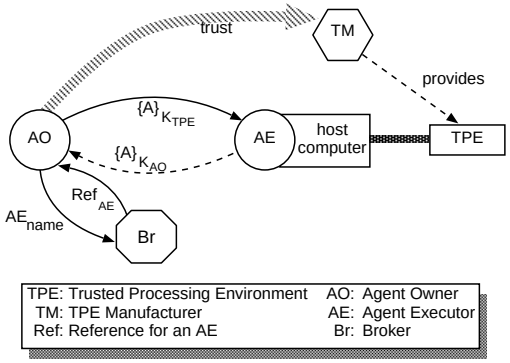

Fig. 4. Usage of the CryPO protocol

debit cards, or SIM cards for GSM). Given sufficient time and resources, it becomes very probable that an attacker can violate the protection of such a module (see e.g., [1]). We believe that the actual realization of the presented TPE with reasonably strong guarantees in real-world settings is also quite difficult, but nonetheless feasible. Especially, since we do not require the prevention but only the detection of tampering with the TPE for most envisioned applications.

We imagine the TPE as a regular computer with a special operating system. It is physically protected with a special hardware that can effectively be sealed to detect tampering, is under continuous video surveillance similar to the systems used to supervise automatic teller machines, and is subject to challenge inspections by the TM or an independent appraisal and inspection organization. As explained in [1], such an installation is conceivable and can even resist massive attacks. A thorough analysis of the remaining risks has to be undertaken, but this is not within the scope of this presentation.

5 Usage of the TPE

The CryPO protocol together with the concept of a TPE guarantee the integrity of the agent platform to the AO and protect the code and data of an agent against manipulation and disclosure, both in transit and during execution. These guarantees are based on the trust relation between the AO and the TM, in which the AO trusts the TM to properly manufacture its TPEs and to control them regularly (if necessary) so that the claimed guarantees hold. The certificate enables the AO to ensure that it really deals with a TPE from a certain manufacturer.

The above guarantees can be extended by additional properties, formulated as rules of a policy, that can effectively be enforced by a TPE. In [24], we have discussed how this approach can be used to allow an agent to base its execution on results of possible previous executions on the same TPE. This can, for instance, be used to limit the number of times an agent can be executed on a given TPE. To

achieve this, it is necessary to identify a policy that provides sufficient support for the agent and to ensure that this policy is enforced on the TPE on which the agent executes. With this approach, the AO does not need to trust the AE on the proper protection of his agent, but it suffices to trust the TM. The question why the AO should trust the TM rather than the AE is discussed in section 6.

Now we want to address a problem that requires the cooperation with a *trusted third party* (TTP). Many security related protocols, in particular those that deal with non-repudiation, rely on such a cooperation [12]. The role of the TTP is to provide a well defined functionality (e.g., timestamping or logging of messages) to create non-repudiable evidence that can later be used to resolve a dispute. The special character of this functionality requires that it is provided on a trustworthy environment, which can be guaranteed in the administrative domain of the TTP. Thus, a user of this functionality has to interact with the TTP server via a remote interaction, which suffers from the usual problems of limited bandwidth, high latency, and inflexibility of the interface. In order to get rid of this remote message exchange we propose to encapsulate the functionality of the TTP in a *TTP agent* that can be executed on a TPE, which is also a trustworthy environment. This allows us to gather all interacting parties on a single platform and to take full advantage of the mobile agent paradigm (see Section 5.4).

We will first introduce a policy for a TPE and discuss its effect on the agent of a regular user at the example of a shopping agent. Then we will show how the same concepts can be applied to a TTP agent and discuss the implications of the approach.

5.1 The policy of the TPE

We assume that the TPE of the service provider enforces the following set of rules, detailed in its policy:

a) the code of an agent will never be disclosed or altered by the TPE.
b) any invocation of the agent's methods will be executed exactly according to the code in the agent.
c) the data of an agent can exclusively be accessed and manipulated through the interface of the agent. If the agent does not provide methods to directly access a particular data item, its value can at most be inferred from the responses to other method invocations.
d) an agent is protected against interference from other agents executing on the same TPE (other than calls on its public interface).
e) prior to a migration, an agent will obtain the certificate of the designated receiver's TPE that also contains the policy. The agent can decide whether it wants to be transferred and the current TPE will honour the agent's decision. The actual transfer follows the CryPO protocol.
f) the TPE provides an internal clock with reasonable accuracy (on the order of several seconds). It will try to synchronize this clock with a trusted time service and will inform the agent if this synchronization did not succeed.

The first rules **a)** and **b)** guarantee the basic protection of the agent's code as well as its proper execution, while **c)** guarantees the protection of the agent's data from undesired disclosure and manipulation. Rule **d)** requires the protection of agents from one another, which is a regular operating system functionality. The next rule **e)** ensures that the agent knows the policy of the TPE to which it is transferred. Thus, the agent can ensure that it will not be sent to a TPE that provides insufficient protection. Finally, rule **f)** can be used for several purposes. For instance, it allows an agent that contains an expiration date to implement a limited lifetime (on the order of a few days or hours). Upon its arrival the agent requests the current time and checks if this time is still within its attributed lifetime. If its expiration date has passed or if the TPE did not succeed to synchronize its clock, the agent can simply abort. An AE can not prevent this if the code of the agent is protected and if it will be executed correctly.

5.2 The shopping agent

Consider a shopping agent that searches for a particular service for its owner. Once it has found a suitable offer, it will negotiate the details of the service provision, such as exact price and various QoS parameters, with the service provider. As a special requirement, we specify that the shopping agent has to create a log entry with a TTP server[5] that contains the details of the negotiated contract before providing the payment data. This allows the AO to reconstruct the activities of his agent in the case of a dispute or if the agent is lost.

In order for the shopping agent to effectively conduct a negotiation it needs to conceal some of its configuration information from the service provider, such as the highest acceptable price or the lowest acceptable QoS parameters. Furthermore, the shopping agent holds the public key of the TTP, which it needs to verify the acknowledgement from the TTP server, as well as the payment data, which should only be provided to the selected service provider after the successful creation of a log entry.

If the shopping agent executes only on TPEs that enforce the policy discussed in Section 5.1, it is clear that it is protected from any interference. Provided that the agent is correct, no other entity will be able to access or manipulate any data contained in the agent other than what is accessible via the methods of its public interface. Thus, the agent can effectively negotiate with a service provider, request the logging of the contract with a TTP server, and delay any further actions until it has received a signed acknowledgement from the TTP server.

[5] The agent could send the corresponding information directly to the AO, but since it needs an acknowledgement for the receipt of the log message and since the AO might not have a permanent connection to the network, it is preferable to delegate this task to a TTP.

5.3 The TTP agent

The interaction described above allows for an efficient negotiation between the agent and the service provider exploiting all the performance advantages of the mobile agent paradigm. However, due to the special requirements of the AO, the agent has to interact with a TTP server via a remote interaction. Apart from the performance penalties of this remote interaction, the TTP server can also become a bottleneck if its resources are consumed by a large number of clients. Therefore, we propose to encapsulate the functionality of the TTP in a TTP agent (*TA*) that can be executed on the TPE, relying on the same protection mechanisms as the shopping agent[6].

In the case of message logging, the functionality of the TTP consists of accepting arbitrary messages, storing them up to a well defined point in time t, and responding with an acknowledgement asserting that the message has been logged. This acknowledgement has to be signed by the TTP and must clearly identify the message that was supposed to be logged, either by including the message itself or, preferably, a hash of the message. Furthermore, the TTP must be capable to reproduce a logged message up to the time t and to provide it (exclusively to authorized principals) upon request. If necessary, a log message can be confidentiality protected with regular encryption methods.

The task of the *TA* is to act as the proxy for the actual TTP on the TPE. It will accept messages that have to be logged from agents on the TPE, store them in a local cache, and respond with an acknowledgement in which it guarantees that it will forward the message to the TTP server unless the TPE is destroyed (see below). Once a log message arrives at the TTP server, it will be handled like a normal message. In order for the *TA* to provide such a guarantee, it needs access to a sufficient amount of non-volatile storage on the TPE, in which it can safely store the log messages. Since this non-volatile storage is a limited resource of the TPE, the *TA* needs a special authorization to use it. If the TPE owner does not grant this authorization, the *TA* will abort.

Apart from the access to the non-volatile storage of the TPE, the requirements of the *TA* are very similar to those of the shopping agent. It has to be protected against manipulation of its code and data and it also has to conceal certain data items from the agent executor such as two different cryptographic keys. The first key is necessary as a means to securely forward the logged messages to the TTP server. Since the *TA* is configured by the TTP, this can simply be a secret key of a symmetric key cryptosystem. The second key is needed as a signature key to sign the acknowledgements for logged messages. This does not necessarily have to be the long-term signature key of the TTP, but can be a temporary key that is validated by a certificate signed with the TTP's long-term signature key.

Again, if the TTP ensures that the *TA* only executes on TPEs that enforce the policy discussed in Section 5.1 with sufficiently high assurance, it is clear that the keys are protected and that the proper execution of the *TA* is guaranteed.

[6] A TTP might require a higher level of assurance in the protection of the TPE than a regular user. This could be a differentiating feature of TPEs from different manufacturers.

The remaining problem is how the *TA* can guarantee that logged messages that are stored in the non-volatile storage of the TPE will eventually be forwarded to the TTP server. The TPE owner could simply intercept all the messages of the *TA* to the TTP server or, ultimately, request the TPE to terminate the *TA*. This functionality has to be offered by the TPE to protect its owner from malicious or simply buggy agents that refuse to terminate.

This problem can be solved with a supervision of the *TA* by the TTP and with the help of the internal clock provided by the TPE. The TTP has to keep track of all the *TA*s it sent to the various service providers and of an expiration date that is associated with each *TA*. A *TA* will accept log messages only until its expiration date. After this date it will refuse to accept and acknowledge any further messages. Thus, the TTP has to receive a final message from the *TA* after its expiration date (there should be an additional delay to accommodate for clock skew) indicating that no further messages for the TTP are stored in the non-volatile storage of the TPE. Provided that the *TA* only deletes messages from this non-volatile storage after sending them to the TTP server and obtaining an acknowledgement, the TTP knows that all the messages that the *TA* acknowledged have been forwarded to the TTP server. If this final message is not received, the TTP will request the TPE owner to restart the *TA* and to forward any messages it sends to the TTP server. Under the assumption that the TTP has a possibility to enforce the access to the TPE by legal means, the only possibility of the TPE owner to avoid the provision of missing messages is to destroy the TPE. Thus, the approach cannot guarantee that all logged messages will be delivered to the TTP server. But it can guarantee that a TPE owner can not cheat without being discovered. Furthermore, if it can be proven that the TPE owner intentionally destroyed the TPE, he can be punished with adequate fines.

The problem of destruction of storage media is not new and also applies to a regular TTP. However, it is assumed that the TTP operator implements adequate measures to avoid this problem.

5.4 Discussion

The relocation of the TTP functionality from a remote site to the locally managed TPE allows us to prevent it from becoming a bottleneck that slows down other components. This is possible since the TPE owner can allocate as many resources as necessary to the *TA* without having to coordinate this with the TTP. Also, since the *TA* can collect and merge several log messages from interactions of different agents with the service provider, it can accumulate several log messages and forward them in a single remote interaction. Another major advantage of the described approach is that the remote interaction with the TTP server is taken off the critical communication path between the agent and the service provider. They can continue their interaction as soon as the *TA* has stored the log message and sent the acknowledgement. Moreover, the entire interaction can exploit the locally available communication links with higher bandwidth and lower latency. This enables not only a better overall performance of the system,

but allows interactions that were not possible before due to an unreasonable overhead. For instance, a *TA* could be used by two interacting parties as an intermediate through which all messages are exchanged. The *TA* could, thus, easily log the entire interaction and forward it to the TTP.

The concept of the *TA* is suitable for any TTP functionality that can be wrapped up in a reasonably small object and that does not have to rely on a large centrally managed database. Other interesting examples are timestamping or fair-exchange. The former consists of a *TA* that adds a timestamp to a message and signs the resulting message with its signature key. The latter is a classical security problem, for which several solutions relying on a TTP have been proposed [2]. The problem of fair-exchange is that of principals A and B who want to exchange the data items D_A and D_B, but neither of them wants to provide its data item before receiving that of the other principal. A *TA* can facilitate the exchange by accepting the data items as well as a description of the data items expected by the designated receivers. It will verify if the descriptions match the actual data items (e.g., in the case of payment, it verifies if the paid amount corresponds with the amount expected by the receiver) and, if this is the case, deliver the data items to the designated receiver.

6 Trust in the TPE manufacturer

We have introduced the mechanism, with which an agent can take advantage of the policy enforced by a TPE. However, as we have mentioned above, in order for a principal to trust in the proper enforcement of this policy, it is necessary that he also trusts the TPE manufacturer to properly design, implement, and produce its TPEs. Since there is no way (to the knowledge of the authors) to enforce a correct behaviour of the TPE manufacturer, it seems that the presented approach simply replaces one required trust relationship with another one. This is a correct observation from a theoretical point of view. Nevertheless, we believe that the replacement of trust in an arbitrary service provider with trust in a TPE manufacturer has several more subtle implications. We will briefly discuss the following advantages that we identified:

- better understanding of security and privacy problems
- centralized control
- resources to build reputation
- separation of concern

The TPE manufacturer is a specialized service provider, which primarily deals in the field of the provision of security devices. Therefore it has a better understanding of security and privacy problems, which makes it a much more capable entity to ensure this service since it is more aware of the potential problems and pitfalls.

We assume that there will be relatively few TPE manufacturers (on the order of several hundreds) compared to the number of possible operators of the TPE (on the order of several millions). This makes the control of their behaviour much

easier for expert appraisal organizations. Also, it is quite conceivable that a TPE manufacturer might invite external experts to control its internal operation, in order to obtain a better position in the market (similar to the approach for quality assurance in the ISO-9000).

The production of TPEs is considered to be a difficult task (see Section 4.4). Therefore, we assume that it will be undertaken by major corporations, which have the necessary resources to build a good reputation and which have an incentive to protect this reputation. This allows us to rely on good reputation as foundation for trust in the TPE manufacturer.

The TPE manufacturer that is responsible for the enforcement of the proper policy rules on the TPE, has nothing to gain by not accomplishing its task. Since the TPE will be operated independent from the TPE manufacturer by a completely different principal and since the TPE manufacturer has no means to access the data that is processed on the TPE (no physical connection), there is no possibility for the TPE manufacturer to draw a direct benefit from a TPE that does not properly enforce its policy[7].

We assume that the above arguments of high expertise, effective controllability, good reputation, and lack of incentive are sound reasons to trust a TPE manufacturer to build reliable and powerful TPEs. The main advantage of the approach lies in the possibility to leverage this trust in the TPE manufacturer onto a completely different principal in the role of the service provider, which

- might not have the proper expertise to ensure a secure operation of its hardware and to guarantee the protection of the processed data.
- is quite difficult to control, due to the sheer number of such service providers.
- might have no particular reputation (and therefore none to lose).
- might have short term goals that (in its point of view) justify a policy violation.

With the presented approach, such a service provider can easily define the policy rules that it would like its TPE to enforce (by selecting from the options offered by the TPE manufacturer) and buy the appropriate TPE from a reputable TPE manufacturer. The service provider can then immediately benefit from the trust that users have in the manufacturer of its TPE to convince them that it will not maliciously abuse an agent sent by a user.

With this, the approach favours the open systems philosophy, where any principal can possibly become a provider of services. Such a service provider simply has to obtain a TPE from some reputable manufacturer and can then easily convince a client that the client's confidential information is sufficiently well protected. Thus, it becomes much easier for a new service provider to establish itself in the market.

[7] There is the possibility that a TPE operator bribes a TPE manufacturer to provide an incorrect TPE. We assume that such a behaviour is a severe offence that is subject to criminal investigation and not within the scope of this discussion.

7 Conclusion

In this paper, we have discussed the notion of trust in the context of mobile agent systems and introduced a structuring for this problem domain. Starting from this structure, we have proposed an approach that relies on a trusted and tamper-resistant hardware device, which allows the prevention of malicious behaviour rather than its correction. We believe this to be the better form of protection for confidential data. We have shown how the approach can be used to effectively protect the confidential data contained in the shopping agent of a user and how it can be extended to protect specialized agents from TTPs that provide facilitation services.

In real-life, there are limitations to the approach. Given sufficient time and resources, a TPE operator might succeed in breaking the system and it would thus be possible for him to violate the policy that should be enforced by the TPE. Our goal is to make this attack so costly that it would negate a possible gain (there may be many different implementations that provide different levels of assurance in the protection of a TPE). As further deterrent, we assume that a non-repudiable proof for a policy violation of an enforced policy or for an attempted or successful breaking of a TPE might be punished much more severely than a mere policy violation since it proves a much larger determination to commit a criminal offence.

Acknowledgements

This research was supported by a grant from the EPFL ("Privacy" project) and by the Swiss National Science Foundation as part of the Swiss Priority Programme Information and Communications Structures (SPP-ICS) under project number 5003-045364.

References

1. R. Anderson and M. Kuhn. Tamper resistance — a cautionary note. In *The Second USENIX Workshop on Electronic Commerce Proceedings*, pages 1–11, Oakland, California, November 1996.
2. H. Bürk and A. Pfitzmann. Value exchange systems enabling security and unobservability. *Computers & Security*, 9(8):715–721, 1990.
3. A. Carzaniga, G. P. Picco, and G. Vigna. Designing distributed applications with mobile code paradigms. In R.Taylor, editor, *Proceedings of the 19th International Conference on Software Engineering (ICSE'97)*, pages 22–32. ACM Press, 1997.
4. D. M. Chess, B. Grosof, C. G. Harrison, D. Levine, C. Parris, and G. Tsudik. Itinerant agents for mobile computing. *IEEE Personal Communications*, 2(3):34–49, October 1995.
5. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6), November 1976.
6. DoD. Trusted Computer System Evaluation Criteria (TCSEC). Technical Report DoD 5200.28-STD, Department of Defense, December 1985.

7. J. Gosling and H. McGilton. The java language environment. White paper, Sun Microsystems, Inc., 1996.
8. R.S. Gray. Agent Tcl: A transportable agent system. In *Proceedings of the CIKM Workshop on Intelligent Information Agents*, Baltimore, MD, December 1995.
9. C. G. Harrison, D. M. Chess, and A. Kershenbaum. Mobile agents: Are they a good idea? In *Mobile Object Systems: Towards the Programmable Internet*, volume 1222 of *Lecture Notes in Computer Science*, pages 25–47. Springer Verlag, 1997.
10. ITU. *ITU-T Recommendation X.509: The Directory – Authentication Framework*. International Telecommunication Union, 1993.
11. D. B. Lange and M. Ishima. *Program and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1998.
12. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC Press, Inc., 1997.
13. J. Ordille. When agents roam, who can you trust? Technical Report Technical Report, Computing Science Research Center, Bell Labs, 1996.
14. RSA Data Security, Inc. *PKCS #1: RSA Encryption Standard*. RSA Data Security, Inc., November 1993.
15. R. A. Rueppel. A formal approach to security architectures. In *EuroCrypt*, pages 387–398, Brighton, England, 1991.
16. T. Sander and C. Tschudin. Towards mobile cryptography. In *IEEE Symposium on Security and Privacy*, May 1998.
17. B. Schneier. *Applied cryptography*. Wiley, New York, 1994.
18. J. G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the USENIX Winter 1988 Technical Conference*, pages 191–202. USENIX Association, Berkeley, USA, February 1988.
19. V. Swarup and J. T. Fabrega. Understanding trust. In *Secure Internet Programming* [22], pages ??–??
20. New York Times. U.S. workers stole data on 11,000, agency says, April 6, 1996.
21. G. Vigna. Protecting mobile agents through tracing. In *Proceedings of the Third Workshop on Mobile Object Systems*, Finland, June 1997.
22. Jan Vitek and Christian Jensen. *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*. Lecture Notes in Computer Science. Springer-Verlag Inc., New York, NY, USA, 1999.
23. J. E. White. Telescript technology: The foundation for the electronic market place. White paper, General Magic, Inc., 1994.
24. U. G. Wilhelm, L. Buttyàn, and S. Staamann. On the problem of trust in mobile agent systems. In *Symposium on Network and Distributed System Security*, pages 114–124. Internet Society, March 1998.
25. I. S. Winkler. The non-technical threat to computing systems. *Computing Systems, USENIX Association*, 9(1):3–14, Winter 1996.
26. T. Y. C. Woo and S. S. Lam. Authentication for distributed systems. *IEEE Computer*, 25(1):39–52, January 1992.
27. B. Yee. A sancturary for mobile agents. In *Secure Internet Programming* [22], pages ??–??
28. P. Zimmermann. *PGP User's Guide*. MIT Press, Cambridge, 1994.