A Pessimistic Approach to Trust in Mobile Agent Platforms

UWE G. WILHELM, SEBASTIAN M. STAAMANN, AND LEVENTE BUTTYÁN
Ecole Polytechnique Fédérale de Lausanne

The problem of protecting an execution environment from possibly malicious mobile agents has been studied extensively, but the reverse problem—protecting the agent from malicious execution environments—has not. The authors propose an approach that relies on trusted and tamper-resistant hardware to prevent breaches of trust, rather than correcting them after the fact.

Mobile agent technologies such as Aglets[1] and Telescript[2] are being deployed on the Internet to support new approaches to distributed computing. In the domain of electronic commerce, a scenario involving these technologies might consist of an agent program that searches a service for its owner by roaming the Internet and visiting the sites of various service (or product) providers. Such an agent is configured by its owner with all the relevant information to describe a desired service, the constraints on an acceptable offer, and a list of potential providers. The agent may also hold confidential information such as data for one or several payment methods to finalize a purchase. The agent should make this data available to a provider only in the event of a purchase. Even then, it should offer only data pertinent to the payment method used in the purchase.

Because the agent is vulnerable while it is executing on the service provider's execution platform, its owner must obtain some guarantees concerning the protection of the agent. Example threats from a malicious service provider include trying to obtain payment data without providing the service or trying to remove information about a better offer from the agent's memory, thereby tricking it into accepting the malicious provider's offer.

The usual approach to protecting mobile agents is to assume that service providers are trusted principals that behave correctly.[3] Although the importance of trust has long been recognized as paramount for the development of secure systems, the meaning associated with trust or a trusted principal is seldom clearly defined. In this article, we address the question of how to base trust on technical reasoning. We present a pessimistic approach to trust, which tries to prevent malicious behavior from occurring in the first place rather than correcting it after it has occurred. Our approach relies on a tamper-resistant hardware device that can be operated safely in an untrusted environment.

The ideas presented here are purely conceptual and have yet to be implemented. Nevertheless, we believe that they can have important repercussions on the design of open mobile agent systems, whereby potentially everyone could become a service provider.

 1089-7801/00/$10.00 ©2000 IEEE

We begin by introducing our model for mobile agents and pointing out the problems related to trust within this model. We then discuss the notion of trust and define its relation to policy before describing a piece of trusted hardware and a protocol that, together, establish a technological basis for trust in the context of mobile agents.

MOBILE AGENT PARADIGM

Many researchers have proposed mobile agents as a promising approach to structure problems in distributed computing (for example, see Chess et al.[4]). The merits of the mobile agent paradigm, however, are still debated; and Harrison et al.[5] have shown that it has no conceptual advantage over classic client-server approaches. On the other hand, these same authors point out that the mobile agent paradigm can offer interesting solutions to many real-life problems in at least two contexts:

- *high-bandwidth interactions*, where the user sends an agent to search for some specific information on a database server that holds a large amount of unstructured data, and
- *mobile users*, where a user who is disconnected from the communication network sends an agent out from a mobile computer in order to accomplish a well-defined task.

In this article, we are not concerned with the underlying technology that implements the mobile agent paradigm. We require only a simple model for our discussion:

- a mobile agent consisting of code, data, and the current execution state. The agent can be marshaled by its owner in a transport format and subsequently sent to an agent executor.
- cryptographic mechanisms that can protect the agent's confidentiality and integrity during transit. These mechanisms also provide origin authentication for the marshaled agent.

The executor will eventually unmarshal the agent and instantiate it on a special environment called the *agent platform*. Here, the mobile agent can interact with local services, as well as other agents located at this platform, and continue the task it was given by its owner. After the agent has completed its local interaction, it can request migration to another platform or back to its owner.

There are many cases where an agent may need to hold confidential information that should not be disclosed to either the executor or the service provider (two principals that could be identical or could easily cooperate to mount an attack on the agent):

- A shopping agent that integrates mechanisms for online payment may hold data for several payment methods, such as different credit cards. In the event of a purchase, the service provider should be able to obtain data for the one payment method used, but not for the others.
- An agent for electronic commerce may hold a private key with which it can sign messages on behalf of its owner. This key must be kept secret to prevent the service provider from illegitimately signing messages in the agent owner's name.
- Finally, an agent that merely searches for some particular financial data, such as stock quotes, may convey some very sensitive information; the request itself already conveys interest in the data.

In a conventional mobile agent system, when an executor receives a mobile agent, the owner loses all control over the agent's code and data. The executor can reverse-engineer the code, analyze the data, or arbitrarily change either one. If no direct attack is feasible, the executor can still experiment with the agent by feeding it arbitrary data to observe its reactions and by resetting it to its initial state. The executor could even do this with a copy of the agent on an isolated platform. The owner has to trust the executor not to use these methods to illicitly obtain confidential information.

This problem does not exist in the client-server paradigm, where the client can rely on many low-level system guarantees (for example, that code will be executed at most once or that it will be executed correctly). The client implementation resides in a physically trusted user environment where it can, for instance, log any irregularities as evidence for a possible dispute with a service provider. This contrasts with the mobile agent paradigm, where the owner cannot control or even reliably know about the executor's behavior. Digitally signing the agent's data can considerably limit the malicious actions of an executor, but it cannot prevent them.

We want to suggest an environment for mobile agents that lets them base their execution on guarantees, similar to those provided by the client-server paradigm, so that a mobile agent can protect itself from a malicious agent executor.

THE NOTION OF TRUST

Central to our definition of trust is a *policy*—that is, a set of rules prescribing a principal's behavior for all relevant situations. This policy must be written down and made available to all other principals that interact with its issuer. The policy is, of course, consistent with the issuer's goals. This policy lets us separate the notion of trust into two components: *adequacy* of the policy and *trustworthiness* of its issuer:

- For a principal A to trust a principal B, A must verify B's policy and decide whether it adequately protects A's interests. This problem is difficult but can be assisted by a formal specification of the policy (similar to the approach described for security architectures in Rueppel[6]).
- Then A must assess B's trustworthiness by establishing a basis for believing that B will adhere to its published policy. This problem is quite difficult to formalize.

Depending on how the trustworthiness of a principal is established, two fundamentally different approaches to trust can be identified: one optimistic and the other pessimistic. (We have named these approaches in accordance with similar concepts in transaction processing.)

Optimistic Approach

In the optimistic approach, principal A gives principal B the benefit of the doubt, assumes that it will behave properly, and tries to punish any violation of the published policy after the event. This approach is easy to implement because it requires no special measures to allow trusted interactions between A and B. It does, however, require some reliable mechanism to discover a policy violation after it has occurred. (The extreme case of this approach, in which such a mechanism does not exist, is not recommended for any important transaction and is not discussed further.)

If internal business processes were transparent to external observers, it would be easier to discover a policy violation. It is unlikely, however, that many principals would support this transparency. An alternative is to designate specialized companies that would execute frequent and in-depth appraisals of company conduct.

If a policy violation is discovered and if it can be attributed irrefutably to one of the principals in the corresponding transaction, this principal could be punished according to the appropriate laws and the damage caused by the policy violation. The primary goal of this punishment is to deter violations from occurring in the first place. Depending on how this punishment is enacted, the optimistic approach can be further subdivided into trust based on (a good) reputation and trust based on explicit punishment.

- In trust based on reputation, A assumes that B is well known and has little to gain but much to lose from a discovered violation of its own policy. This loss is attributed to lost revenue when customers take their business to another principal.
- In trust based on explicit punishment, A does not necessarily trust B, but rather trusts the underlying legal framework to marshal B's behavior. The tradeoff is similar to that in trust based on reputation, with disciplinary legal actions substituting for lost revenues. An obvious problem with this approach is enforcing such laws, particularly if different countries are involved.

By definition, the optimistic approach cannot prevent malicious behavior but tries only to compensate for violations. In many real-world situations, however, proper functioning of the system is absolutely essential, and any violation can have irreparable effects.

Pessimistic Approach

In the pessimistic approach, principal A trusts that principal B is prevented from performing actions that do not conform to its defined policy. The behavior of B—so far as it is constrained by its policy—becomes completely visible, eliminating the need to scrutinize any particular action. If the policy prescribes a particular action for some event and if the policy is enforced, then the action is guaranteed to take place. Unfortunately, this approach cannot be realized in its full generality but is limited to those policies (or rules of a policy) that can be effectively enforced with a mechanism that cannot be circumvented. For policies that cannot be enforced, principals must rely on optimistic approaches to trust.

PROPOSED IMPLEMENTATION

We know of no simple way to conceive an enforcement mechanism that cannot be circumvented without relying on trusted and tamper-resistant hardware. This was in principle the conclusion reached by Chess et al.[4] Sander and Tschudin[7] have since described an approach that may eventually provide some agent protection based purely on cryptographic mechanisms; however, their approach supports only polynomial and rational functions and does not yet allow

the creation of agents that encode arbitrary programs. Therefore, our proposed approach is implemented on trusted and tamper-resistant hardware.

The proposed implementation relies on public key cryptography. In this approach, principals A and B each have a pair of keys, one public and one private (security can be further increased if each principal has two pairs of keys, one for encryption-decryption and one for digital signatures). Given these keys and the corresponding cryptographic algorithms, A can encrypt one message with B's public key to obtain an encrypted message, which only B can decrypt by using its private key. Principal A can use its private key to generate a signed message (including a digital signature) that can be verified by anyone who has A's public key. (For details on public key cryptography, see Menezes et al.[8])

In the following we assume the use of optimization schemes, such as encrypting a large message with a symmetric session key, which in turn is encrypted through the use of public key cryptography and hash algorithms to reduce the computational complexity of signing.

Figure 1 gives an overview of the trusted principals in the proposed implementation: a trusted *manufacturer* produces the execution environment, called the *trusted processing environment* (TPE), which can be purchased by an *agent executor*. An *agent owner* has to trust the manufacturer to design and produce its execution environments properly. The *broker* is primarily a directory service to locate other principals.

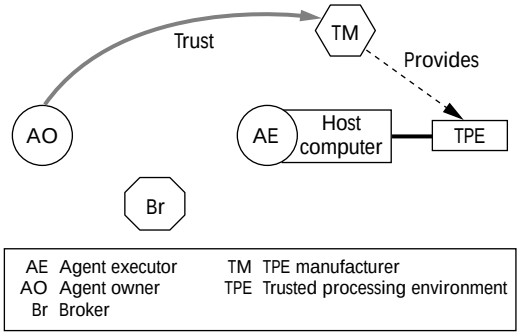

Figure 1. Trusted principals in the proposed implementation. A trusted manufacturer produces the execution environment, which can be purchased by an agent executor. An agent owner has to trust the manufacturer to design and produce its execution environments properly. A broker is primarily a directory service for locating other principals.

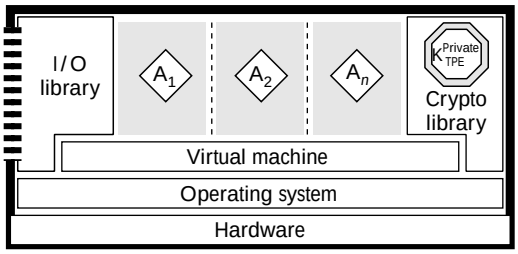

Figure 2. The trusted processing environment. The TPE is a tamper-resistant computer that contains all the code on its own ROM that it requires to boot and to operate a complete agent platform.

Processing Environment

The concept of tamper-resistance usually applies to a well-defined hardware module that executes a given task. The outside environment cannot interfere with the task except through a restricted interface that is completely controlled by the tamper-resistant module. We call this device a trusted processing environment; the TPE provides a complete agent platform, as shown in Figure 2, which cannot be inspected or tampered with. Any agent residing on the TPE is thus protected from both disclosure and manipulation.

The TPE is a complete computer that consists of a CPU, RAM, and ROM. It boots from the ROM, which contains all the code required to operate the TPE. This includes the code of the virtual machine (VM), which provides the platform for agent execution and which guarantees that an agent's code will be executed correctly. The VM is governed by a trusted operating system, which is also loaded from the ROM. The operating system coordinates and controls the TPE resources, namely the CPU, the external I/O interface, and the access to memory. (The use of virtual memory techniques can protect agents from each other by establishing and enforcing protection domains; it does not, however, overcome the problem of covert channels between agents on the same TPE.) The TPE also provides cryptographic functionality in the form of an extensive cryptographic library. This library contains a private encryption key that is not known outside the TPE—even the physical owner of the TPE does not know this private key. This secrecy can be achieved, for instance, by generating the keys on the TPE itself. The secrecy of this private key, ensured by the TPE's tamper-resistance, is a crucial requirement to the proposed implementation.

The TPE is connected to a host computer controlled by the TPE's owner. This host computer can access the TPE only through a well-defined interface that allows, for instance, the following operations on the TPE:

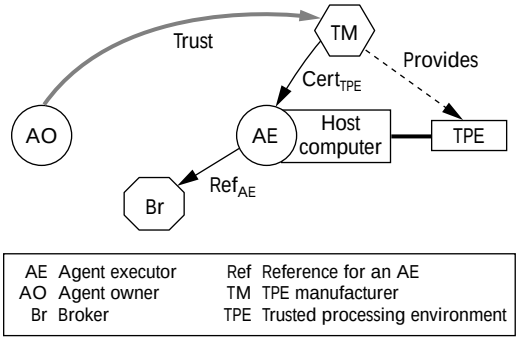

Figure 3. Initialization phase of the CryPO protocol. The participants exchange the required information key information. An agent owner holds a copy of a TPE manufacturer's public key. An agent executor registers the certificate for its TPE with a broker.

- uploading, migrating, or removing agents;
- facilitating interactions between the host and agents;
- verifying certain properties of the TPE, such as which agents are currently executing.

Because the TPE is implemented as a tamper-resistant module with restricted access via the I/O interface, no one outside the TPE can directly access information inside it. This property is ensured by the TPE manufacturer, which also provides a signed *certificate* to the agent executor. The certificate contains information about the TPE, such as its manufacturer, type, guarantees, and public key. The agent owner has to trust the manufacturer that the TPE actually does provide the protection claimed in the certificate (discussed further in the section, "Trust in the Manufacturer").

CryPO Protocol

We have defined a protocol called CryPO (short for Cryptographically Protected Objects) to transfer agents exclusively in encrypted form over the network to a TPE by using the TPE's public key. The CryPO protocol makes it impossible for anyone who does not know the private TPE key to obtain the code or data of an agent in transit to a TPE. CryPO has two distinct phases.

Phase 1: Initialization. The first phase consists of an initialization, shown in Figure 3, which allows the agent owner to verify that it interacts with the TPE of a trusted manufacturer. This phase sets up the required key information exchanges:

- The agent owner holds an authentic copy of the trusted TPE manufacturer's public key, which is used to verify TPE certificates.
- The trusted manufacturer sends the signed TPE certificate to the agent executor.
- The executor registers its *reference*—name, physical network address, policy, and signed TPE certificate—with one or several brokers.

Phase 2: Operation. After the initialization, the participants can actually transfer mobile agents, as shown in Figure 4.

- The agent owner queries the broker for the reference of the executor that it wants to interact with.
- The owner verifies the executor's policy to decide whether it is adequate and checks the manufacturer-issued TPE certificate to decide whether the TPE provides sufficient support to enforce this policy. If any of these checks fail, the owner will abort the protocol.
- If the checks do not fail, the owner sends the agent, encrypted with the TPE public key, to the executor.
- The executor cannot decrypt the TPE-encrypted agent, nor can it do anything other than upload the agent to its TPE.
- The TPE uses its private key to decrypt the message sent by the agent owner, thus obtaining the executable agent, which it will eventually start. The agent can then interact with the executor's local environment or with other agents on the TPE.
- After finishing its task, the agent can migrate back to its owner or to another TPE-certified executor to which it holds a reference.

The problem of protecting the TPE from malicious agents is independent of this approach and must be solved by using other mechanisms. The problem of protecting the TPE from agents that have been tampered with can be solved by concatenating the agent with a hash of the entire agent, including its execution state, before encrypting it. The TPE simply has to verify the correct hash before starting the agent.

Limitations of the Proposed Implementation

Constructing a device that can actually resist tampering is a decidedly nontrivial task. Given sufficient time and resources, an attacker could violate the protection of any device, so this is a weakness in the proposed approach. A disadvantage is that any compromise of the TPE's private key gives an attacker

complete control over the agents sent to this TPE. These problems, however, are very similar to those in areas where the use of tamper-resistant devices is well established, such as debit cards and the subscriber identity module (SIM) cards used for global system for mobile (GSM) communications.

If the TPE's basic physical protection is considered too weak, it could be periodically inspected by an independent appraisal organization with capabilities to detect tampering. This would limit the time an attacker could benefit from a successful attack. It might also provide an effective deterrent if an attempted or successful breaking of a TPE were severely punished. Thus, in the rare case when an attacker does break a TPE, the protection provided by the optimistic approach is still available. As discussed in Anderson and Kuhn,[9] a tamper-resistant device can resist even massive attacks if only detection—not prevention—of tampering is required.

The TPE could become a performance bottleneck. It is possible to alleviate these performance problems by establishing a maintenance contract with the manufacturer, in which it ensures proper operation and adequate performance of a service provider's TPE installation. Such a maintenance contract might not be available to the small service providers who might have the most to gain from the availability of a TPE.

Maintaining and upgrading a tamper-resistant device can be difficult and the device itself may be expensive. All these limitations must be weighed against the perceived advantages.

EXAMPLE SCENARIO

Together, the TPE concept and the CryPO protocol guarantee the integrity of the agent platform to an agent owner. Further, they protect the agent's code and data against manipulation and disclosure, both in transit and during execution. These basic guarantees could be extended by formulating new rules for the TPE policy. The policy itself must support the desired protection, and it must be enforced on the TPE where the agent executes. The policy can be verified with the help of the certificate issued by the manufacturer, who also assures its enforcement.

With this approach, the agent owner does not need to trust the executor; it suffices to trust the manufacturer to properly manufacture and control its TPE so that the claimed guarantees hold. We will now describe how the approach can be used to implement agents with a limited lifetime as well as a mechanism that lets them base their execution on results of previous executions. (For a more detailed discussion of protections that mobile agents can implement, see Wilhelm[10].)

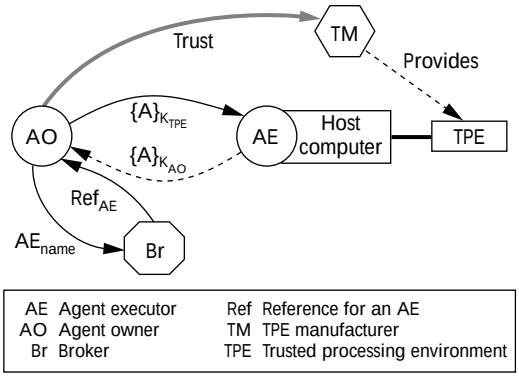

Figure 4. CryPO protocol operations. The agent owner verifies the policy and TPE mechanism of an agent executor through a broker, then uses the TPE public key to transfer agents to the TPE for processing.

The TPE Policy

Consider the case of a shopping agent that contains data for more than one payment method but should disclose data for no more than one of them. For the sake of this discussion, assume furthermore that the service provider's TPE enforces a well-defined set of rules that are detailed in its policy. In the example scenario, the TPE enforces the following six rules:

1. The TPE will never disclose or alter an agent's code.
2. Any invocation of the agent's methods will be executed exactly according to the code in the agent.
3. An agent's data can be accessed and manipulated exclusively through the agent's interface. If the agent does not provide methods to access a data item directly, its value can, at most, be inferred from responses to other method invocations.
4. Before a migration, an agent will obtain the reference to the designated receiver's TPE, which also contains the policy. The agent can decide whether it wants to be transferred, and the current TPE will honor the agent's decision. The actual transfer follows the CryPO protocol.
5. The TPE provides an internal clock with reasonable accuracy (on the order of several seconds). It synchronizes this clock with a trusted time service and tells the agent whether its synchronization was successful.
6. The TPE then provides a small amount of

Related Work in Trusted Mobile Systems

Several researchers have explored the idea of using trusted hardware to ensure a certain behavior of a system.

Herzberg and Pinter[1] describe a device that can be used to protect software against piracy. Chaum and Pedersen[2] describe a wallet architecture that carries a database with personal information and that protects the database from unauthorized access (even from the owner of the wallet). That system incorporates trusted hardware (called the observer) comparable to the hardware proposed in our approach, but it is explored in a very different setting.

A more recent approach by Yee and Tygar[3] has much in common with the one we present here. However, the authors are more interested in the classical issues of ensuring that the system functions securely, while we are more interested in data protection.

Sander and Tschudin[4] describe a completely different approach to code protection that relies on the execution of encrypted functions and does not need trusted hardware. Unfortunately, their approach does not support arbitrarily complex functions and is not sufficiently powerful for our application. A similar idea, presented by Hohl,[5] allows arbitrarily complex functions but guarantees protection only for a certain time interval.

References

1. A. Herzberg and S.S. Pinter, "Public Protection of Software," *Advances in Cryptology: Crypto 85)*, Springer, Berlin, 1985, pp. 158-179.
2. D. Chaum and T.P. Pedersen, "Wallet Databases with Observers," *Advances in Cryptology: Crypto 92*, Lecture Notes in Computer Science, Vol. 740, Springer, New York, 1992, pp. 89-105.
3. B. Yee and D. Tygar, "Secure Coprocessors in Electronic Commerce Applications," *Proc. First Usenix Workshop on Electronic Commerce*, Usenix Assn., Berkeley, Calif., USA, 1995, pp. 155-170.
4. T. Sander and C. Tschudin. "Toward Mobile Cryptography," *IEEE Symp. Security and Privacy*, IEEE Computer Society, Los Alamitos, Calif., 1998, pp. 215-224.
5. F. Hohl. "Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts," *Mobile Agents and Security*, Vol. 1419, Lecture Notes in Computer Science, G. Vigna, ed., Springer-Verlag, Berlin, 1998, pp. 92-113.

nonvolatile storage for a fixed period of time even to an agent that has terminated its execution.

Protecting the Shopping Agent

With this policy in place, it is possible to implement a shopping agent that will not, under any circumstances, reveal more than the data for one payment method.

The agent's overall behavior is this: It finds a suitable offer, queries the service provider for the supported payment methods, chooses from these the preferred owner's method, and finalizes the purchase by disclosing the payment information.

To implement the desired protection, the agent actualizes the policy: Rules 1 and 2 guarantee basic protection of its code and ensure its proper execution; rule 3 guarantees the protection of its data from undesired disclosure and manipulation. Therefore, the owner can rely on the agent's programmed methods not to disclose more than data for a single payment method. Rule 4 guarantees that the agent knows the TPE policy to which it is transferred and will thus not be sent to a TPE that provides insufficient protection.

This level of protection, however, is not yet sufficient for the shopping agent because the executor could store the originally received and encrypted agent before uploading it to its TPE, thereby obtaining the agent's data for the first payment method through normal interaction. The executor could then replace the agent on the TPE with the stored version and conduct another interaction with the agent, this time requesting data for another payment method.

To counter this attack, the agent must know about possible previous executions on the TPE in question. Rule 5 allows an agent containing an expiration date to implement a limited lifetime of, say, a few days or hours. When it arrives, the agent requests the current time and determines whether it is within its attributed lifetime. If its expiration date has passed or the TPE did not successfully synchronize its clock, the agent will abort. This sets a time after which the agent can no longer be executed.

Finally, rule 6 lets the agent store its identity and its chosen payment method, during the limited lifetime, in the TPE's nonvolatile storage. The agent does all of this before it discloses the payment information. If it finds its identity already stored in the TPE, it refuses to disclose more payment information. The limited lifetime of the stored information removes expired entries and prevents a memory overflow in the TPE. The agent can be executed only during its lifetime, and it has information about its previous executions.

If these guarantees are enforced, an agent can be created to use several payment methods but disclose only one of them.

TRUST IN THE MANUFACTURER

The mechanism introduced in the example scenario requires the agent owner to trust the manufacturer to properly design, implement, and produce its TPEs. We know of no way to enforce correct behavior of the manufacturer, so our approach might seem simply to replace one required trust relation-

ship with another. Nevertheless, we believe that this replacement has several subtle implications.

The manufacturer is a specialized service provider that primarily provides security devices and has a good understanding of security and privacy problems. Expert appraisal organizations could control the relatively small number of manufacturers (several hundreds) more easily than they could the large number of possible owners of a TPE (several millions). Further, because TPEs are difficult to produce, manufacturers would be major corporations with resources to build a good reputation. For example, a manufacturer could invite external experts to control its operation, similar to the approach for quality assurance in the ISO-9000. Finally, because the manufacturer and the owner of the TPE are independent, the manufacturer cannot draw a direct benefit from a TPE that does not enforce its policy.

We believe that these arguments of expertise, controllability, good reputation, and lack of incentive to violate policy are sound reasons to trust a manufacturer to build reliable and powerful TPEs. An agent executor, who may offer no incentive to place trust in it, can leverage the trust an agent owner has in a TPE manufacturer, in order to convince the agent owner to start a business transaction with it. This favors the open systems philosophy, by which any principal can become a provider of services.

CONCLUSION

Agent owners currently face a dilemma when they want to interact with an unknown and untrusted service provider that needs confidential information to complete a transaction. We have proposed an approach that lets a service provider define a trust policy, which is actually enforced through an execution environment based on trusted and tamper-resistant hardware purchased from a third-party manufacturer. With this approach, the service provider can benefit immediately from the trust that users have in the TPE manufacturer.

While making it easier for a new service provider to establish itself in the market, this solution also allows an agent owner to protect specific data in a mobile agent. It can thus encourage the construction of open mobile agent systems that allow any principal to become a service provider. ■

ACKNOWLEDGMENTS

Research 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. D. B. Lange and M. Ishima, *Program and Deploying Java Mobile Agents with Aglets*, Addison-Wesley, Boston, 1998.
2. J.E. White, "Mobile Agents," *Software Agents*, J. M. Bradshaw, ed., AAAI Press/MIT Press, Menlo Park, Calif., 1997.
3. V.A. Pham and A. Karmouch, "Mobile Software Agents: An Overview," *IEEE Comm*., Vol. 36, No. 7, 1998, pp. 26-37.
4. D.M. Chess et al., "Itinerant Agents for Mobile Computing," *IEEE Personal Comm*., Vol. 2, No. 5, Oct. 1995, pp. 34-49.
5. C.G. Harrison, D.M. Chess, and A. Kershenbaum, "Mobile Agents: Are They a Good Idea?" *Mobile Object Systems: Toward the Programmable Internet*, Vol. 1222 of Lecture Notes in Computer Science, J. Vitek and C. Tschudin, eds., Springer-Verlag, New York, 1997, pp. 25-47.
6. R.A. Rueppel, "A Formal Approach to Security Architectures," *Proc. EuroCrypt*, Springer, Berlin, 1991, pp. 387-398.
7. T. Sander and C. Tschudin. "Toward Mobile Cryptography," *IEEE Symp. Security and Privacy*, IEEE Computer Society, Los Alamitos, Calif., 1998, pp. 215-224.
8. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Fla., 1997.
9. R. Anderson and M. Kuhn, "Tamper Resistance—A Cautionary Note," *Proc. Second Usenix Workshop on Electronic Commerce*, Usenix Assn., Berkeley, Calif., 1996, pp. 1-11.
10. U.G. Wilhelm, *A Technical Approach to Privacy Based on Mobile Agents Protected by Tamper-resistant Hardware*, doctoral dissertation, École Polytechnique Fédérale de Lausanne, Switzerland, 1999.

Uwe Wilhelm was a researcher and lecturer at the Operating Systems Laboratory (LSE) in the Computer Science Department of the Swiss Federal Institute of Technology (EPFL) at the time this article was written. He has since joined T-Nova, Deutsche Telekom Innovationsgesellschaft mbH, as a researcher and project manager. Wilhelm received a diploma in computer science from the University of Kaiserslautern in 1992 and a PhD from the EPFL in 1999.

Sebastian Staamann is cofounder and co-CEO of Xtradyne Technologies, a company specializing in security middleware for extranets. From 1997 to mid 1999, he led several research projects related to security in middleware-based service platforms in the EPFL's LSE at Lausanne. His research interests are security and high availability.

Levente Buttyan is a research and teaching assistant and a PhD student in the Institute for Computer Communications and Applications (ICA) in the Computer Science Department of the EPFL. He received an MSc in computer science from the Technical University of Budapest in 1995.

Readers may contact the authors at Wilhelm@w9f00992.dmst02.telecom.de and {Sebastian.Staamann,Levente.Buttyan}@epfl.ch.