Integrity of Mobile Agents: A New Approach

Aneta Zwierko[1] and Zbigniew Kotulski[1,2]
(Corresponding author: Aneta Zwierko)
Warsaw University of Technology, Institute of Telecommunication [1]
Warsaw, Poland (Email: azwierko@tele.pw.edu.pl)
Polish Academy of Sciences Institute of Fundamental Technological Research[2]
Warsaw, Poland (Email: zkotulsk@ippt.gov.pl)

(Received Nov. 9, 2005; revised and accepted Jan. 7, 2006)

Abstract

The recent developments in the mobile technology (mobile phones, middleware) created a need for new methods of protecting the code transmitted through the network. The oldest and the simplest mechanisms concentrate more on integrity of the code itself and on the detection of unauthorized manipulation. The newer solutions not only secure the compiled program, but also the data, that can be gathered during its journey and even the execution state. Some other approaches base on prevention rather than detection. This paper describes a new idea of securing mobile agents. The presented method protects all: the code, the data and the execution state. The proposal is based on a zero-knowledge proof system and a secure secret sharing scheme, two powerful cryptographic primitives. The paper also includes security analysis of the new method and comparison to currently most widespread solutions.

Keywords: Agent systems, cryptographic protocols, integrity, mobile code

1 Introduction

A software agent is a program that can exercise an individual's or organization's authority, work autonomously toward a goal, and meet and interact with other agents [11]. Agents can interact with each other to negotiate contracts and services, participate in auctions or barter. Agents are commonly divided into two types: stationary agents and mobile agents.

The stationary agent resides at a single platform (host), the mobile one can move among different platforms (hosts) at different times.

The mobile agent systems offer new possibilities for the e-commerce applications: creating new types of electronic ventures from e-shops and e-auctions to virtual enterprises and e-marketplaces. Utilizing the agent system helps to automate many electronic commerce tasks. Beyond simple information gathering tasks, mobile agents can take over all tasks of commercial transactions, namely price negotiation, contract signing and delivery of (electronic) goods and services. Such systems are developed for diverse business areas, e.g., contract negotiations, service brokering, stock trading and many others [4, 11, 12]. Mobile agent systems have advantages even over grid computing environments:

- require less network bandwidth,
- increase asynchrony among clients and servers,
- dynamically update server interfaces,
- introduce concurrency.

The benefits from utilizing the mobile agents in various business areas are great. However, this technology brings some serious security risks; one of the most important is the possibility of tampering an agent. In the mobile agent systems the agent's code and internal data autonomously migrate between hosts and could be easily changed during the transmission or at a malicious host site. The agent cannot itself prevent this, but different countermeasures can be utilized in order to detect any manipulation made by an unauthorized party. They can be integrated directly into the agent system, or only into the design of an agent to extend the capabilities of the underlying agent system.

There exist several degrees of agent's mobility, corresponding to possibilities of relocating code and state information, including the values of instance variables, the program counter, execution stack, etc. The mobile agent technologies can be divided in to two groups:

- weakly mobile: only the code is migrating; no execution state is sent along with an agent program
- strong mobile: a running program is moving to another execution location (along with its particular state).

In this paper we discuss the agent system mobile in the strong sense.

The system described in our paper is not a standalone solution for agent security but rather part of a big security architecture that would provide agents with such services as: confidentiality, accountability, availability, and authentication.

Organization of this paper: First, we briefly survey the known techniques for protecting agent's integrity (Section 2). In the Sections 3 and 4 we present a new concept for preventing agent's tampering, based on a zero-knowledge proof system. The security of proposed solution is evaluated in Section 5. Section 6 compares different method of protecting agent's code and data, while Section 7 deals with scalability aspects of the security. Conclusions and future work are presented in Section 8.

2 Related Work

Providing security is complex and tough for most existing services. It is even more problematic in distributed environment, such as agents' systems.

The most important security services required in agents' systems are [11]:

- Confidentiality: any private data stored on a platform or carried by an agent must remain confidential. Mobile agents also need to keep their present location and route confidential.
- Integrity: the agent platform must protect agents from unauthorized modification of their code, state, and data and ensure that only authorized agents or processes carry out any modification of the shared data.
- Accountability: each agent on a given platform must be held accountable for its actions: must be uniquely identified, authenticated, and audited.
- Availability: every agent (local, remote) should be able to access data and services on an agent platform, which responsible to provide them.
- Anonymity: agents' actions and data should be anonymous for hosts and other agents; still accountability should be enabled.

Threats to security generally fall into three main classes: disclosure of information, denial of service, and corruption of information [11].

Threats in agent system can be categorized into four groups:

- an agent attacking an agent platform,
- an agent platform attacking an agent,
- an agent attacking another agent on the agent platform,
- other attacks.

The last category covers cases of an agent attacking an agent on another agent platform, and of an agent platform attacking another platform, and also more conventional attacks against the underlying operating system of the agent platform. Other category of attacks are those connected with traffic analysis [13]. In this paper we will focus on the threats from an agent's perspective.

There are two main concepts for protecting mobile agent's integrity:

- detection or prevention of tampering,
- providing trusted environment for agent's execution.

The second group of methods is more concentrated on the whole agent system than on an agent in particular. These seem to be easier to design and implement but, as presented in [14], mostly lead to some problems. The idea that agent works only with a group of trusted hosts makes the agent less mobile than it was previously assumed. Also an agent may need different levels of trust (some information should be revealed to host while in another situation it should be kept secret). Sometimes, it is not clear in advance that current host can be considered as trusted.

A method to provide such an environment is special tamper-resistant hardware, but the cost of such a solution is usually very high.

In this paper we concentrate on the "built-in" solutions because they enable agent to stay mobile in a strong sense (as it is presented in Section 5.1) and, moreover, provide the agent with mechanisms to detect or prevent tampering.

Detection means that the technique is aimed at discovering unauthorized modification of the code or the state information. Prevention means that the technique is aimed at preventing change of the code and the state information in any way. To be effective, detection techniques are more likely than prevention techniques to depend on a legal or other social framework. The distinction between detection and prevention can be arbitrary sometimes, since prevention often involves detection (see [10]).

2.1 Time Limited Black-box Security and Obfuscated Code

These methods are based on a *the black-box* approach. The main idea of *the black-box* is to generate executable code from a given agent's specification that cannot be attacked by read (disclosure) or modification attacks. An agent is considered to be black-box if at any time the agent code cannot be attacked in the above sense, and if only its input and output can be observed by the attacker. Since it is not possible to implement it today, the relaxation of this notion was introduced [14]: it is not assumed that *the black-box protection* holds forever, but only for a certain known time. According to this definition, an agent has the time-limited black-box property

if for a certain known time it cannot be attacked in the above-mentioned sense.

The central idea of this approach is to generate an executable agent from a given agent specification, which cannot be attacked by read or manipulation attacks (see [6]). The *time limited blackbox* fulfills two *black-box* properties for this limited time:

- code and data of the agent specification cannot be read,
- code and data of the agent specification cannot be modified.

This scheme will not protect any data that is added later, although the currently existing variables will be changeable.

In order to achieve *the black-box property*, several conversion algorithms were proposed. They are also called obfuscating or mess-up algorithms. These algorithms generate a new agent out of an original agent, which differs in code but produces the same results.

The *code obfuscation* methods make it more complicated to obtain the meaning from the code. To change a program code into a less easy "readable" form, they have to work in an automatic and parametric manner. The additional parameters should make possible that the same original program is transformed into different obfuscated programs. The difficulty is to transform the program in a way that the original (or a similar, easily understandable) program cannot be re-engineered automatically. Another problem is that it is quite difficult to measure the quality of obfuscation, as this not only depends on the used algorithm, but on the ability of the re-engineering as well.

Since an agent can become invalid before completing its computation, the obfuscated code is suitable for applications that do not convey information intended for long-lived concealment. Also it is still possible for an attacker to read and manipulate data and code but as the role of these elements cannot be determined, the results of this attack are random and have no meaning for the attacker.

2.2 Encrypted Functions

The Encrypted Functions (EF) is one step forward in implementing the perfect black-box security. It has been proposed initially in [16]. Since then other similar solutions were introduced [1, 3, 7, 17] and the method is believed to be a one of canonical solutions for preserving agent's integrity [10, 14].

The goal of Encrypted Functions [10] is to determine a method which will enable the mobile code to safely compute cryptographic primitives, such as digital signature, even though the code is executed in non-trusted computing environments and operates autonomously without interactions with the home platform. The approach is to enable the agent platform to execute a program assimilating an encrypted function without being able to extract the original form. This approach requires differentiation between a function and a program that implements the function.

The EF system is described as follows [14]:

> A has an algorithm to compute function f. B has an input x and is willing to compute $f(x)$ for A, but A wants B to learn nothing substantial about f. Moreover, B should not need to interact with A during the computation of $f(x)$.

The function f can be, e.g., a signature algorithm with an embedded key or an encryption algorithm containing the one. This would enable the agent to sign or encrypt data at the host without revealing its secret key.

Although the idea is straightforward, it is hard to find the appropriate encryption schemes that can transform arbitrary functions as showed. The techniques to encrypt rationale functions and polynomials were proposed. Also the solution based on the RSA cryptosystem was described (see [3]).

2.3 Cryptographic Traces

Giovanni Vigna introduced cryptographic traces (also called execution traces) to provide a way to verify the correctness of the execution of an agent [18, 19]. The method is based on traces of the execution of an agent, which can be requested by the originator after the agent termination, and used as a basis for execution verification. The technique requires each platform involved to create and retain a non-repudiation log or trace of the operations performed by the agent while resident there, and to submit a cryptographic hash of the trace upon conclusion as a trace summary or fingerprint. The trace is composed of a sequence of statement identifiers and the platform signature information. The signature of the platform is needed only for those instructions that depend on interactions with the computational environment maintained by the platform. For instructions that rely only on the values of internal variables, a signature is not required and, therefore, is omitted.

This mechanism allows detecting attacks against code, state and control flow of mobile agents. This way, in a case of tampering, the agent's owner can prove that the claimed operations could never been performed by the agent.

The technique also defines a secure protocol to convey agents and associated security related information among the various parties involved, which may include a trusted third party to retain the sequence of trace summaries for the agent's entire itinerary.

The approach has a number of drawbacks, the most obvious being the size and number of logs to be retained, and the fact that the detection process is triggered sporadically, based on suspicious results' observations or other factors.

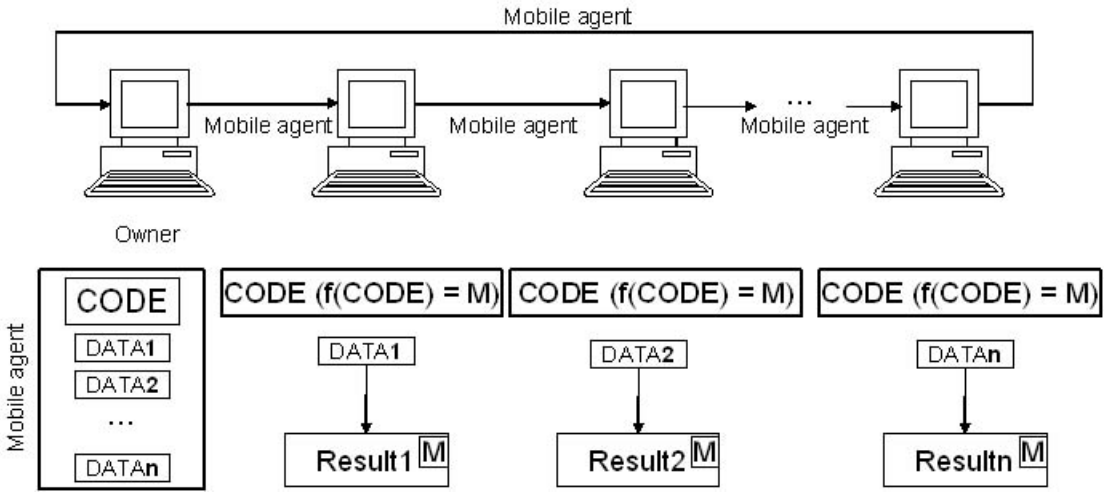

Figure 1: Example of watermarking

2.4 Watermarking and Fingerprinting

Watermarking is mainly used to protect the copyrights for digital contents. A distributor or owner of the content embeds a mark into a digital object, so its ownership can be proved. This mark is usually secret. Most methods exploit information redundancy. Some of them can also be used to protect the mobile agent data and code.

One of methods of watermarking is proposed in [5]. A mark is embedded into the mobile agent by using software watermarking techniques. This mark is transferred to the agents' results during the execution. For the executing hosts the mark is a normal part of results and is invisible. If the owner of agent detects that the mark has been changed (it is different from expected) then he has a proof that the malicious host was manipulating the agent's data or code. Figure 1 illustrates how the mark is appended to data during the mobile agent's computations on various hosts.

The paper [5] presents three ways of embedding the watermark into the agent: marking the code, marking the input data, and marking the obfuscated code.

The mark or marks are validated after the agent returns to its originator.

Fingerprinting, contrary to the watermarking methods presented previously, generate the different embedded mark for each host. When the agent returns to the owner, all results are validated. So the malicious host is directly traced.

The paper [5] presents two ways of embedding the fingerprint into the agent:

- marking the code: in this case, malicious hosts have the possibility of comparing their different codes in order to locate their marks.

- marking the input data: the data are usually different for each host, so it is harder to identify the mark.

The difference between mobile agent watermarking and fingerprinting is the fact that in the second case it is possible to detect collusion attacks performed by a group of dishonest hosts. The procedure is similar to the mobile agent watermarking approach.

3 A New Concept of Integrity Protection

In the proposed system we assume that there exist at least three parties: a manager, an agent, and a host.

The manager can be an originator of the agent. It plays a role of a verification instance in the scheme and creates initial countermeasures for the agent. The manager also plays a role of a Trusted Third Party.

3.1 Basic Idea

The zero-knowledge proof systems (see Section 4.1.1) enable verifier to check validity of the assumption that the prover knows a secret. In our system the verifier would be the manager or owner of agents and, obviously, agents would be the provers. In the initial phase, the manager computes a set of secrets. The secrets are then composed into the agent, so that if the manager asks the agent to make some computations (denote them as a function f), the result of this would be a valid secret. This function should have the following property:

- if we have x_1 and $f(x_1)$ then it is computationally infeasible to find such x_2 that

$$f(x_2) = f(x_1).$$

If the secret is kept within an agent, then also the host can use the zero-knowledge protocol to verify it. Every authorized change of agent's state results in such a change of the secret that the secret remains valid. On the other

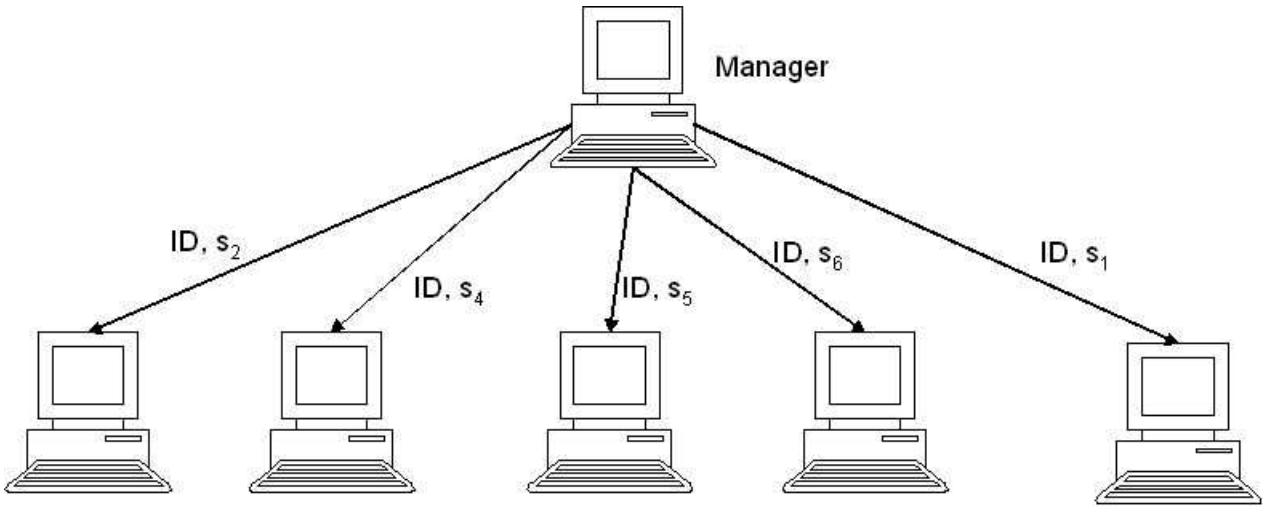

Figure 2: Distributing ID and shares to hosts

hand, every unauthorized change leads to loosing the secret, so at the moment of verification by host or manager, the agent is not able to prove possession of a valid secret. In our system the host can tamper the agent and try to make such changes that he will be still able to obtain the proper secret, but the characteristics of function f will not allow doing this. Some possible candidates for the function f can be a hash function. Our approach is a detection rather than prevention.

4 The Protocol

4.1 Cryptographic Primitives

We utilized two cryptographic primitives in the proposed scheme:

- a zero-knowledge proof (in the form of an identification protocol)
- a secure secret sharing scheme.

Below is a short description of the protocols utilized.

4.1.1 Zero-Knowledge Proof

Zero knowledge proof system [15] is a protocol which enables one party to prove the possession or knowledge of a "secret" to the other party, without revealing anything about it in the information-theoretical sense. Such a protocol is also known as minimum disclosure proof. Zero knowledge proof involves two parties: the prover, who possesses a secret and wishes to convince the verifier (the second party), that he indeed has a secret. The protocol is realized as an interaction between the parties. At the end of the protocol, the verifier should be convinced only if the prover really knows the secret. However, if the prover does not know it, the verifier will be sure of it with an overwhelming probability.

The zero-knowledge proof systems are ideal for constructing identification schemes. A direct use of a zero-knowledge proof system allows unilateral authentication of P (Peggy) by V (Victor) and requires a large number of iterations, so that verifier is convinced, with an initially assumed probability, that the prover knows the secret (or has the claimed identity). This can be expressed as the requirement that the probability of false acceptance is 2^{-n} where n is the number of iterations. The zero knowledge identification protocol reveals no information about the secret held by the prover, under some reasonable computational assumptions.

4.1.2 Secure Secret Sharing Scheme

A (t, n) threshold secret sharing scheme [2, 15] distributes a secret among n participants in such a way, that any t of them can recreate the secret, but any $t-1$ or fewer members gain no information about it. The piece held by a single participant is called a *share* or *shadow* of the secret. Secret sharing schemes are set up by a trusted authority, called a dealer, who computes all shares and distributes them to participants via secure channels. The participants hold their shares until some of them decide to combine shares and recreate the secret. The recovery of the secret is done by the so-called combiner who on behalf of the co-operating group computes the secret. The combiner is successful only if the reconstruction group has at least t members.

4.2 A Host Validating an Agent

Our protocol is not directly based on the complete zero-knowledge proof, but on the particular identification system based on zero-knowledge proof. We choose the Guillou-Quisquater (GQ) scheme [8] as the most convenient for our purposes. In this scheme the manager has a pair of RSA-like keys: a public K_P and a private one k_p.

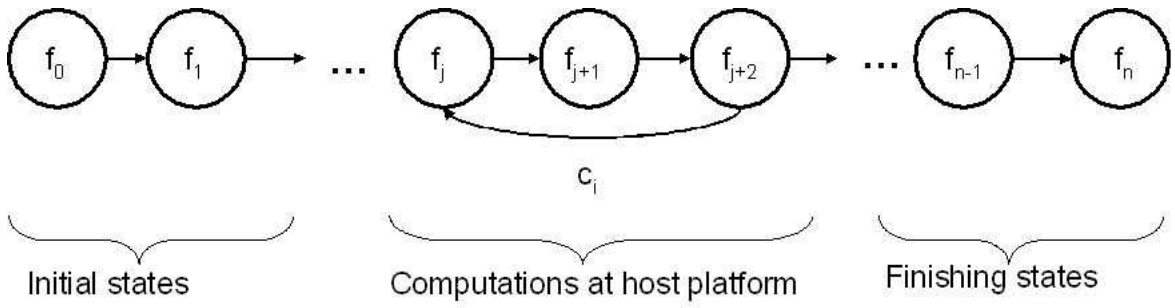

Figure 3: Mobile agent as a *FSM*

The manager also computes the public modulus $N = p \cdot q$, where p, q are RSA-like primes. The following equation has to be true:

$$K_P \times k_p \equiv 1 \pmod{\varphi(N)},$$

where $\varphi(N)$ is the value of Euler function of N. The pair $(K_P,\ N)$ is made public.

4.2.1 The Initial Phase

The initial phase has three steps:

1) The manager computes set of so-called identities, denoted as ID_p, and their equivalences denoted as J_p. It does not matter how J_p is obtained if it is obvious for all participants how to obtain J_p from ID_p. The pairs (ID_p, J_p) are public and can be distributed among hosts. The manager computes a secret value for each ID_p: $\sigma_p \equiv J_p^{-k_p} (\text{mod } N)$. The σ_p is a secret that will be "hidden" in an agent.

2) To compose the σ_p into an agent, the Asmuth and Bloom secure secret sharing scheme [2] is used. The manager randomly chooses m prime or co-prime numbers (called public moduli):

$$p_j \ (j = 1, \ldots, m,\ p_0 < p_j < \ldots < p_m).$$

They are publicly known. Then, the manager (playing the role of a dealer in the secret sharing scheme) instead of selecting at random an integer s, such that

$$\prod_{j=1}^{t-1} p_j < s < \prod_{j=1}^{t} p_j,$$

he computes it, preserving moreover following conditions:

$$s \equiv \sigma_p \pmod{p_0}$$

and

$$p_m < s < \prod_{j=1}^{t} p_j.$$

After computing s, the manager creates appropriate shares:

$$s_i \equiv s \pmod{p_i}.$$

Then, the $t-1$ shares are composed into an agent and the rest are distributed among the hosts via secure channels (this is illustrated in Figure 2).

3) The manager now needs to glue the shares into an agent in such a way, that when the agent is in a proper execution state, he is able to obtain from his code/state variables the correct shares. Since the agent is nothing more than a computer program, it can be described as a *Finite State Machine* (*FSM*). Assume, we have the agent of the form $< \Sigma, S, S_I, S_F, \delta >$, where

   - Σ is the input alphabet,
   - $S = \{f_0, \ldots, f_n\}$ is a set of all possible states,
   - S_I is a subset of S with all initial states,
   - S_F is a subset of S with all finishing states, possibly empty,
   - $\delta : \Sigma \times S \rightarrow S$ is a state transition function.

The Figure 3 shows an example of agent's *FSM*. It is obvious that only some execution states should be observed during the computation at the host platform (e.g., the ones connected with gathering and storing the data). If the state f_j is the first state of agent's computations at the host platform, then it is natural that the shares should be generated only from this state. Additionally, some internal variables that differ for each host, should be utilized to obtain different secrets for each host. Thus, to create agent's shares, $f_j,\ c_i \in \Sigma$, and the code should be used. The idea of generating shares from variables and selected states of agents execution is illustrated in the Table 1.

In other cases, where the pair f_j and c_i is not unique for each host, the previous states or other data should be used. It should be possible to obtain the proper shares for current host basing on appropriate execution state and internal variables. If there are more than one unique combination of (f_j, c_i) for one host, then for each of them the host should obtain an ID and a share. The agent's code (in a certain form) should be a part of the data that is required to recreate the secret to enable detection of every unauthorized manipulation, which could be performed by previous host.

Table 1: Agent's shares

State	Variable	Obtained shares
f_0	...	no proper shares can be generated
...	...	...
f_j	c_i^1	$h(f_i, c_i^1, code) \rightarrow \{s_1^1, \ldots, s_{t-1}^1\}$, (shares for the 1^{st} host)
f_j	c_i^2	$h(f_i, c_i^2, code) \rightarrow \{s_1^2, \ldots, s_{t-1}^2\}$, (shares for the 2^{nd} host)
...	...	...
f_j	c_i^l	$h(f_i, c_i^l, code) \rightarrow \{s_1^l, \ldots, s_{t-1}^l\}$, (shares for the l^{th} host)
...	...	...
f_{n-1}	...	no proper shares can be generated
f_n	...	no proper shares can be generated

To create the shares from the mentioned data, the hash function or an encryption function with the manager's public key can be used.

4.2.2 The Validation Phase

The Figure 4 shows the general steps of the protocol of deciding by the host if the migrated agent is valid.

1) The host, which wants to verify an agent's integrity, sends him his share s_h.

2) The agent creates the rest of the shares from his code and the execution state. He recreates the secret (playing a role of a combiner for the Asmuth and Bloom secure secret sharing scheme) by solving the following system of equations:

$$s \equiv s_{i_1} \pmod{p_{i_1}}$$

$$\ldots$$

$$s \equiv s_{i_t} \pmod{p_{i_1}}$$

This system has an unique solution according to the Chinese Reminder Theorem. The agent computes the secret σ and uses it for the rest of the scheme, which is a zero-knowledge proof identification protocol.

3) The agent sends the host a challenge: a number u computed basing on a random value r, $r \in \{1, \ldots, N-1\}$,

$$u = r^{K_P} \pmod{N}.$$

4) After receiving the challenge the host chooses a random value $b \in \{1, \ldots, N\}$ and sends it to the agent.

5) The agent computes next value (v) basing on the number b obtained from the host and on agent's secret value σ:

$$v \equiv r \times \sigma^b \pmod{N}.$$

6) The host uses information received from the manager, ID_p to obtain J_p and verifies if v is a proper value. To validate the response received from the agent, the host checks if

$$J_p^b \times v^{K_P} \equiv u \pmod{N}.$$

If the equation is true, then the agent proved that he knows the proper secret and, what follows, neither his code, nor execution state were changed.

The manager can compute many identities, which may be used with different execution states. In that situation the agent should first inform host which identity should be used, or the host can try to validate the received value v for all possible identities. The second part of this protocol, starting from the agent sending a challenge to the host, can be repeated to minimize the probability of not detecting prohibited manipulations in the agent's code.

4.3 Securing the Data Obtained by an Agent

A similar scenario can be used to provide integrity to the data obtained by the agent from different hosts. A malicious host could try to manipulate the data delivered to agent by the previously visited hosts. To ensure that this is not possible, the agent can use the zero-knowledge protocol to protect the data. For each stored data d, the agent can choose at random $r \in \{1, \ldots, N-1\}$ and compute

$$v \equiv r \times \sigma^d \pmod{N}.$$

Then, the manager can verify the data by computing and comparing:

$$J_p^d \times v^{K_P} \equiv u \pmod{N}.$$

That way for every received data d the agent would have an unique "proof" that the data was not manipulated.

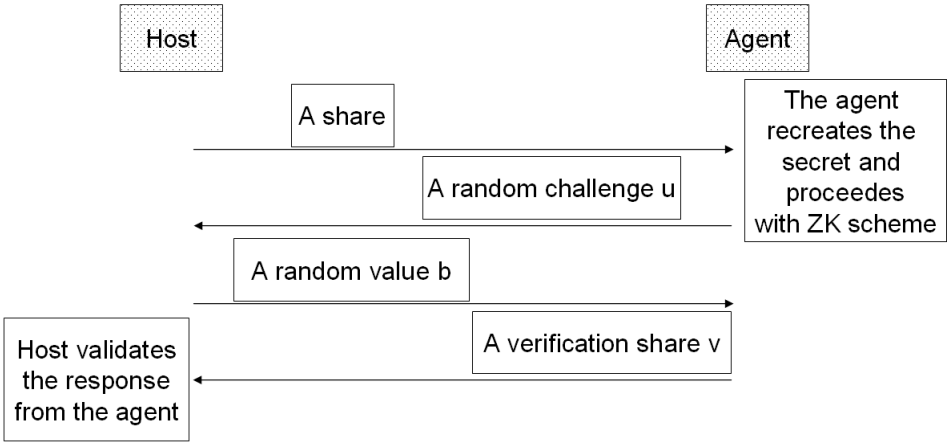

Figure 4: The verification of an agent's integrity

5 Security Analysis

5.1 Definitions and Notions

This section presents basic notions concerning agent's integrity that will be later used in description of the selected solutions (most of the definitions come from [6]).

The integrity of an agent means that neither its code nor execution state can be changed by an unauthorized party or such changes should be detectable (by an owner, a host or an agent platform, which want to interact with the agent).

The authorized changes occur only when the agent have to migrate from one host to another. Below is a more formal definition:

Definition 1 (integrity of an agent). *An agent's integrity is not compromised if no unauthorized modification can be made without the agent's owner noticing this modification.*

The concept of forward integrity is also used for evaluation of many methods. This notion is used in a system where agent's data can be represented as a chain of partial results (a sequence of static pieces of data). Forward integrity can be divided into two types, which differ in their possibility to resist cooperating malicious hosts. The general goal is to protect the results within the chain of partial results from being modified. Given a sequence of partial results, the forward integrity is defined as follows:

Definition 2 (weak forward integrity). *If i_n is the first malicious agent place on the itinerary, the integrity of each partial result $m_0, ..., m_{n-1}$ is provided.*

Weak forward integrity is conceptually not resistant to cooperating malicious hosts and agent places that are visited twice. To really protect the integrity of partial result, we need a definition without constraints.

Definition 3 (strong forward integrity). *None of the encapsulated messages m_k, with $k < n$, can be modified with out notifying the manager.*

In this paper we refer to forward integrity as to strong forward integrity (when applicable). To make notion of forward integrity more useful, we define also publicly verifiable forward integrity, which enables any host to detect compromised agents:

Definition 4 (publicly verifiable forward integrity). *Any host i_n can verify that the chain of partial results $m_{i_0}, \ldots, m_{i_n}$ has not been compromised.*

The other important notion concerning agent's integrity, a concept of *black-box security* (see [9, 14]) was introduced in Section 2.1.

5.2 Analysis

The proposed scheme should be used with more that one identity (ID_p). This would make very hard to manipulate the code and the data. The best approach is to use one secret for each host.

We assume that the malicious host is able to read and manipulate an agent's data and code. He can try to obtain from an agent's execution state the proper shares. He can also try to obtain a proper secret and manipulate the agent's state and variables in a way that the obtained secret would stay the same. But he does not know other secrets that are composed into the agents; also he does not know more shares to recreate those secrets, so, any manipulation would be detected by the next hosts.

Also even when host is able to recreate the current secret, he is not able to manipulate the data that was obtained by the agent earlier from other hosts. Since he cannot produce a valid secret σ for the given data d, he is not able to forge the v, the way that using a zero-knowledge proof would not reveal the changes.

The protocol is not able to prevent any attacks that are aimed at destroying the agent's data or code, meaning that a malicious host can "invalidate" any agent's data. But this is always a risk, since the host can simply delete an agent.

Weak forward integrity: The proposed method posses the *weak forward integrity* property: the malicious host cannot efficiently modify previously generated results.

Strong forward integrity: The protocol provides agent also with *strong forward integrity*, because host cannot change previously stored results (without knowledge of secrets created for other hosts). He cannot also modify the agent in a way that could be undetectable by the next host on the itinerary or by the owner.

Publicly verifiable forward integrity: Each host can only verify if the agent's code or the execution state has not been changed. They cannot check wherever the data obtained on other platforms has not been modified. This can be only done by the agent's owner, who created all secrets.

Black-box security: The proposed system is not resistant to read attacks. The code or data can be modified by a malicious host, but it is detectable by agent's owner, so it is resistant to manipulation attack. The system does not have full *black-box* property.

6 Comparison with Other Methods

It is a difficult task to compare systems based on such different approaches as presented here. We decided to split comparison into two categories:

- practical evaluation (Table 2): if the method is hard or easy to implement:
  - hard: no practical implementation exists at the moment;
  - medium: the method has been implemented, with much effort;
  - easy: the method is widely used and has been implemented for different purposes;

  and what elements of agent it protects,
- theoretical evaluation (Table 3): if the method satisfies the security definitions from Section 5.1.

The theoretical evaluation is quite hard, because some methods that have the *black-box* property, does not "fit" to other definitions. If the code or data cannot be read or manipulated (the ideal case), then how we can discuss if it can be verifiable, or, if it fulfills the forward integrity.

As for evaluation of the *black-box* property, it is very hard to provide the code that cannot be read. In all cases, marked by $*$, the adversary can modify the agent but not in a way that owner or other host would not notice. This means that no efficient manipulation attack can be made, so one part of the *black-box* property is satisfied.

In # case the *publicly verifiable forward integrity* is satisfied only partially, because the agent's code can be verified but the data cannot.

7 Scalability

Our solution has two main phases: the initialization phase and the operating phase.

The initialization phase. The first phase is similar to the bootstrap phase of the system. The hosts and the manager create a static network. It is typical for agents systems that all hosts are known to the manager or the owner of an agent, so distribution of all IDs and shares is efficient. We can compare this to sending a single routing update for entire network as in OSPF protocol (the flooding). Whenever a new agent is added to the system, the same amount of information to all hosts have to be sent. Since the messages are not long (a single share and few IDs) and are generated only during creating a new agent, that amount of information should not be a problem. The sizes of parameters (keys lengths, number of puzzles, number of shares) are appropriate adjusted to the agents' network size.

The operating phase. During the validation phase no additional communication between the manager and the hosts is required.

8 Concluding Remarks

One area for development is to find the most appropriate function for composing secrets into hosts: the proposed solution fulfills the requirements, but some additional evaluation should be done.

The next possibility for the future work would be to integrate the proposed solution to some agents' security architecture, possibly the one that would also provide an agent with strong authentication methods and anonymity [20]. Then, such a complex system should be evaluated and implemented. At the moment the agent system equipped with the anonymous authentication and the proposed integrity mechanisms is under development: it is an electronic elections system for mobile devices.

This paper provides description of various protocols and methods for preserving the agent's integrity. The basic definitions and notions were introduced. The most important mechanisms are introduced and discussed. We also proposed a new concept for detection of the tempering an agent, based on a zero-knowledge proof system.

Table 2: Practical comparison of integrity protection methods

Method	Implementation	Protects code	Protects data	Protects execution state
Encryption functions	Hard	Yes	Yes	No
Obfuscated code	Medium	Yes	No	No
Cryptographic traces	Hard	Yes	No	Yes
Watermarking	Easy	Yes	Yes	No
Fingerprinting	Easy	Yes	Yes	No
Zero-knowledge proof method (this paper)	Easy	Yes	Yes	Yes

Table 3: Theoretical comparison of integrity protection methods

Method	Weak forward integrity	Strong forward integrity	Publicly verifiable forward integrity	*Black-box* property
Encryption functions	No	No	No	Yes
Obfuscated code	Yes	Yes	No	partially∗
Cryptographic traces	Yes	Yes	Yes?	No
Watermarking	Yes	No	No	partially∗
Fingerprinting	Yes	Yes	No	partially∗
Zero-knowledge proof method (this paper)	Yes	Yes	No#	partially∗

The proposed scheme secures, both, an agent's execution state and the internal data. The system requires some additional research and development work, but it seems to be a promising solution to the problem of providing agent with effective countermeasures against attacks on its integrity.

References

[1] J. Algesheimer, C. Cachin, J. Camenisch, and G. Karjoth, *Cryptographic Security for Mobile Code*, IBM Security Resaerch, 2000.

[2] C. Asmuth, and J. Bloom, "A modular approach to key safeguarding," *IEEE Transactions on Information Theory*, IT-29, no. 2, pp. 208-211, 1983.

[3] M. Burmester, V. Chrissikopoulos, and P. Kotzanikolaou, "Secure transactions with mobile agents in hostile environments," in *Information Security and Privacy, Proceedings of the 5th Australasian Conference ACISP*, LNCS 1841, pp. 289-297, Springer-Verlag, 2000.

[4] A. Corradi, M. Cremonini, R. Montanari, and C. Stefanelli, "Mobile agents integrity for electronic commerce applicatons," *Informations Systems*, vol. 24, no. 6, pp. 519-533, 1999.

[5] O. Esparza, M. Fernandez, M. Soriano, J. L. Munoz, and J. Forne, "Mobile agents watermarking and fingerprinting: Tracing malicious hosts," *DEXA 2003*, LNCS 2736, pp. 927-936, Springer-Verlag, 2003.

[6] L. Fischer, *Protecting Integrity and Secrecy of Mobile Agents on Trusted and Non-Trusted Agent Places*, Diplomarbeit, Universitat Bremen, Fachbereich Informatik, 2003.

[7] J. A. Foss, S. S. Harrison, and L. Hyungjick, "The use of encrypted functions for mobile agent security," in *Proceedings of the 37th Hawaii International Conference on System Sciences*, pp. 297-306, 2004.

[8] L. C. Guillou and J. J. Quisquater, "A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory," *Advances in Cryptology-EUROCRYPT'88*, LNCS 0330, pp. 123-128, Springer-Verlag, 1988.

[9] F. Hohl, "Time limited blackbox security: Protecting mobile agents from malicious hosts," in *Mobile Agents and Security*, LNCS 1419, pp. 90-111, Springer-Verlag, 1998.

[10] W. A. Jansen, *Countermeasures for Mobile Agent Security*, NIST publication.

[11] W. A. Jansen, and T. Karygiannis, *NIST Special Publication 800-19 - Mobile Agents Security.*

[12] K. Kulesza and Z. Kotulski, "Decision systems in distributed environments: Mobile agents and their role in modern e-commerce," *INFORMATION IN XXI CENTURY SOCIETY*, w: A. Lapinska, [ed.], Warmia-Mazury University Publishing, Olsztyn 2003, ISBN 83-89112-60-4.

[13] K. Kulesza, Z. Kotulski, and K. Kulesza, "On mobile agents resistant to traffic analysis," *Electronic Notes in Theoretical Computer Science*, vol. 142, pp.

1-254, 3 Jan. 2006, Proceedings of the First International Workshop on Views on Designing Complex Architectures (VODCA 2004), Bertinoro, Italy, 11-12, Edited by M. ter Beek and F. Gadducci, pp.181-193, Elsevier, ISSN: 1571-0661. Sep. 2004.

[14] R. Oppliger, *Security technologies for the World Wide Web*, the Computer Security Series, Artech House Publishers, 2000.

[15] J. Pieprzyk, T. Hardjono, and J. Seberry, *Fundamentals of Computer Security*, Springer-Verlag, Berlin 2003.

[16] T. Sander and C. F. Tschudin, "Towards mobile cryptography," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 215-224, 1998.

[17] T. Sander and C. F. Tschudin, "Protecting mobile agents against malicious hosts," in *Mobile Agents and Security*, LNCS 1419, pp. 44-60, Springer-Verlag, 1998.

[18] G. Vigna, "Protecting mobile agents through tracing," in *Proceedings of the 3rd ECOOP Workshop on Mobile Object Systems*, pp. 1-14, Jyvlskyl, Finland, Jun. 1997.

[19] G. Vigna, "Cryptographic traces for mobile agents," in *Mobile Agents and Security*, LNCS 1419, pp. 137-153, Springer-Verlag, 1998.

[20] A. Zwierko and Z. Kotulski, "Mobile agents: Preserving privacy and anonymity," in *Proceedings of IMTCI2004, International Workshop on Intelligent Media Technology for Communicative Intelligence in Warsaw*, Poland, pp. 189-192, Sep. 13-14, 2004, ISBN 83-89244-30-6, extended version published in LNAI 3490, pp. 246-258, Springer, Heidelberg 2005, ISBN 3-540-290-35-4.

Aneta Zwierko is presently doing her PhD on application of cryptographic protocols in mobile environment at Warsaw University of Technology, Institute of Telecommunications. Her current interest include zero-knowledge proofs and its application, identification and authentication protocols, anonymity and privacy, security issues of the agent systems, E/M-voting protocols, electronic payments, AI and its application in security. She holds MSc in telecommunications from Warsaw University of Technology, Poland.

Zbigniew Kotulski received his M.Sc. in applied mathematics from Warsaw University of Technology and Ph.D. and D.Sc. Degrees from Institute of Fundamental Technological Research of the Polish Academy of Sciences. He is currently a professor at IFTR PAS and professor and head of Security Research Group at Department of Electronics and Information Technology of Warsaw University of Technology, Poland.