A Configurable Mobile Agent Data Protection Protocol

Paolo Maggi
paolo.maggi@polito.it

Riccardo Sisto
riccardo.sisto@polito.it

Dipartimento di Automatica ed Informatica
Politecnico di Torino
Corso Duca degli Abruzzi, 24
I-10129 Torino, ITALY

ABSTRACT

This paper addresses the problem of protecting the data carried by mobile agents from the possible attacks of malicious execution hosts. Specifically, we consider protection mechanisms that, instead of preventing malicious hosts from tampering with the collected data, simply aim at detecting tampering attempts. The various proposals that appeared in the literature so far are characterized by several limitations, some of which have been pointed out recently. In particular, data truncations may not be detected, and a binding of the static code to the collected data is missing. This paper takes into account such criticisms and formally defines a new protocol that does not suffer from all the previous limitations. Such a protocol is also configurable, according to the protection level needed. In this way, the minimum protection level compatible with the needed security properties can be selected each time.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Security, Theory

Keywords

Mobile agents, cryptographic protocols, data integrity, data authenticity

1. INTRODUCTION

The development of several kinds of distributed software applications can benefit from the use of mobile agents. For instance, a promising application domain is electronic commerce on the Internet. Researchers envision, for example, mobile agents dispatched to visit the sites of different companies in order to find out the price at which they sell a given product so as to select the cheapest one.

However, to make the use of the mobile agent paradigm really acceptable, it is necessary to face all the numerous security threats that arise from it. Such threats generally fall into two main categories. On one hand, it is necessary to protect hosts from malicious agents coming from the network, and, on the other hand, it is necessary to protect agents from malicious hosts and network intruders.

As noted in [6], research efforts in the field of mobile agent security have been quite intense between 1996 and 1998, but they have recently become weaker. This however does not necessarily mean that all the main problems have been solved. In fact, while satisfactory solutions are already available to protect hosts from malicious agents, some open issues remain in the protection of agents from malicious hosts.

Researchers have envisaged several different protection mechanisms for mobile agents, mainly aiming at confidentiality and integrity of the data carried by the agent. The various proposals share many common ideas and characteristics. While confidentiality can be obtained by encrypting the data with the public key of the intended recipient, integrity is more difficult to achieve, because an hostile execution environment has full control over the agent operations. Among the various techniques proposed, a lot of attention has been paid to the ones that, using cryptography, provide a way for detecting tampering attempts on the collected data, rather than preventing such attempts.

All the proposed protocols that follow this approach have two main limitations. On the one hand, as pointed out by Roth in [5, 6], they lack (or do not specify) a mechanism to bind the dynamic data of an agent to its static data (i.e. its code and initial parameters). The absence of such a binding enables several attacks, based on stripping out the data from an agent and using them in other similar agents. On the other hand, as recognized by the various protocol authors, the proposed mechanisms are not resistant to all the kinds of tampering: typically, truncations of the collected data in some cases cannot be detected (e.g. [1, 2, 3, 4, 7, 10]). Moreover, some of the proposed protocols have been designed specifically for particular application domains and work correctly only if certain environment assumptions hold (e.g. [1, 2, 3]). In some cases (e.g. [10]), the protocols have been specified only in an informal, incomplete way.

The main goal of this paper is to put together the underlying ideas of various proposals that appeared in the literature, and the related criticisms, and to formally define a protocol that incorporates them and does not suffer from the previously mentioned limitations. In particular, solutions to the problem of binding the static and dynamic parts of a mobile agent have already been sketched in [6] and proposed in [7], while, for what concerns resistance to truncations, this paper shows how an improvement can be achieved by forcing the agent to securely store the addresses of the next hosts to be visited, as long as they are computed or become available. This can be done using a technique similar to the one used for storing the other collected data.

The paper is not restricted to any particular application domain and also considers free-roaming agents, i.e. agents that are free to autonomously choose the next host to visit at each step. No particular assumptions are made about the agent functionalities. Simply, it is assumed that the agent collects (or computes) some data at each step. The objective of the protocol is to ensure some form of confidentiality, authenticity, integrity or non-repudiability on the collected data.

Another issue that is addressed in this paper is protocol configurability, according to the needs of the applications. So far, various different protocols enjoying different sets of security properties have appeared in the literature. However, each of them provides only a particular combination of properties, which does not necessarily fit exactly the needs of specific applications. Since protection mechanisms have a cost, it would be quite useful to have a single protocol that can be configured according to the specific security needs of the application under development, but always maintaining the same base structure.

In this paper the configurability issue is addressed by defining a basic protocol message structure with a set of fields that can take slightly different forms, according to the required properties.

The rest of the paper is organized as follows. Section 2 precisely states our initial assumptions and presents the formal notation used in this paper to describe mobile agent protocols, which is similar to the one introduced in [5]. Then, section 3 defines the security properties for mobile agents that are addressed here, and section 4 recalls the protocols that have been previously proposed in the literature, discussing the extent to which they meet such properties and pointing out their limitations. Section 5 describes the base structure of our configurable protocol. Subsequently, some different possible choices of the configurable fields are specified, and the resulting protocol instances discussed. Section 6 concludes and points out further research issues.

2. NOTATION AND INITIAL ASSUMPTIONS

A mobile agent is a program that can move from host to host during its execution. In this paper, when describing the execution of a mobile agent, i_n ranges over the visited hosts: i_0 denotes the initiator host, i.e. the host that initially creates the agent, whereas $i_1, \ldots, i_k$ denote the k hosts where the agent is subsequently executed, in order of visit. It is assumed that each mobile agent finally returns back to its originator, where its execution terminates. In our agent execution description, this means that, after having visited the k-th host i_k, the agent returns back to i_0 where it terminates. So, the entire agent lifetime is divided into $k+2$ slots, executed respectively on hosts $i_0, i_1, \ldots, i_k, i_0$. It is worth noting that in principle this execution model does not exclude that a non-initiator host is visited more than once, i.e. $i_1, \ldots, i_k$ are not necessarily pairwise distinct. It is also worth pointing out that this execution model does not represent the ideal, or wanted agent behavior, but it represents any real agent execution which terminates, including executions affected by malicious activities. The only agent executions that are not represented by this model are the ones that do not terminate, i.e. the ones where the agent fails to come back to its initiator.

Each mobile agent is composed of a static (immutable) part, which includes both the agent code and any constants used by the agent, and a dynamic part, which changes during agent execution. The static part is denoted Π.

The dynamic part is assumed to be divided into two areas: one of them collects the relevant results of the computations carried out by the agent, while the other one includes only scratch variables used during the agent computations. Only the former section is cryptographically protected and transferred with the agent from host to host. Consequently, the agent cannot rely on the contents of its scratch variables when it moves from host to host. In practice, we assume that scratch variables are cleared during each move, and agents are programmed so as to move to a new host only after having saved in the protected area any data that may subsequently be needed. It is assumed also that the protected area functionally behaves as an add-only container, where new data can be added by the agent during its execution.

The data added while the agent is visiting host i_n is denoted d_n. If the agent has no data to add when visiting a host, it is required to add a dummy data item before leaving the host.

From a formal point of view, the protected area is treated as a *set* of data items, and not as an ordered sequence. However, this is not a limitation: if an ordering on the collected data is needed, this can be achieved including an order number in each data item d_n. Similarly, if it is required that each collected data item be unique, it is enough to include a unique identifier in it (e.g. the host identifier plus a progressive number).

When the agent moves from host i_n to host i_{n+1}, a message M that includes the agent in its current state is built and sent from host i_n to host i_{n+1}. This is written $i_n \rightarrow i_{n+1} : M$. Of course, M includes both the agent static part Π and the protected area of the agent dynamic part. The data protection mechanisms that are considered here do not aim at prevention, but only at detection of tampering. Moreover, mechanisms with various protection levels are considered. Therefore, it is always possible that, because of malicious activities carried out by malicious hosts or network intruders, the set of data items contained in the protected area of the agent when it reaches back host i_0 differs from the set of data $\{d_1, \ldots, d_k\}$ gathered by the agent during its lifetime. For example, it is possible that an agent adds d_1 to its protected area on host i_1, then it visits a malicious host i_2 where d_1 is deleted, and finally reaches back i_0 without carrying d_1 in its protected area. Of course, the protection mechanism should enable i_0 to detect such situations. The set of data items contained in the protected area of the agent when it reaches back host i_0 is denoted $\{d'_1, \ldots, d'_{k'}\}$. It can be noted that we consider the most general case where the final number k' of data items in the protected area can differ from k.

In protocol descriptions, the encryption of a *plaintext* m by a *key* K is denoted $\{m\}_K$. The digital signature of m by host i_n is written as an encryption with a private signing key $S_{i_n}^{-1}$, i.e. $\{m\}_{S_{i_n}^{-1}}$. To denote the bare signature (rather than the union of the signature and the signed data) the notation $\mathcal{S}_{i_n}^{-1}(m)$ is used. We also assume that the identity of the signer can always be extracted from her signature.

Concatenation of m_1 and m_2 is denoted as $m_1 \parallel m_2$.

3. SECURITY PROPERTIES

In this section we define a set of security properties related to the data carried by a mobile agent with originator i_0 which visits hosts $i_1, \ldots, i_k$ and then returns back to i_0.

Some properties are defined with respect to one of the data items put in the agent protected area. If such an item is denoted d', such properties can be defined as follows:

- *Data Confidentiality*: d' can be read on host i_0 only.
- *Origin Confidentiality, or Forward Privacy*: the identity of the host where d' has been added to the protected area of the agent can be determined by host i_0 only.
- *Data Authenticity*: after having received back the agent, host i_0 can determine for sure the identity of the host where data d' has been added to the protected area.
- *Data Non-Repudiability*: after having received back the agent, host i_0 can build a proof about the identity of the host where d' has been added to the protected area. Note that data non-repudiability implies data authenticity.

Data integrity properties are defined with respect to the whole set of data actually collected by the agent ($\{d_1, \ldots, d_k\}$) and with respect to the set representing the received protected area ($\{d'_1, \ldots, d'_{k'}\}$).

We call *truncation at* j the particular integrity attack where the protected area is restored to the state that it had when the agent left host i_j (thus eliminating all subsequently added items). If the attacker performs just one truncation at j immediately after the agent has visited host i_m, the set of data representing the final protected area contents is

$$\{d_1, \ldots, d_j, d_{m+1}, \ldots, d_k\}$$

We call *cancellation* the particular integrity attack where a data item is eliminated from the set representing the protected area contents. We call *insertion* the particular integrity attack where a new data item is added to the set representing the protected area contents. Finally, we call *substitution* a cancellation followed by an insertion of a data item different from the canceled one.

The strongest data integrity property can be expressed as follows

- *Strong Data Integrity*: After having received back the agent, host i_0 can detect if $\{d'_1, \ldots, d'_{k'}\} \neq \{d_1, \ldots, d_k\}$.

Strong data integrity implies that the originator can detect any insertion and/or cancellation of the set of data added to the protected area of the agent during its execution.

Since strong data integrity is quite difficult to achieve, weaker integrity properties have been defined in the literature. We consider the following definitions:

- *Weak Data Forward Integrity*: let $i_1, ..., i_m$ be the *honest prefix* path of the agent, i.e. the path that starts from the initiator and ends before the first malicious host visited. Then, after having received back the agent, host i_0 can detect any cancellation of the data collected by the agent in its honest prefix path, i.e. it can detect whether $\{d_1, \ldots, d_m\} \nsubseteq \{d'_1, \ldots, d'_{k'}\}$ This property was initially introduced in [10].
- *Strong Data Forward Integrity*: after having received back the agent, host i_0 can detect any attack where, for some $1 \leq j < m \leq k$ data d_j is substituted in the protected area after the agent has visited a trusted host i_m. This property was initially introduced in [3].
- *Strong Data Truncation Resilience*: after having received back the agent, host i_0 can detect any truncation attack on the collected data. With our formalization, this property can be specified as follows: whenever the received protected area has the contents $\{d_1, \ldots d_j, d_{m+1}, \ldots, d_k\}$ for some $1 \leq j < m \leq k$, i_0 can detect that the agent data integrity has been violated.
- *Data Truncation Resilience*: this is a weaker form of the previous property, requiring that a truncation attack be detected unless it restores the state that the agent had when visiting a malicious host. Formally, if the received protected area has the contents

$$\{d_1, \ldots d_j, d_{m+1}, \ldots, d_k\}$$

  for some $1 \leq j < m \leq k$, with i_j trusted, then i_0 can detect that the agent data integrity has been violated. This property was initially introduced in [3].
- *Trusted Data Integrity*: after having received back the agent, host i_0 can detect any cancellation of the data items collected by the agent on trusted hosts, i.e., if I_t is the set of trusted hosts, it can detect whether $\{d_i \mid i_i \in I_t\} \nsubseteq \{d'_1, \ldots, d'_{k'}\}$

A property is *publicly verifiable* when any intermediate host or the agent itself can verify if the property holds.

4. RELATED WORK

Several different protection mechanisms aiming at confidentiality and integrity of the data carried by a mobile agent have been proposed in the literature. Among them, we are concerned with the ones that aim at providing a way for detecting tampering attempts (without making use of additional trusted hardware).

In [10], Yee proposed several cryptographic approaches to the agent security problem. The first mechanism proposed in [10] is the so-called *per-server digital signature* and consists in signing the data gathered by the agent on each host with the host private key. This protocol aims at achieving *data authenticity* and *data non-repudiability* but clearly it does not address data integrity, because a malicious host can remove previously gathered data from the agent protected area without being detected.

In his paper, Yee also proposed a protection mechanism based on Partial Result Authentication Codes (PRACs). This mechanism can be used to guarantee *weak data forward integrity* and consists in encapsulating the agent state at each host by using cryptographic checksums (called PRACs) computed by means of secret key cryptography. This technique requires the agent and its originator to maintain or incrementally generate a list of secret keys used in the PRAC computation. Each key must be used only once and the agent must destroy it before migrating to the next host. According to Yee, PRACs should enable the initiator to identify the first malicious host, and so trust only the data gathered before visiting that host. However, it is not clear how this identification can take place in practice.

An enhanced mechanism using publicly verifiable PRACs is also suggested in [10]. This one makes use of public key cryptography and digital signatures instead of relying on secret keys and achieves (publicly verifiable) *data authenticity* too.

In [3], Karjoth et al. reformulate and improve the Yee's proposals. In particular, they introduce a family of protocols which aim at preserving the integrity and confidentiality of data acquired by free-roaming agents. All the proposed protocols are based on the same idea: binding each data to all the previously gathered data and to the identity of the subsequent host to be visited. However, as described in a detailed way by Roth in [5], such protocols still lack a mechanism to bind the dynamic data of an agent to its static data, which makes them all subject to attacks based on separating agent code from agent data.

In the Publicly Verifiable Chained Digital Signature Protocol (P1), each host digitally signs the data it provides using its private key and uses a secure hash function to link its data to the previously gathered data and to the identity of the next host. In this way, if a mechanism for binding the static and dynamic agent parts is provided, *(publicly verifiable) strong data forward integrity* is achieved, i.e. an host cannot modify a previously gathered data even when it was provided by the host itself or by a colluding untrusted host[1].

Furthermore, the protocol accomplishes *data confidentiality* by encrypting data with the public key of the agent's originator and *data non-repudiability.* The protocol does not achieve any kind of data truncation resilience. In fact, being the data involved in all the chaining relations known, a malicious host i_n can easily remove all data $\{d_j, \ldots, d_{n-1}\}$ gathered by the agent on hosts $i_j, \ldots, i_{n-1}$ with $0 < j < n-1$ and send the agent to host i_j again without being detected.

The Chained Digital Signature Protocol with Forward Privacy (P2) is a variation of protocol P1 with the order of encryption and signature computation being swapped. The goal of this arrangement is to hide the identity of the hosts that provided the data so as to achieve *forward privacy* (instead of publicly verifiable strong data forward integrity)

The Chained MAC Protocol (P3) extends the Yee's PRAC based mechanism using the same idea on which P1 and P2 are based. Differently from previous protocols, it requires that each pair of hosts be connected via a confidential channel. It achieves the same properties of P2 except data non-repudiability. It also achieves *data truncation resilience.*

The last protocol presented in [3] is the Publicly Verifiable Chained Signatures protocol (P4). It extends P3 by making PRACs publicly verifiable.

[1]This is not completely true since if i_{n-1} and i_n are untrusted cooperating hosts then they can modify d_{n-1} without breaking the chain.

It is worth noting that since P3 and P4 do not achieve data authenticity, their security relies on the quite strange assumption that an attacker does not change the last element in the chain.

Despite Karjoth et al.'s protocols represent a big improvement with respect to Yee's proposal, they still have some important limitations. On one hand, as already pointed out, they lack a mechanism to bind the dynamic data of an agent to its static data. For this reason, legitimate hosts can be abused by malicious hosts as oracles that decrypt, sign or compute data on behalf of an adversary. On the other hand, as acknowledged by the authors themselves, these protocols do not achieve *strong truncation resilience.* If two untrusted hosts i_n and i_j (with $j < n$) conspire or if i_j and i_n are the same host (i.e. the agent visits the same host twice), they can truncate the chain of gathered data at j without being detected. Furthermore, in the case of protocols P3 and P4, an untrusted host can also "grow a fake stem", i.e. append data to the chain in place of other hosts.

In [4], Karnik and Tripathi propose an *append only container* mechanism. Its goal is to protect a container of data in an object such that new data can be added to it but any subsequent modification of a data contained therein can be detected by the agent's originator. To achieve this goal Karnik and Tripathi propose to append to the container a cryptographic checksum that binds the current data to the previous ones.

This scheme aims at data integrity, but presents some limitations. First of all, as in the previous protocols, since the dynamic data of the agent are not bound to its static data, untrusted hosts can easily abuse other hosts as oracle.

Furthermore, since the dynamic data of the agent are not bound to the identity of the next hosts, an untrusted host i_n cooperating with a second untrusted host i_j can easily truncate the chain of data at k for each $j \leq k < n$ without being detected. Finally, as acknowledged by the authors themselves, since the verification process requires the originator private key, it can be accomplished only by the agent's originator itself, i.e. modifications of the collected data are not publicly verifiable. Being each data signed using the data originator private key, *data not-repudiability* is achieved.

Another protocol proposed in the literature is the second version of the *multi-hops protocol* by Corradi et al. [2] [2]. Such protocol, as the previous ones, does not solve the problem of binding the static and dynamic parts of the agent. It is based on a simple idea: agents carry along a cryptographic proof of the data they have already gathered. This proof, called Message Integrity Code (MIC) serves as a chaining relation that binds data previously gathered by the agent to the ones obtained at the current host and to the identity of the next host to be visited. Thanks to this double chaining relation, *strong data forward integrity* and *data truncation resilience* are achieved (under the assumption that the problem of binding the static and dynamic parts of the agent is solved). Since all gathered data are signed using the data originator private key, *data non-repudiability* is accomplished too. *Data confidentiality* is not achieved but can be easily added. *Forward privacy* is not achieved.

[2]The first version of the multi-hop protocol [1] did not achieve data authenticity and for this reason had severe limitations.

In conclusion, the multi-hops protocol suffers from the same limitations of the Karioth et al.'s protocols: hosts can be abused as oracles and lack of strong truncation resilience.

A further protocol that is worth citing is described in [9]. As the append-only container mechanism, current data is bound only to the previously gathered data (and not with the identity of the next host). An attacker location mechanism is described too. However, to make this procedure possible, visited hosts must store a certain amount of information about all the visiting agents.

Recently, in [7], a new protocol has been described. As the Corradi et al's protocol, it achieves *data non-repudiability*, *strong data forward integrity* and *data truncation resilience*, but differently by all the other previous protocols it is robust against interleaving attacks described in [5].

5. THE PROTOCOL

Our proposal is to have a single abstract protocol that can be instantiated in several different ways. The abstract protocol defines in an abstract way the structure of the message sent from host i_n to host i_{n+1} when an agent moves from i_n to i_{n+1}. As we already said this message contains the agent static part Π and the protected area of the agent dynamic part.

According to the abstract protocol, on each host i_n the agent extends its protected area appending a component $\mathcal{M}_n$ that contains the data d_n provided by or computed at the host. $\mathcal{M}_n$ is composed of two parts, denoted $\mathcal{D}_n$ and $\mathcal{C}_n$.

These ones can take slightly different forms according to the required properties. Moreover, $\mathcal{D}_n$ is always a function $\mathcal{D}$ of i_n and d_n, i.e. $\mathcal{D}_n = \mathcal{D}(i_n, d_n)$. Similarly, $\mathcal{C}_n$ is always a function $\mathcal{C}$ of i_n, d_n, $\mathcal{C}_{n-1}$ and i_{n+1}, i.e. $\mathcal{C}_n = \mathcal{C}(i_n, d_n, \mathcal{C}_{n-1}, i_{n+1})$. Functions $\mathcal{D}$ and $\mathcal{C}$ are the configuration parameters of the protocol.

The agent static part Π is paired with a timestamp t, and the pair is signed by the initiator, so as to obtain a static piece of data $\Pi_0 = \{\Pi, t\}_{S^{-1}_{i_0}}$ which uniquely identifies each agent instance. Moreover, each visited host can verify the authenticity of the agent static part and the identity of the agent, thus for example being able to detect whether the same agent visits the host more than once.

From a formal point of view, the abstract protocol can be written as follows:

$$\forall n \in [0, k]$$

$$\begin{aligned}
\mathcal{M}_n &= \mathcal{D}_n \parallel \mathcal{C}_n \\
\mathcal{D}_n &= \mathcal{D}(i_n, d_n) \\
\mathcal{C}_n &= \mathcal{C}(i_n, d_n, \mathcal{C}_{n-1}, i_{n+1}) \\
\mathcal{C}_{-1} &= \epsilon \\
\Pi_0 &= \{\Pi, t\}_{S^{-1}_{i_0}} \\
i_n \rightarrow i_{n+1} &: \Pi_0, \{\mathcal{M}_0, ..., \mathcal{M}_n\}
\end{aligned}$$

It can be noted that, although in general it will be empty, d_0 has been included as well.

The encoding of protocol messages is not specified, but it is assumed that it is such that each host can uniquely extract a Π_0 and a set of $\mathcal{M}_i$ from a correctly encoded received message. Each host i_{n+1}, on reception of a message, must check that it is correctly encoded, and discard it if it does not conform to the encoding rules. If the message is correctly encoded, the host must extract Π_0 and the set $\{\mathcal{M}_0, ..., \mathcal{M}_n\}$ from it. Then, the host must extract the initiator identity from Π_0 and verify the signature before making the agent runnable. If any of such operations fails, the message must be discarded.

Instantiating a concrete protocol from the basic abstract protocol entails defining functions $\mathcal{D}$ and $\mathcal{C}$ and specifying any additional verification operations that must be performed on the various hosts.

Let us start explaining how a concrete protocol achieving *data authenticity* can be obtained. The easiest way to do it is to define $\mathcal{M}_n$ in such a way that the originator can determine for sure the host where $\mathcal{M}_n$ was generated. Assuming that the originator i_0 can always verify i_n's signatures, this can be obtained by signing data d_n with i_n's private key. More precisely, d_n is signed together with the agent static part Π_0. This is needed to bind d_n to the unique agent identity, thus making sure that d_n is really the data produced by the agent instance identified by Π_0 on host i_n. In practice, by signing Π_0 and d_n together, i_n certifies that the agent described by Π_0 has really been executed on i_n producing data d_n. This avoids attacks as the ones described in [5] where malicious hosts substitute the agent code in order to collect signed data which are then attached to the original agent code. Of course, a malicious host can always add a d_n which has not been computed while executing the agent, but our concern is only data integrity, not data correctness.

Hence, the protocol (MS1) which achieves data authenticity can be formally defined choosing the parameters as follows:

$$\begin{aligned}
\mathcal{D}(i_n, d_n) &= \begin{cases} \epsilon & \text{if } i_n = i_0 \\ d_n & \text{otherwise} \end{cases} \\
\mathcal{C}(i_n, d_n, \mathcal{C}_{n-1}, i_{n+1}) &= \begin{cases} \epsilon & \text{if } i_n = i_0 \\ \mathcal{S}^{-1}_{i_n}(d_n, \Pi_0) & \text{otherwise} \end{cases}
\end{aligned}$$

On intermediate hosts, no other verification operation must be executed, apart from the ones specified for the abstract protocol. The initiator, instead, upon receiving back the agent must also verify all the signatures and discard the agent if any verification fails.

It is worth noting that this protocol formalizes the basic mechanism used in Yee's per-server digital signature protocol [10], and satisfies the same properties. In particular, since no one, except the data originator itself, can generate a valid $\mathcal{S}^{-1}_{i_n}(d_n, \Pi_0)$, for data d_n, *data authenticity* is achieved.

It can be noted that this protocol satisfies *data non-repudiability* too, because, after having received back the agent, the initiator owns a proof, namely $\mathcal{S}^{-1}_{i_n}(d_n, \Pi_0)$, of the fact that the agent identified by Π_0 really visited host i_n and, while visiting it, d_n was generated.

This protocol can be used in several realistic situations, where data integrity is not needed. For example, this protocol is useful when agents are used in non-competitive environments or when the nature of the gathered data makes untrusted hosts not interested in deleting information gathered on other hosts. This may hold for example if a network management agent visits all the nodes of a network to ask them the number of users currently logged in. In this case, the agent sender may be only interested in data authenticity.

It is worth noting that, although in the most general case this protocol does not guarantee any data integrity property, it can provide *trusted data integrity* in the special case where the set of hosts to be visited is already known at agent departure and the value of d_n depends only on Π, i_0 and i_n.

Besides data integrity, there are some other protection aspects that might be useful, but are not provided by this protocol. For example, all the hosts can read the data gathered by the agent on previous hosts (lack of *data confidentiality*) and all the hosts can know the identity of the host where the already gathered data was generated (lack of *origin confidentiality*). However, the protocol can be extended at will, in order to achieve data confidentiality and/or forward privacy.

To achieve data confidentiality, it is enough to encrypt d_n using the initiator public key K_{i_0} as shown below:

$$\mathcal{D}(i_n, d_n) = \begin{cases} \epsilon & \text{if } i_n = i_0 \\ \{d_n\}_{K_{i_0}} & \text{otherwise} \end{cases}$$

Of course, the initiator will subsequently decode it.

Alternatively, a shared secret key known only by i_n and i_0 can be used if available.

In fact, with one of such encryptions, if the encryption scheme is secure, no intermediate host can extract d_n from the agent protected area and so data confidentiality is achieved.

To achieve origin confidentiality, instead, it is necessary to hide the identity of the hosts where data was generated. This can be achieved, for example, by encrypting the host signature with the agent initiator public key:

$$\mathcal{C}(i_n, d_n, \mathcal{C}_{n-1}, i_{n+1}) = \begin{cases} \epsilon & \text{if } i_n = i_0 \\ \{\mathcal{S}_{i_n}^{-1}(d_n, \Pi_0)\}_{K_{i_0}} & \text{otherwise} \end{cases}$$

and having it decrypted on the initiator.

If the encryption scheme is secure, only the agent initiator i_0 can decrypt the sealed signature and so get the identity of the host where the data was generated.

It is worth noting that, although such a measure makes it impossible for intermediate hosts to get data origin information directly from the agent itself, it is not enough to get origin confidentiality globally, because such information can be obtained otherwise. First of all, it is always possible for a network intruder to get the identity of the host i_n where data d_n was generated by analyzing the agent state just before and just after the agent visited i_n. To avoid this kind of information leakage, confidential channels can be used to send agents. However, even with confidential channels, host i_{n+1} can know the identity of its predecessor i_n and hence, if no precautions are taken, it can get the identity of the host where the last data item d_n was generated. This possible attack can be avoided by using anonymous connections [8], which make it impossible for the receiver to know the sender's identity.

It can be concluded that origin confidentiality is possible only provided that the network environment used implements adequate concealment techniques.

As we already said, under certain conditions, the above described concrete protocol achieves *trusted data integrity.* This one is a rather weak property which should be used with caution. For example, the above protocol does not prevent an untrusted host i_n from modifying its own data in the case it is visited by an agent more than once or to modify the data of a cooperating untrusted host i_j (with $j < n$) and, since no way to discover unstrusted hosts is provided, this can lead to unfair behaviors in the case untrusted hosts can read the data provided by other hosts (for example when we are dealing with shopping agents).

If the *sequence* (and not just the set) of the hosts to be visited by the agent is already known at agent departure, a stronger integrity property can be achieved by introducing a *chaining relation* [3] that binds d_n to the previously gathered data (actually to $\mathcal{C}_{n-1}$). This can be achieved by defining the $\mathcal{C}$ function as follows (MS2):

$$\mathcal{C}(i_n, d_n, \mathcal{C}_{n-1}, i_{n+1}) = \begin{cases} \epsilon & \text{if } i_n = i_0 \\ \mathcal{S}_{i_n}^{-1}(d_n, \Pi_0, \mathcal{C}_{n-1}) & \text{otherwise} \end{cases}$$

and the verification operations as in the previous protocol. In this way i_n will not be able to substitute data d_j gathered by the agent on host i_j (with $j < n-1$) without breaking the chaining relation even in the case i_j conspires with i_n. On the basis of the above reasoning we can conclude that, if the agent itinerary is known at agent departure, the above mechanism achieves *strong data forward integrity* too. Actually, i_n can substitute d_j (with $j < n$) if all the hosts $i_j, i_{j+1}, \ldots, i_{n-1}$ conspire with i_n.

Although from a theoretical point of view the protocol does not provide any form of truncation resilience, from a practical point of view it can be successfully used also in the cases where stronger forms of data integrity are required, if it is assumed that the agent must not visit an intermediate host more than once and each host records the identities of the agents it has already hosted. In fact, let us assume for example that an untrusted host i_n substitutes data d_j provided by a cooperating host i_j (with $j < n$) by removing all the data gathered by the agent on the hosts following i_j in the agent itinerary and by making the agent restart from host i_j. Being Π_0 unique, any trusted host i_l (with $j < l < n$) could notice the fact it had already hosted the agent and would discard it. However, it is worth noting that, also with such assumptions, strong data integrity is not fully achieved, because, as we have already observed, untrusted host i_n can substitute d_j (with $j < n$) in the case all the hosts $i_j, i_{j+1}, \ldots, i_{n-1}$ conspire with it.

Clearly, even if the obtained results are quite interesting, they cannot be applied in the case we are dealing with free-roaming agents. In fact they strictly depend on the fact that the agent itinerary is known at agent departure.

To achieve similar properties in this case too, $\mathcal{C}$ can be modified so as to introduce a second *chaining relation*, binding d_n to the identity of the next host i_{n+1}. The resulting concrete protocol (MS3) corresponds to the Karjoth et al.'s publicly verifiable chained digital signature protocol [3] is defined by:

$$\mathcal{C}(i_n, d_n, \mathcal{C}_{n-1}, i_{n+1}) = \begin{cases} \mathcal{S}_{i_0}^{-1}(\Pi_0, i_1) & \text{if } i_n = i_0 \\ \mathcal{S}_{i_n}^{-1}(d_n, \Pi_0, \mathcal{C}_{n-1}, i_{n+1}) & \text{otherwise} \end{cases}$$

Like the previous concrete protocol we presented, this one *achieves data authenticity*, *data non-repudiability* and *strong data forward integrity.*

Nevertheless, like the similar Karjoth et al.'s protocol, it does not achieve any form of truncation resilience, which, as we already observed, is one of the main limitations of the current mobile agent data integrity mechanisms. In fact, as far as we know, no proposed protocol provides a way for detecting truncation at j of a chain of collected data in the case i_j colludes with the attacker. In fact, if host i_j sends to the attacker the state of the agent as it reaches i_j, the attacker will always be able to restore the saved agent state and so truncate the agent data at i_j.

In the case the data gathered by the agents on each host i_n only depends on Π, i_0, i_n and the current contents of the agent protected area, a satisfactory, even though not definitive, solution to this problem can be achieved by forcing the agent to securely store the addresses of the next hosts to be visited as long as they are computed or become available. In practice, on each visited host i_n the agent gathers two pieces of data: a set p_n of hosts to be visited (eventually empty in case the visited host does not want to add new hosts to the agent's itinerary) and the actual data d_n (eventually a dummy data in case i_n does not want to or is not able to provide a data).

Both p_n and d_n are stored by the agent in a secure way, using the last mechanism described (MS3). In this way, *strong data forward integrity* is achieved.

It is worth noting that saving the hosts to be visited in a list while they are computed does not mean that the agent is not a free-roaming one. In fact, each time the agent can choose the next host to be visited from the list and can also compute and add new hosts to the list. The only thing the agent cannot do is deleting hosts from the list. In practice, if the agent has decided that it will eventually visit an host, it cannot then change its mind and decide not to visit that host.

From a formal point of view, the proposed protocol (MS4) can be obtained instantiating the abstract protocol in the following way:

$$\mathcal{D}(i_n, < d_n, p_n >) \quad = \quad \begin{cases} p_0 & \text{if } i_n = i_0 \\ < d_n, p_n > & \text{otherwise} \end{cases}$$

$$\mathcal{C}(i_n, < d_n, p_n >, \mathcal{C}_{n-1}, i_{n+1}) \quad =$$

$$\begin{cases} \mathcal{S}_{i_0}^{-1}(p_0, \Pi_0, i_1) & \text{if } i_n = i_0 \\ \mathcal{S}_{i_n}^{-1}(< d_n, p_n >, \Pi_0, \mathcal{C}_{n-1}, i_{n+1}) & \text{otherwise} \end{cases}$$

The agent chooses i_{n+1} from the set

$$(\bigcup_{j=0}^{n} p_j) \ \setminus \{i_0, \ldots, i_n\}$$

If the set is empty, then $i_{n+1} = i_0$. The result of the choice must deterministically depend on Π, i_0, i_n and the current contents of the agent protected area.

All the hosts in p_0 will be visited by the agent. Since protocol MS3 achieves strong data forward integrity, it can be observed that all the hosts added to the agent's itinerary by the trusted hosts belonging to p_0 will be visited as well. Going on in a recursive way, we can guess that all the hosts added to the agent's itinerary by a trusted host that has been added through a chain of trusted hosts will be visited too.

Reasoning about the previous statement we can understand how this protocol solves in a satisfactory way the truncation problem. In fact, only truncations of the data gathered by the agent on hosts directly or indirectly added to the agent itinerary by an untrusted host cannot be detected.

Then, it can be concluded that this protocol achieves a slightly weaker form of trusted data integrity, that we call *weak trusted data integrity*, even with free-roaming agents: what is guaranteed is integrity of any data collected on the trusted hosts that have been directly or indirectly added to the agent itinerary through trusted hosts.

It is worth pointing out that this solution requires that the agent cannot return home after visiting each of the hosts added to the set in its protected area.

To add *data confidentiality* to the above protocol, $\mathcal{D}$ must be redefined in order to encrypt d_n using the public key of the agent's originator i_0 (or, if available, a shared secret key known only by i_0 and i_n):

$$\mathcal{D}(i_n, < d_n, p_n >) \quad = \quad \begin{cases} p_0 & \text{if } i_n = i_0 \\ < \{d_n\}_{K_{i_0}}, p_n > & \text{otherwise} \end{cases}$$

In this way, if the encryption scheme is secure no one except i_0 can extract d_n from the agent protected area.

The various configuration choices presented in this section make it possible to select a protocol instance according to the needed properties and according to the agent and environment features.

It can be noted that confidentiality properties can be selected independently of integrity properties, and, for what concerns data integrity, several protection levels are possible.

The strongest integrity property that can be achieved with the proposed mechanisms depends on the agent and environment features, but in most cases *trusted data integrity* can be achieved.

6. CONCLUSIONS

Previous work on mobile agent data protection presented some interesting ideas about cryptographic mechanisms aimed at guaranteeing properties such as data integrity and data confidentiality. However, such proposals had several limitations and were formalized only partially.

In this paper, a twofold contribution has been presented. On the one hand, it has been shown how the underlying ideas of some previous proposals can be improved, so as to reduce their limitations. In particular, it has been shown how an acceptable form of truncation resilience can be achieved. On the other hand, the resulting protection mechanisms have been expressed formally, within the framework of a configurable protocol, where it is possible to select one of several protection levels, according to the required security properties, and according to the particular assumptions that can be done about the agent and environment features.

In this paper we have presented some of the possible configuration choices for the proposed configuration protocol, which cover many realistic situations and increase the extent to which a mobile agent can be protected, with respect to similar previous solutions.

However, some questions are left for further research. Specifically, a more systematic exploration of the possible configuration choices for the abstract protocol presented here is needed. An important open problem remains the implementation of even stronger truncation resilience properties.

7. ACKNOWLEDGMENTS

This work has been (partially) funded by the Center of Excellence on Multimedia Radiocommunications (CERCOM) of Politecnico di Torino and by the Italian National Research Council (grant number CNRC00FE45_002).

8. REFERENCES

[1] A. Corradi, R. Montanari, and C. Stefanelli. Mobile agents integrity in E-commerce applications. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems Workshop (ICDCS'99)*, pages 59 – 64, Austin, Texas, May 31 – June 5 1999. IEEE Computer Society Press.

[2] A. Corradi, R. Montanari, and C. Stefanelli. Mobile agents protection in the Internet environment. In *Proceedings of the 23th Annual International Computer Software and Applications Conference (COMPSAC'99)*, pages 80 – 85, Phoenix, Arizona, October 25 – 26 1999. IEEE Computer Society Press.

[3] G. Karjoth, N. Asokan, and C. Gülcü. Protecting the computation results of free-roaming agents. In K. Rothermel and F. Hohl, editors, *Proceedings of the 2nd International Workshop on Mobile Agents*, volume 1477 of *Lecture Notes in Computer Science*, pages 195–207. Springer-Verlag: Heidelberg, Germany, 1998.

[4] N. M. Karnik and A. R. Tripathi. A security architecture for mobile agents in Ajanta. In *Proceedings of 20th International Conference on Distributed Computing Systems*, pages 402–409. IEEE Computer Society Press, 2000.

[5] V. Roth. On the robustness of some cryptographic protocols for mobile agent protection. In G. P. Picco, editor, *Proceedings of the 5th International Conference on Mobile Agents (MA 2001)*, volume 2240 of *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag: Heidelberg, Germany, 2001.

[6] V. Roth. Programming Satan's agents. In *Proceedings of the 1st International Workshop on Secure Mobile Multi-Agent Systems*, Montreal, Canada, 2001.

[7] V. Roth. Empowering mobile software agents. In N. Suri, editor, *Proc. 6th IEEE Mobile Agents Conference*, volume 2535 of *Lecture Notes in Computer Science*, pages 47–63. Spinger Verlag, October 2002. ISBN 3-540-0085-2.

[8] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous connections and onion routing. In *IEEE Symposium on Security and Privacy*, pages 44–54, Oakland, California, 4–7 1997.

[9] X. F. Wang, X. Yi, K. Y. Lam, and E. Okamoto. Secure information gathering agent for internet trading. In C. Zhang and D. Lukose, editors, *Proceedings of the 4th Australian Workshop on Distributed Artificial Intelligence on Multi-Agent Systems : Theories, Languages, and Applications (DAI-98)*, volume 1544 of *Lecture Notes in Computer Science*, pages 183–193. Springer-Verlag: Heidelberg, Germany, July 13–13 1998.

[10] B. S. Yee. A sanctuary for mobile agents. In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, volume 1603 of *Lecture Notes in Computer Science*, pages 261–273. Springer-Verlag, Berlin Germany, 1999.