Experiments on Formal Verification of Mobile Agent Data Integrity Properties

Paolo Maggi and Riccardo Sisto
Dip. di Automatica e Informatica - Politecnico di Torino
Corso Duca degli Abruzzi 24, I-10129 Torino, ITALY
email:maggi@athena.polito.it, sisto@polito.it

***Abstract*— This paper explores the possibility of applying existing verification techniques and tools to the cryptographic mechanisms specifically designed for the protection of mobile agents from their environment, with a particular emphasis on agent data integrity. In a previous paper we explored the use of the CSP-based tools FDR and Casper. Here, instead, we present our experience with a prototype of a new verification tool based on spi-calculus. The symbolic techniques employed in this new tool make it efficient enough to analyze protocol configurations larger than the ones that could be analyzed with the CSP-based tools. This made it possible to formally find attacks that did not show up in the simplified configurations previously analyzed.**

I. 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 (MA) paradigm really acceptable, it is necessary to face all the numerous security threats that arise from it [1], [2]. 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.

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 diffcult 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, instead of preventing malicious hosts from tampering with the collected data, provide a way for detecting tampering attempts (e.g. [3], [4], [5], [6], [7]).

In general, all the proposed solutions are based on some kind of cryptographic protocol which, in turn, uses basic cryptographic operations such as encryption and digital signatures. Despite their apparent simplicity, cryptographic protocols have revealed themselves to be very error prone, especially because of the difficulty generally found in foreseeing all the possible attacks and all the possible behaviors of various parallel protocol sessions. For this reason, a lot of attention is being paid to formal methods which can help in developing error-free protocols or in analyzing the vulnerability of existing protocols. Up to now, such methods have been successfully employed to formally verify security properties of classical message-based protocols, such as authentication protocols (eg. [8], [9]), whereas only a few attempts to use them to analyze the security of agent-based systems have been documented [10], [11]. It is worth noting that, as explained in [10], the specification and verification of security issues related to mobile agent systems involves new aspects not encountered in classical cryptographic protocols, such as for example the need to model the fact that the behavior of an agent potentially becomes unpredictable each time it visits an untrusted host.

This paper explores the possibility of applying existing verification techniques to the verification of the cryptographic mechanisms specifically designed for the protection of agents from their environment, with a particular emphasis on agent data integrity.

While in [10] we showed how to verify mobile agent data integrity properties using the CSP-based tools Casper [12] and FDR [13], in this paper we extend the work presented in [10] and [11] describing a new and really promising approach based on the use of a spi-calculus model checker recently developed at Politecnico di Torino and based on symbolic data representations [14].

The rest of the paper is organized as follows. In section II we present the modeling approach in general terms. Section III reports a brief introduction on spi-calculus. In section IV, we present a sample mobile agent cryptographic protocol that will be used to illustrate our modeling approach and its potential. In section V, we show how to write a formal model of an instance of the sample protocol using spi-calculus and how to define data integrity properties. In section VI we present some verification results, obtained using the spi-calculus model checker developed at Politecnico di Torino. Section VII concludes.

II. Modeling a mobile agent system

Formal models of cryptographic protocols typically follow the Dolev-Yao [15] approach. They are composed of a set of

principals which send messages to each other according to the protocol rules, and an intruder, representing the activity of possible attackers. The intruder can perform any kind of attack: it can not only overhear all the transmitted messages, learning their contents, but it can also intercept messages and send new messages created using all the items it has already learned, as well as new nonces. So the intruder can fake messages and sessions.

Since such models are meant to reveal possible security flaws in the protocols and not flaws in the cryptosystems used by the protocols, cryptography is modeled in a very abstract way and it is assumed to be "perfect". This means that:

- the only way to decrypt an encrypted message is to know the corresponding key;
- an encrypted message does not reveal the key that was used to encrypt it;
- there is sufficient redundancy in messages so that the decryption algorithm can detect whether a ciphertext was encrypted with the expected key.
- the attacker cannot guess and/or forge any secret protocol data.

Although such assumptions are obviously not completely true for real cryptosystems, they represent the properties of an ideal cryptosystem, so they are useful to isolate the flaws of the protocol itself. In other words, any flaw found with this model is a real protocol flaw, but it is possible that the model does not reveal other weaknesses due to the used cryptosystems.

Mobile agents are programs that can migrate from one network host to another one while executing. They are executed by agent interpreters that run on each host and communicate by message passing. An agent migrating from one host to another host consists of a static part, typically including the agent code and, possibly, some static data, and a dynamic part, including all the agent elements that can change over time (program counter, stack, variables, etc.).

In order to model a mobile agent system, we use a technique quite similar to the one used for normal cryptographic protocols, based on the same assumptions about perfect cryptography and intruders.

Agents are not modeled as autonomous mobile principals, but the whole agent-based system is represented at a lower level of abstraction, closer to the real system. Principals represent hosts which, by their agent execution platform, can execute mobile agent code. The migration of an agent from host to host is represented by a message, exchanged by the principals that represent the involved hosts, containing the agent code and data.

Since the integrity of the static agent code and the static agent data is a problem shared by all mobile agents, it can be solved by the MA platform, independently on the particular functionality of the agent. Since we are not interested in validating this part of the protocol, we assume that this problem is already solved in a reliable way and we do not model code transmission explicitly, but we simply assume that trusted hosts always execute the right code. So each agent hop is represented by a message containing only the dynamic part of the agent data.

The main new aspect in mobile agent cryptographic protocols with respect to classical authentication protocols is the possibility of having attacks due both to network intruders and to untrusted hosts that may alter the behavior of agents hosted by their execution platform in an unpredictable way.

Modeling agents by messages exchanged by the hosts helps us in taking all such issues into account. Let us assume that we have a single untrusted host H_u. Attacks due to H_u can be taken into account in the above model giving the intruder the possibility of totally replacing it. To obtain this, it is enough to give the intruder all the secrets known by H_u, for example H_u's private key, and remove the process running on H_u from the model. Knowing all H_u's secrets, the intruder can totally replace H_u, i.e. it can intercept any message directed to H_u, decrypt it exactly as H_u could do and forge any message H_u could produce in response to it. In other words, the intruder incorporates the behaviors of all possible network intruders as well as those of all possible untrusted hosts. This models any kind of malicious behavior of H_u, including any modification in the execution of the mobile agent on H_u, as well as the case in which the agent is sent by H_u to a host different from the one where it should go.

This approach can be easily extended to model environments with several untrusted hosts: it is enough to insert all their secrets in the initial intruder knowledge and remove the processes running on them from the model. This corresponds to modeling several untrusted hosts that can cooperate. The intruder process knows the secrets of all the untrusted hosts and so can replace each of them and use the total knowledge of all of them.

Modeling untrusted hosts that can cooperate is adequate for most applications. Nonetheless, a solution can be found even for cases in which it could be useful to model untrusted hosts that are unable to cooperate. These cases are difficult to model using the modeling approach explained above, because the specifier cannot control how the intruder is modeled, the only thing that can be specified about the intruder being its initial knowledge. However, the model with cooperating untrusted hosts includes, as a special case, the one with uncooperating hosts. Indeed, by analyzing the first model, we can find out all possible attacks against the protocol, including those that do not require any untrusted host cooperation. So a way out of this problem is to analyze all the attacks reported by the analysis tool and then filter out the ones involving cooperating untrusted host.

For what concerns the specification of data integrity properties, they can be expressed in the same way authenticity properties are specified in classical authentication protocols. Indeed, requiring that some data that is considered valid at a given site has actually been delivered by the expected one is really an authenticity property.

III. SPI-CALCULUS

Spi-calculus [16] is an extension of π-calculus [17], which adds a few cryptographic primitives to π-calculus, namely hashing and shared, private and public key encryption. Like π-calculus, it is a first order process algebra, so it does not address mobility explicitly. Its syntax and semantics can be found in [16]. We give here only a brief overview of the constructs used in this paper.

In π-calculus, a system is described as a set of concurrent communicating processes. The basic computational step and

synchronization mechanism is interaction, in which a term M is communicated from an output process to an input process via a named channel m. A term can be basically a name (representing a data or a channel), a variable or a pair (N, L), where N and L are in turn terms. An output process $\overline{m}(M).P$ is ready to output M on channel m. If an interaction occurs, term M is communicated on m and then process P runs. An input process $m(x).Q$ is ready to input on channel m. If an interaction occurs in which term M is communicated on m, then process $Q[M/x]$ runs, where the post-fix operator $[M/x]$ indicates literal substitution of variable x by term M.

A composition $P|Q$ behaves as processes P and Q running in parallel. Each of them can interact with the other one on channels known to both, or with the outside world independently of the other one.

A restriction $(\nu n)\ P$ is a process that makes a new, private name n and then behaves as P. This operator is used to specify data that is not known to the intruder.

A pair splitting process $let\ (x, y) = M\ in\ P$ behaves as $P[N/x][L/y]$ if term M is the pair (N, L). Otherwise the process is stuck.

A match process $[M\ is\ N]P$ behaves as described by P if terms M and N are the same, and otherwise is stuck.

New kinds of terms have been introduced in spi-calculus to represent cryptographic operations. Term $\{[M]\}_{H^+}$ is the encryption of term M with the public key of H, denoted H^+. A term of this form can be decrypted by using the private key of H, H^-, by the process $case\ y\ of \{[x]\}_{H^-}\ in\ P$, which behaves as $P[M/x]$ if y is $\{[M]\}_{H^+}$ and is stuck otherwise. Term $hash(M)$ is the *hashing* of M. The hashing function is assumed to be perfect and non-invertible. The notation $hash^n(x)$, which is not part of spi calculus, is used in this paper to denote that the $hash$ function is applied n times on x.

The description of spi calculus in [16] introduces testing equivalence ($\simeq$), which is a kind of observational equivalence that can be used to express security properties. If a process P is ready to synchronize immediately on channel β, we say that P exhibits barb β, and we write $P \downarrow \beta$. If P can exhibit β, immediately or after a few internal reactions, we write $P \Downarrow \beta$. A test is a pair (R, β), where R is a testing process and β is a barb. A test (R, β) is passed by process P if $(P|R) \Downarrow \beta$. Testing equivalence $\simeq$ is formally defined by:

$$P \simeq Q \ \Leftrightarrow\ P \sqsubseteq Q\ and\ Q \sqsubseteq P$$

where the preorder $\sqsubseteq$ is defined as

$$P \sqsubseteq Q \ \Leftrightarrow \forall (R, \beta)\ (P|R) \Downarrow \beta \Rightarrow (Q|R) \Downarrow \beta$$

In other words, $P \sqsubseteq Q$ iff all the tests passed by P are passed also by Q. As a consequence, two processes are testing equivalent iff they pass the same tests, i.e. they are indistinguishable by testing.

IV. A SAMPLE PROTOCOL FOR MOBILE AGENTS DATA INTEGRITY

In this section, we present a simple protocol which mains at the integrity of a data gathering mobile agent that runs on several possibly untrusted hosts..

This agent visits several hosts and simply picks up pieces of data on its way. For example, the agent could be a shopping agent dispatched to visit different companies and find out the prices at which they sell a given product, so as to select the company that offers the cheapest one. Data integrity of such an agent means that a host cannot tamper with the data already collected without being detected. This is a classical problem for which different protocols have been proposed [3], [4], [5], [6], [7]. The specific protocol we consider here was proposed in [6] by Corradi et al.. The idea is that agents carry along a cryptographic proof of the data they have already gathered. This proof prevents hosts from tampering with the data already collected without being detected.

For the description of the protocol we use the following notation. $hash()$ is a cryptographic hash function, i.e. a function which, theoretically, cannot be inverted (x cannot be deduced from $hash(x)$). Following the spi calculus notation, the private and public part of the key of host H are denoted H^- and H^+ respectively. Encryption of data x by private (public) key of host H is denoted $\{[x]\}_{H^-}$ ($\{[x]\}_{H^+}$).

We also take some notation from [6], where MIC stands for "Message Integrity Code" and is the cryptographic proof we have just mentioned, C is a "cryptographic counter", which is incremented by successive applications of $hash$ by the agent and AD is the list of already collected data. The hosts where data have to be collected are decided by the agent dynamically, in such a way that each host is visited at most once. The hosts are denoted, in order of visit by the agent, H_0, which is the initiator, and H_i $(1 \leq i \leq n)$, which are the hosts where data must be collected. The initiator initially creates the agent, sends it out and, at the end of the computation, receives it with the collected data. Each host H_i $(1 \leq i \leq n)$ has a piece of data D_i that will be collected by the agent.

AD_i and MIC_i are respectively the collected data and the MIC value after the agent has left H_i. Similarly, successive values taken by the cryptographic counter are denoted C_i. CID is the (static) code of the agent. It is signed by a trusted party for authentication and it is carried along from host to host with the agent. The agent moving from host H_{i-1} to host H_i can be represented by a message containing CID, AD_{i-1}, MIC_{i-1} and C_i.

A. Informal protocol description

- Initialization: H_0 generates a secret number C_0. It creates the agent and passes $C_1 = hash(C_0)$ to it.
- First Hop: The agent encrypts C_1 with H_1^+ to let it be accessible only on H_1, and then moves to H_1, carrying with itself only the encrypted C_1 (i.e. $\{[C_1]\}_{H_1^+}$). The collected data list AD_0 and the initial MIC MIC_0 are empty.
- On host H_1: After the agent has reached H_1, it asks H_1 to decrypt $\{[C_1]\}_{H_1^+}$, thus obtaining C_1 and collects D_1, so having $AD_1 = \{D_1\}$. Then it computes $MIC_1 = hash(D_1, C_1)$, and it increments the cryptographic counter by computing $C_2 = hash(C_1)$.
- Hop i ($2 \leq i \leq n$): The agent encrypts C_i with H_i^+ and then moves to host H_i carrying with itself the already collected data AD_{i-1}, the cryptographic proof MIC_{i-1}, and

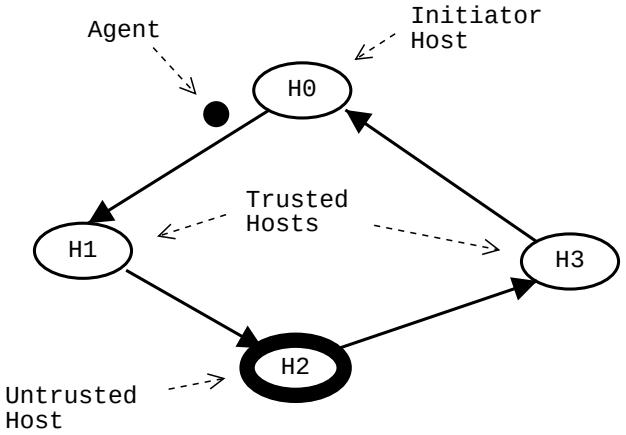

Fig. 1. Modeled instance of the Corradi et al.'s protocol.

the value of the cryptographic counter encrypted with H_i's public key: $\{[C_i]\}_{H_i^+}$.

- On host H_i ($2 \leq i \leq n$): After having decrypted $\{[C_i]\}_{H_i^+}$, the agent collects D_i and appends it to the collected data list, so computing $AD_i = AD_{i-1} \cup \{D_i\}$. It then computes a new proof by:

$$MIC_i = hash(D_i, C_i, MIC_{i-1})$$

and increments the cryptographic counter.
- Last hop: The agent encrypts C_n with H_0^+ and then moves from H_n back to H_0 carrying with itself the whole data AD_n, the computed checksum MIC_n, and $\{[hash(C_n)]\}_{H_0^+}$.
- Termination: H_0 receives from the agent AD_n, MIC_n, and $\{[hash(C_n)]\}_{H_0^+}$. From the values of C_0 and AD_n, H_0 can compute MIC_n, and check that it is the same that has just been received from the agent. If any difference is found, the agent data is considered to be invalid. This guarantees the integrity of validly collected data.

A host cannot modify the already collected data, basically because it cannot retrieve C from $hash(C)$. Since H_i does not know C_j $(j < i)$, it cannot modify D_j $(j < i)$ and so it cannot reconstruct a valid MIC.

As also noted in [6], this solution does not work if the agent visits a host to collect data twice, or if malicious hosts cooperate.

V. The sample protocol model

In this section we show how to model an instance of the Corradi et al. protocol reported in section IV using the spi-calculus.

For simplicity, we deal with a fairly small instance of the protocol. In this instance (Fig. 1), an agent is sent out from the initiator H_0, visits three hosts, identified as H_1, H_2 and H_3, and comes back to H_0. D_1, D_2 and D_3 are the data gathered by the agent on H_1, H_2 and H_3 respectively. Hosts H_0 H_1 and H_3 are trusted, whereas H_2 is not. In this way we can express the integrity property asserting that the data collected at H_1 and H_3 should not be modified. Of course, it is possible to extend the model to incorporate an arbitrary number of visited hosts. Moreover, we can have multiple sessions of the protocols, each one corresponding to a different agent.

As explained in section II, the model is composed by a set of processes, one for each trusted host and one representing the whole protocol instance.

Processes running on trusted hosts H_0, H_1 and H_3 are called respectively $Host_0$, $Host_1$ and $Host_3$. Both $Host_1$ and $Host_3$ take five parameters: d and k are respectively the data gathered by the agent on the host where the process is running and the host key (public and private); k_{next} is the public key of the next host to be visited; $Chan_{in}$ is the channel from which the agent has been received and $Chan_{out}$ is the channel used by the process to send the agent to the next host.

Process $Host_1$ is defined as given below:

$$\begin{aligned}
&Host_1(d, k, k_{next}, Chan_{in}, Chan_{out}) = \\
&Chan_{in}(y). \\
&case\ y\ of \{[C_1]\}_{k^-}\ in \\
&\overline{Chan_{out}}(d, hash(d, C_1), \{[hash(C_1)]\}_{k_{next}}). \\
&0
\end{aligned}$$

The process behavior is quite simple: first the message y is read from $Chan_{in}$ and decrypted using key k^- to obtain C_1; then message $(d, hash(d, C_1), \{[hash(C_1)]\}_{k_{next}})$ is sent out on channel $Chan_{out}$.

The description host $Host_3$ is:

$$\begin{aligned}
&Host_3(d, k, k_{next}, Chan_{in}, Chan_{out}) = \\
&Chan_{in}(x). \\
&let(x_1, x_2, MIC_2, y) = x\ in \\
&case\ y\ of \{[C_3]\}_{k^-}\ in \\
&\overline{Chan_{out}}(x_1, x_2, d, hash(d, C_3, MIC_2), \\
&\{[hash(C_3)]\}_{k_{next}}). \\
&0
\end{aligned}$$

Process $Host_0$ takes four parameters: k is H_0's key (public and private); k_{next} is the public key of the next host to be visited; $Chan_{in}$ is the channel from which the agent is received back at the end of its itinerary and $Chan_{out}$ is the channel used by $Host_0$ to send the agent to the first host of the agent itinerary.

The description of $Host_0$ (the originator host) is the following one:

$$\begin{aligned}
&Host_0(k, k_{next}, Chan_{in}, Chan_{out}) = \qquad (1) \\
&(\nu\ C_0) \\
&\overline{Chan_{out}}(\{[hash(C_0)]\}_{k_{next}}). \\
&Chan_{in}(x). \\
&let(x_1, x_2, x_3, MIC_3, y) = x\ in \\
&case\ y\ of \{[C_4]\}_{k^-} in \\
&[C_4\ is\ hash^4(C_0)] \\
&[MIC_3\ is\ hash(x_3, \\
&hash^3(C_0), hash(x_2, hash^2(C_0)),
\end{aligned}$$

$hash(x_1, hash(C_0)))]$
$\overline{Chan_{out}}(x_1, x_2, x_3).$
0

First of all, it creates secret C_0, encrypts it with k_{next} and sends it to the first host of the agent itinerary. The it waits for the agent coming back on channel $Chan_{in}$. Once received the agent back, it checks that the value of C_4 is equal to $hash^4(C_0)$ and that the received MIC_3 corresponds to the one it can compute starting from initial secret C_0 and data x_1, x_2 and x_3 gather by the agent on the three visited hosts.

If the computed MIC_3 matches, x_1, x_2 and x_3 are sent out on channel $Chan_{out}$. This last step has been added for testing the protocol (see testing equivalence, section III).

The whole protocol instance is:

$$
\begin{aligned}
&P() = \\
&(\nu\ K_0)(\nu\ K_1)(\nu\ D_1)(\nu\ K_3)(\nu\ D_3) \\
&(\overline{F}(K_0^+, K_1^+, K_3^+). \\
&(Host_0(K_0, K_1^+, c_{H_1}, c_{H_0})| \\
&Host_1(D_1, K_1, K_2^+, c_{H_1}, c_{H_2})| \\
&Host_3(D_3, K_3, K_0^+, c_{H_3}, c_{H_0})))
\end{aligned} \tag{2}
$$

where K_i is the key of host H_i and c_{H_i} is the channel connecting H_{i-1} to H_i (c_{H_0} connects H_3 to H_0). Note that K_2 is public, since H_2 is an untrusted host. Note also that we need to send public keys K_0^+, K_1^+ and K_3^+ on public channel F in order to let the intruder adds them to its initial knowledge.

A. Security Properties

We are interested in agent integrity, i.e. we want any tampering of D_1 and D_3 by H_2 or by network intruders to be detected.

Let us write the following specification of $Host_0$:

$$
\begin{aligned}
&Host_{0\,Spec}(k, k_{next}, Chan_{in}, Chan_{out}, d) = \\
&(\nu\ C_0) \\
&\overline{Chan_{out}}(\{[hash(C_0)]\}_{k_{next}}). \\
&Chan_{in}(x). \\
&let(x_1, x_2, x_3, MIC_3, y) = x\ in \\
&case\ y\ of\{[C_4]\}_{k^-}in \\
&[C_4\ is\ hash^4(C_0)] \\
&[MIC_3\ is\ hash(x_3, \\
&hash^3(C_0), hash(x_2, hash^2(C_0)), \\
&hash(x_1, hash(C_0)))] \\
&\overline{Chan_{out}}(d, x_2, x_3). \\
&0
\end{aligned} \tag{3}
$$

and the following whole protocol specification:

$$
\begin{aligned}
&P_{Spec}() = \\
&(\nu\ K_0)(\nu\ K_1)(\nu\ D_1)(\nu\ K_3)(\nu\ D_3) \\
&(\overline{F}(K_0^+, K_1^+, K_3^+). \\
&(Host_{0\,Spec}(K_0, K_1^+, c_{H_1}, c_{H_0}, D_1)| \\
&Host_1(D_1, K_1, K_2^+, c_{H_1}, c_{H_2})| \\
&Host_3(D_3, K_3, K_0^+, c_{H_3}, c_{H_0})))
\end{aligned} \tag{4}
$$

As the reader can note, equation (3) differs from equation (1) only in the last line. Process $Host_{0\,Spec}$ is a "magical" process that is ready to output the value of the paramter d on $Chan_{out}$ for any incoming agent that passes the validity tests. If we prove that specification (4) is testing equivalent to model (2), we have proved that the protocol ensures integrity of the data gathered by the agent on host H_1. Indeed, in that case, we are sure that the value of D_1 that H_0 has received has not been tampered with. Otherwise, the specification and the model would be willing to output different values through channel c_{H_0}, and they would not be testing equivalent.

So, the formal expression of integrity for the above agent is :

$$P_{Spec}() \simeq P() \tag{5}$$

The same has to be done for data D_3 gathered by the agent on host H_3.

Equation (5) is only an integrity specification. To specify additional properties other equations are needed.

VI. Checking the sample protocol

The model described in the previous section has been checked using a prototype of the new model checker for spi-calculus being developed at Politecnico di Torino. This tool builds a finite state transition system representing the joint behavior of a protocol model with an intruder that can interact with the protocol according to the Dolev-Yao approach. Since the intruder can in principle send an infinite number of different messages at each step, this state transition system is potentially infinite. However, as explained in [14], symbolic techniques are used to make it finite. The tool can also compare the protocol and specification transition systems, checking their testing equivalence. In case the equivalence does not hold, the tool gives a spi-calculus specification of an intruder that can exploit the difference.

As expected, we have been able to verify that, with respect to the analyzed instance, the Corradi et al. protocol is able to ensure the integrity of data D_1, but not of data D_3. In fact, the tool reported that $P()$ and $P_{Spec}()$ are not equivalent: the spi-calculus model of a process I representing one the possible attackers (intruders) as reported by the tool is the following one:

$$
\begin{aligned}
&I() = \\
&F(z_1).let(x_1, x_2, x_3) = z_1\ in \\
&C_{H_2}(z_2).let(x_4, x_5, x_6) = z_2\ in \\
&case\ x_6\ of\{[x_7]\}_{K_2^-}\ in \\
&(\nu\ y_1)(\nu\ y_2) \\
&\overline{C_{H_0}}(x_4, y_1, y_2, \\
&hash(y_2, hash(x_7), hash(y_1, x_7, x_5), \{[hash^2(x_7)]\}_{x_1}. \\
&0
\end{aligned}
$$

First of all the intruder reads message z_1 from public channel F adding K_0^+, K_1^+ and K_3^+ to its initial knowledge. Then it gets the agent from channel C_{H_2} and reads the agent's data as though it was H_2 (this is possible since it knows K_2^-). Then it creates two nonces y_1 and y_2 and sends the agent back to H_0, appending both y_1 and y_2 to the list of already gathered data and generating the corresponding MICs.

Note that this kind of attack is possible since the protocol does not require data authentication, so an untrusted host can add more than one data item without being detected.

The time needed to check the protocol security properties is about one minute on an AMD Athlon 850 Mhz CPU with 512 Mb of RAM.

It is worth noting how the results obtained with this new model checker are far better than the ones we obtained using the CSP-based tools Casper and FDR [10]. In that case, in fact, we were only able to analyze models where an agent visits up to three hosts. Moreover, even in such smaller test cases, on the same machine, the time needed to check the protocol security properties was far larger. This can be explained mainly by the fact that the CSP-based tools do not use symbolic techniques, but they put a limitation on the maximum length of the messages the intruder can send.

It is worth also noting that in [10], because of the limited size of the analyzed protocol instance, we failed to discover the attack we have reported in this paper.

VII. Conclusions

In this paper we have shown how a new model checker based on spi-calculus can be used to analyze the data integrity properties of a mobile agent cryptographic protocol. In a previous paper, we showed the analysis of the same protocol using the CSP-based tools Casper and FDR.

When using Casper and FDR, we only succeeded in analyzing models where an agent visits up to three hosts. The new tool based on spi calculus is more powerful than the previous one because it uses symbolic data reprersentations. Such a new tool enabled us to formally analyze reasonably sized models, and to find attacks on the protocol that did not occur in the simple models previously analyzed.

From the specification point of view, it can be noted that a clear formal specification as the one that has been presented in this paper is very important to unambiguously and precisely describe a mobile agent security mechanism and its properties. It expresses not only the contents of the exchanged messages, but all the aspects that are relevant to guarantee the security properties of interest, including the checks that must be done when messages are received. So a formal specification of this kind is a valid and practical basis for a correct implementation.

VIII. Acknowledgments

This work has been (partially) funded by the Center of Excellence on Multimedia Radiocommunications (CERCOM) of Politecnico di Torino.

References

[1] C. Tschudin. Mobile agent security. In M. Klusch, editor, *Intelligent Information Agents: Cooperative, Rational and Adaptive Information Gathering on the Internet*, Lecture Notes in Computer Science, pages 431–446. Springer-Verlag, Berlin, Germany, 1999.

[2] W.M. Farmer, J.D. Guttman, and V. Swarup. Security for mobile agents: Issues and requirements. In *Proceedings of the 19th National Information Systems Security Conference*, pages 591–597, Baltimore, Md., October 1996.

[3] B. S. Yee. A sanctuary for mobile agents. Technical Report CS97-537, UC San Diego, Department of Computer Science and Engineering, April 1997.

[4] G. Vigna. Cryptographic Traces for Mobile Agents. In Giovanni Vigna, editor, *Mobile Agent Security*, Lecture Notes in Computer Science No. 1419, pages 137–153. Springer-Verlag: Heidelberg, Germany, 1998.

[5] 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.

[6] A. Corradi, R. Montanari, and C. Stefanelli. Mobile agents integrity in E-commerce applications. In *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.

[7] X. F. Wang, X. Yi, K. Y. Lam, and E. Okamoto. Secure information gathering agent for internet trading. In Chengqi Zhang and Dickson 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 *LNAI*, pages 183–193, Berlin, Germany, July 1998. Springer.

[8] G. Lowe and B. Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 23(10):659–669, October 1997.

[9] S. Schneider. Verifying authentication protocols with CSP. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[10] Xavier Hannotin, Paolo Maggi, and Riccardo Sisto. Formal specification and verification of mobile agent data integrity properties: A case study. *Lecture Notes in Computer Science*, 2240:42–53, 2001.

[11] Xavier Hannotin, Paolo Maggi, and Riccardo Sisto. Using process algebras to formally specify mobile agent data integrity properties: A case study. In Andrea Omicini and Mirko Viroli, editors, *WOA 2001 – Dagli oggetti agli agenti: tendenze evolutive dei sistemi software*, Modena, Italy, 4–5 September 2001. Pitagora Editrice Bologna.

[12] G. Lowe. Casper: A compiler for the analysis of security protocols. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[13] Formal Systems (Europe) Ltd. Failures-Divergence Refinement. FDR2 User Manual. Available at http://www.formal.demon.co.uk/fdr2manual/index.html, 3 May 2000.

[14] L. Durante, R. Sisto, and A. Valenzano. A state exploration technique for spi-calculus testing equivalence verification. In *Proceedings of FORTE/PSTV 2000*, pages 155–170, Pisa, Italy, October 2000. Kluver.

[15] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, March 1983.

[16] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. Research Report 149, Digital Equipment Corporation Systems Research Center, January 1998. A shortened version of this report appeared in Information and Computation 148(1999):1-70.

[17] R. Milner, J. Parrow, and D. Walker. A calculus for mobile processes. Part I and II. *Information and Computation*, 100(1), 1992.