# USING ENTITY LOCATIONS FOR THE ANALYSIS OF AUTHENTICATION PROTOCOLS

RICCARDO FOCARDI
*Dipartimento di Matematica Applicata ed Informatica,*
*Università Ca' Foscari di Venezia,*
*via Torino 155, I - 30173 Venezia Mestre (Italy)*
*e_mail: focardi@dsi.unive.it* *

Entity authentication should allow the verification of an entity's claimed identity, by another entity. In the last years a number of definitions of entity authentication have been proposed (see, e.g., [1,2,3,4]). However they seem to be only good characterizations of authentication, i.e. they do not directly reveal the fact that the enemy is "breaking authentication". In this paper we propose a different approach where we try to directly model the idea of authentication: we make addresses of entities explicit and we basically reduce entity authentication to a mapping of addresses into entities. We require that even in the presence of an enemy this mapping is correct. We observe that this property can be very easily checked with an automatic tool and that it can be used to detect the classic attack to the Needham-Schroeder protocol.

## 1 Introduction

In the last years a number of formal definitions of entity authentication have been proposed. Some of them are similar to liveness, i.e., at the end of the protocol an entity $A$ can conclude that the other entity $B$ has participated in the protocol or at least wished to communicate with $A$ (see, e.g., [1,2,4]). Others, like *Agreement* [3], require some agreement on exchanged data.

These properties have been successfully applied to the verification of authentication protocols. However they seem to be good characterizations of entity authentication rather than formal definitions of it. Indeed, when an authentication protocol terminates correctly we should reasonably be able to conclude that each party thinks that the other one wished to (or did) communicate with it and/or that both agree on some exchanged data. On the other hand, in these models, an attack could be for example revealed by observing that one party is able to conclude the protocol without the participation of the other party. This indirectly implies that the intruder is substituting one of the two parties. So it seems that we do not have a "direct" modeling of what entity authentication is, e.g., when we have an attack we are able to observe

*This work has been partially supported by MURST project 9701248444_004 "Tecniche formali per la specifica, l'analisi, la verifica, la sintesi e la trasformazione di sistemi software".

this since something wrong is happening in the protocol execution. We though do not directly observe the fact that the enemy is "breaking authentication".

This kind of definition has risen up because of the intrinsic difficulties in finding a "definition of authentication", whatever this means.

In this paper we propose a different approach where we try to directly model the idea of authentication. What we do is to exploit off-line information, i.e., information that is invisible to the parties but is available before the execution to those who are checking the correctness of the protocol. This is not done in properties like the ones defined in [1,2,3,4].

The idea is to consider the locations of entities. We associate a location (e.g., the address on the net) to every entity. This represents the fact that when the protocol starts we have a particular situation on the net where entities are bound to their relative addresses. Moreover this represents our "off-line" information which is useful to model authentication. In fact we assume that a certain entity does not have any idea of the locations of all the other entities. At this point we say that "authenticate an entity" means "map an entity to its address". This is very intuitive and gives a direct definition of authentication. For example when we have a correct termination of an authentication protocol we see that we obtain only "correct mappings". Otherwise when we have an attack and the enemy is impersonating an entity $A$ we obtain the incorrect mapping of $A$ into the enemy's address.

We call this property *LocatedAuthentication*. It is very simple and can be easily checked with an automatic tool. We show that it can be used to detect interesting known flaws: for example, we show that it allows us to find the classic attack to the Needham-Schroeder protocol in a very simple and intuitive way. We formally compare *LocatedAuthentication* with the *Agreement* property, finding out that in some situations (i.e., when *Agreement* does not care about addresses) no one of them implies the other.

We use the Needham-Schroeder public-key protocol as a running example. In Section 2, we briefly present the basic protocol and the Lowe's fix. In Section 3.1 we show how to make addresses explicit in the specification of entity authentication protocols. In Section 3.2, we present a natural extension of the *Agreement* property [3] which exploits explicit addresses considering them as relevant data. Section 3.3 is about the *LocatedAuthentication* property. Finally, Section 4 contains some concluding remarks.

## 2 The Needham-Schroeder public key protocol

The aim of the Needham-Schroeder protocol [5] is to establish mutual authentication between two users *Alice* ($A$) and *Bob* ($B$). It uses public key cryp-

tography.

We will denote with $PK_U$ and $SK_U$ the public and secret keys for a generic user $U$, respectively. The protocol uses *nonces* which are random numbers generated with the purpose of being used in a *single* run of the protocol. In general, $N_A$ will represent a nonce generated by $A$.

The protocol is composed by seven steps, but four of them can be avoided if $A$ and $B$ have local caches of commonly used public key. In this way we obtain the "core" of the protocol:

$$\begin{array}{l} Message\ 1.\ A \rightarrow B : PK_B(N_A, A) \\ Message\ 2.\ B \rightarrow A : PK_A(N_A, N_B) \\ Message\ 3.\ A \rightarrow B : PK_B(N_B) \end{array}$$

Intuitively, *Message 1* represents the authentication request from $A$ to $B$. In this message $A$ sends to $B$ the nonce $N_A$ encrypted with $B$'s public key so that only $B$ can decrypt it. In *Message 2* $B$ replies to $A$, sending the new nonce $N_B$ and the one received from $A$, both encrypted with $A$'s public key. When $A$ receives the nonce $N_A$ back, she can conclude that is really talking with $B$ since only $B$ could have decrypted the first message. In the last message $A$ replies to $B$, sending back the $N_B$ nonce. Following the same reasoning as above, $B$ will conclude that is indeed talking with $A$.

The protocol presented above has been shown to be insecure [6]. To realize the cheat, two runs have to be active at the same time: one between *Alice* and *Evil* ($E$, the intruder), and one between *Evil* (that claims to be *Alice*) and *Bob*. Using $\alpha$ and $\beta$ to indicate the two runs and $E(A)$ to represent the intruder that claims to be $A$, the sequence that allows to cheat is:

$$\begin{array}{lll} Message\ \alpha 1.\ A & \rightarrow E & : PK_E(N_A, A) \\ Message\ \beta 1.\ E(A) & \rightarrow B & : PK_B(N_A, A) \\ Message\ \beta 2.\ B & \rightarrow E(A) & : PK_A(N_A, N_B) \\ Message\ \alpha 2.\ E & \rightarrow A & : PK_A(N_A, N_B) \\ Message\ \alpha 3.\ A & \rightarrow E & : PK_E(N_B) \\ Message\ \beta 3.\ E(A) & \rightarrow B & : PK_B(N_B) \end{array}$$

In message $\alpha 1$, $A$ sends her nonce and identifier to $E$, trying to establish a connection with him. When $E$ receives this message, he forwards it to $B$, pretending to be $A$ ($\beta 1$). $B$ accepts the message and sends back his nonce and the nonce just received ($\beta 2$). $E$ intercepts this message and, since he cannot decrypt it, he uses $A$ as an oracle forwarding to her the message ($\alpha 2$). $A$ accepts and decrypts it; then she sends to $E$ the last message containing $N_B$ ($\alpha 3$), and considers established the session with $E$. In this way $E$ has learned $B$'s nonce. Then $E$ can send it back to $B$ ($\beta 3$), that finally thinks to have established a session with $A$.

The problem is in message 2 of the protocol. Since there is no identifier inside it, anyone can intercept and forward it claiming to be anyone else.

In [6] it is also presented a corrected version of the protocol:

$$Message\ 1.\ A \rightarrow B : PK_B(N_A, A)$$
$$Message\ 2.\ B \rightarrow A : PK_A(N_A, N_B, B)$$
$$Message\ 3.\ A \rightarrow B : PK_B(N_B)$$

where message 2 now contains also the sender identifier. We will refer to this corrected version as the Lowe-Needham-Schroeder protocol.

## 3 Located Entity Authentication

In this section we introduce the idea of *located entity authentication*: we refine the specification model by making explicit the addresses of entities and we consider them as relevant for authentication. The section is composed of three subsections. In Section 3.1 we explain how to make addresses explicit in the specification of entity authentication protocols. In Section 3.2, we present a natural extension of the *Agreement* property [3] which exploits explicit addresses considering them as relevant data. In Section 3.3, we present another property which is based only on the fact that addresses are explicit and that is very simple and easy to check. We show that it can be used to detect the classic attack to the Needham-Schroeder protocol presented in Section 2 and we compare it with the *Agreement* based properties presented in Section 3.2.

### *3.1 Making Addresses Explicit*

Consider the attack to the Needham-Schroeder protocol presented in Section 2. Using the classic notions of authentication based on liveness [1,2,3,4] we detect it by observing that $B$ terminates the protocol convinced to communicate with $A$ but $A$ has never started a session of the protocol with $B$. By this observation we deduce that someone has impersonated $A$. Here we want to study a different approach to authentication based on entity locations. We will present a definition where entity authentication is reduced to the problem of mapping an entity to its (correct) location. As an example the attack we are considering here can be revealed by simply observing that $B$ is convinced to communicate with someone which should be $A$ but is not logged on $A$'s address.

In order to formalize this property we need to make explicit the address fields in the messages. We can think that at a certain instant (when the protocol is executed) we have a particular mapping of the entities into the

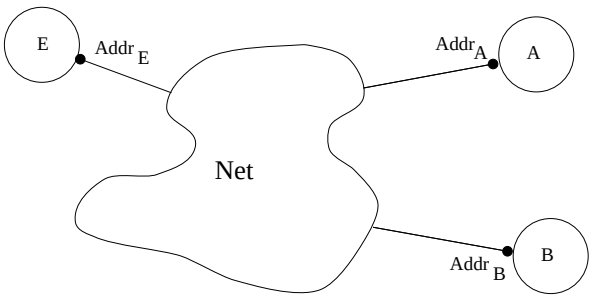


Figure 1. Mapping of entities to addresses

addresses. We call $Addr_E$ the current address for a generic entity $E$ (see Figure 1). We can write:

$$A \rightarrow B : Addr_A, Addr_B, PK_B(N_A, A)$$

In this way we make the address fields explicit. We have a more interesting situation for Message $\beta 1$ where $B$ is sending messages to $E$ thinking that it is $A$. This becomes:

$$B \rightarrow E(A) : Addr_B, Addr_E, PK_A(N_A, N_B)$$

So, it might be possible that a message addressed to a certain entity (e.g., $A$) is sent to a wrong address ($Addr_E$ in this case). In the example above, $B$ has no way of relating $A$ with $Addr_A$ at the beginning of the protocol. We can say that $E(A)$ represents entity $E$ (connected at $Addr_E$) behaving like or pretending to be $A$. Note that part of the message above is encrypted with $A$'s public key since it was addressed to $A$. Similarly for Message $\beta 1$ we have:

$$E(A) \rightarrow B : Addr_E, Addr_B, PK_B(N_A, A)$$

Note that we are not adding new information to the messages. We are simply trying to make explicit some data that are implicit in the notation $A \rightarrow B : Msg$. We can re-write the attack as follows:

$$\begin{array}{lll}
Message\ \alpha 1.\ A & \rightarrow E & : Addr_A, Addr_E, PK_E(N_A, A) \\
Message\ \beta 1.\ E(A) & \rightarrow B & : Addr_E, Addr_B, PK_B(N_A, A) \\
Message\ \beta 2.\ B & \rightarrow E(A) & : Addr_B, Addr_E, PK_A(N_A, N_B) \\
Message\ \alpha 2.\ E & \rightarrow A & : Addr_E, Addr_A, PK_A(N_A, N_B) \\
Message\ \alpha 3.\ A & \rightarrow E & : Addr_A, Addr_E, PK_E(N_B) \\
Message\ \beta 3.\ E(A) & \rightarrow B & : Addr_E, Addr_B, PK_B(N_B)
\end{array}$$

Now, in the last message we can observe that when $B$ conclude the protocol convinced to communicate with $A$ is indeed communicating with an entity logged on $Addr_E$. In the following we will see a formal definition which exploit this point.

### *3.2 Strong Agreement*

In this section we present an extension of the *Agreement* property [3] which exploits explicit addresses by considering them as relevant data. Intuitively, when a protocol satisfies this property we have that at the end of the execution the two parties agree on their respective addresses. In this way it is not possible that a session started by $A$ with an entity that pretends to be $B$, terminates correctly. This is because such an entity is not connected to the net on $B$'s address, and at the end of the execution $A$ and $B$ (wherever he is) must agree on their addresses. We first report from [3] an informal definition of *Agreement*:

> "A protocol guarantees to an initiator $A$ *Agreement* with a responder $B$ on a set of data items $ds$ if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder $B$, then $B$ has previously been running the protocol, apparently with $A$, and $B$ was acting as responder in his run, and the two agents agreed on the data values corresponding to all the variables in $ds$, and each such run of $A$ corresponds to a unique run of $B$".

Using a simplification of the notation used in [3], we call this property $Agreement(res, ini, ds)$. Basically it authenticates the responder to the initiator. In order to have mutual authentication we need also the symmetric property $Agreement(ini, res, ds)$. These two properties have been verified on the Lowe-Needham-Schroeder protocol considering $N_A$ and $N_B$ as relevant data (the data items in $ds$) [3]. As we said above, we add the addresses in the set of relevant data. For example, $Agreement(ini, res, \{n_ini, n_res\})$ (one of the property verified on the Lowe-Needham-Schroeder protocol) becomes $Agreement(ini, res, \{addr_ini, addr_res, n_ini, n_res\})$, where variables $addr_ini$, $addr_res$, $n_ini$, $n_res$ represent the addresses and the nonces of the initiator and the responder, respectively. In this way we obtain that at the end of the protocol $A$ and $B$ also agree on respective addresses.

In [3] the *Agreement* property is defined and automatically checked using the CSP language [7] and the FDR tool [8]. What is technically done in the *Agreement* property is to specify the two parties and the enemy in CSP, then we add for each party an action representing the running of the protocol and

another one representing the completion of it. These actions contain the data that the two parties have to agree on. For example in the execution we are analyzing, we have an action of the form:

$$signal_running_ini(A, B, \{Addr_A, Addr_E, N_A, N_B\})$$

for $A$ running the protocol as initiator with $E$ thinking it is $B$. The set of relevant data $ds$ is included in brackets. It is composed of the addresses and the nonces for the initiator and the responder, respectively. We also have:

$$signal_commit_res(B, A, \{Addr_E, Addr_B, N_A, N_B\})$$

for $B$ as a responder, at the end of the protocol. $B$ is convinced to communicate with $A$ on address $Addr_E$. We see that the data do not coincide. In particular, we do not have agreement on the addresses. [a] In order to check this, $Agreement(ini, res, \{addr_ini, addr_res, n_ini, n_res\})$ requires that when $B$ executes:

$$signal_commit_res(B, A, \{Addr_{ini}, Addr_{res}, N_{ini}, N_{res}\})$$

then $A$ has previously executed:

$$signal_running_ini(A, B, \{Addr_{ini}, Addr_{res}, N_{ini}, N_{res}\})$$

where $Addr_{ini}, Addr_{res}, N_{ini}$ and $N_{res}$ are possible values for the variables $addr_ini, addr_res, n_ini$ and $n_res$, respectively. This means that every time $B$ completes the protocol convinced that addresses and nonces are $\{Addr_{ini}, Addr_{res}, N_{ini}, N_{res}\}$, then $A$ must have been running the protocol using the same data as $B$ but we have seen that in the situation presented this is not true.

The idea of *Agreement* is general and it can be obviously defined using different models and languages. For example, in [9] it is given the following simple characterization of *Agreement* which is based on a slight extension of CCS [10]. Suppose that $P$ is the CCS specification of the two parties with the enemy between them.

**Definition 3.1** *$P$ guarantees $Agreement(role1, role2, ds)$ iff for every possible instance $D$ of the variables in $ds$ we have that:*

$$P \setminus \{signal_running_role1(A1, A2, D)\}$$
$$\approx_T$$
$$P \setminus \{signal_running_role1(A1, A2, D), signal_commit_role2(A2, A1, D)\}$$

*where ($role1 = ini$, $role2 = res$) or ($role1 = res$, $role2 = ini$) and $A1$ and $A2$ are the two entities playing respectively the roles $role1$ and $role2$.* ■

[a] Note instead that we have agreement on nonces $N_A$ and $N_B$.

We do not want to give the details of the CCS language here. It is sufficient to observe that $E \setminus L$ is the *restriction* operator which forbids to process $E$ the execution of the actions in set $L$, and $\approx_T$ is the *trace equivalence* which equates processes that can execute exactly the same sequences of actions. So $P \setminus \{signal_running_role1(A1, A2, D)\}$ behaves like protocol $P$ but it is not allowed to execute the action $signal_running_role1(A1, A2, D)$. We require its trace equivalence to protocol $P$ where we also forbid the execution of action $signal_commit_role2(A2, A1, D)$. This means that when we forbid the execution of $signal_running_role1(A1, A2, D)$ we cannot have a $signal_commit_role2(A2, A1, D)$ anymore, hence, in protocol $P$, action $signal_commit_role2(A2, A1, D)$ is always executed after a $signal_running_role1(A1, A2, D)$.

Note that this definition reduces the problem of checking *Agreement* to a problem of equivalence check over CCS terms. Hence, we can use for example the Concurrency Workbench [11] (or every other tool which is able to check trace equivalence over CCS terms) to automatically check this property.

Now we want to be sure that at the end of the protocol both $A$ and $B$ agree on respective addresses. Indeed, if $A$ and $B$ agree on the addresses they are sure (at least for the instance of the authentication exchange) to communicate one to the other. $A$ is sure that $B$ is connected at $Addr_B$ and $B$ is sure that $A$ is at $Addr_A$. We call this extension of the *Agreement* property *StrongAgreement*. Once we have made explicit the addresses, this property can be easily defined as follows:

**Definition 3.2** *A protocol satisfies* $StrongAgreement(role1, role2, ds)$ *iff it satisfies property* $Agreement(role1, role2, ds \cup \{addr_ini, addr_res\})$, *where addr_ini and addr_res represent the addresses of the two parties.* ■

As we have shown above, the Lowe-Needham-Schroeder protocol does not satisfy $StrongAgreement(ini, res, \{n_ini, n_res\})$. As a matter of fact, at the end of the protocol $B$ is wrongly convinced that the address of $A$ is $Addr_E$ thus $A$ and $B$ do not agree on respective addresses.

### *3.3 Located Authentication*

In the previous section we have presented a definition of authentication which takes care of addresses. However it is a little bit complicated and it is not clear if we obtain authentication because of the agreement on addresses or on other data (e.g., nonces). We conclude presenting a property that is very simple to check and is based only on the presence of the addresses in the specification. It is based as well on the presence of the *signal_commit* actions. However we do not need to have a global view of the (distributed) execution, we can

simply observe if certain events do not occur locally. The definition can be given as follows:

**Definition 3.3** *A protocol guarantees LocatedAuthentication*$(role1, role2)$ *if and only if it never generates inconsistent commitments, i.e., iff every time a signal_commit_role*$2(A2, A1, \{addr1, addr2\})$ *is generated, we have that addr*$1 = Addr_{A1}$ *and addr*$2 = Addr_{A2}$*, where A*1*, A*2*, represent the entities playing the roles role*1 *and role*2*.* ■

Intuitively, every time the entity with role *role*2 completes the protocol and is convinced that the addresses of the two parties are *addr*1 and *addr*2 then these addresses are indeed correct. The idea is that every entity is connected at a certain address and we require that the protocol never gives incorrect bindings between entities and addresses. This property is weaker than *StrongAgreement*. Basically *LocatedAuthentication* does not imply liveness which is instead implied by *Agreement* [3]. In fact, here we are simply guaranteed that if $A$ completes the protocol then she will know the right address of $B$. However $B$ could have never run the protocol. So this property does not guarantee that the two parties have participated in the protocol. This "weakness" of the property is indeed what makes it so simple. In fact if we want to have guarantees about the participation to the protocol of the two parties, we have to observe if there is some kind of matching between their executions (as in *Agreement*). Here we are simply requiring that certain events (inconsistent commits) do not occur.

*LocatedAuthentication* can indeed detect the classic attack found in [6] and reported in Section 2. Consider it again as reported in section 3.1, i.e., with explicit addresses. At the end of this execution we obtain from agent $A$ an action $signal_commit_ini(A, E, \{Addr_A, Addr_E\})$ which is consistent and represent the fact that $A$ is (correctly) convinced to communicate with $E$. We also have a $signal_commit_res(B, A, \{Addr_E, Addr_B\})$ which is inconsistent and so the protocol does not satisfy *LocatedAuthentication*. It represents the fact that B is convinced that $E$ (i.e., the entity connected at $Addr_E$) is $A$. So *LocatedAuthentication* is able to detect this attack as well as *Agreement*. Note that they discover the attack in a completely different way. Here we directly observe that $B$ is convinced to communicate with $A$ but he has run the protocol with $E$. This can be done because we exploit the presence of explicit addresses and the "off-line" information about how entities are bound to addresses, i.e., about the consistence of commitments. In *Agreement* we observe that $B$ correctly terminates the protocol with $A$ but $A$ has never run the protocol with $B$. Note that with *Agreement* we deduce that there is an attack by observing the behaviour of both the parties. Here we only observe an inconsistent behaviour of $B$.

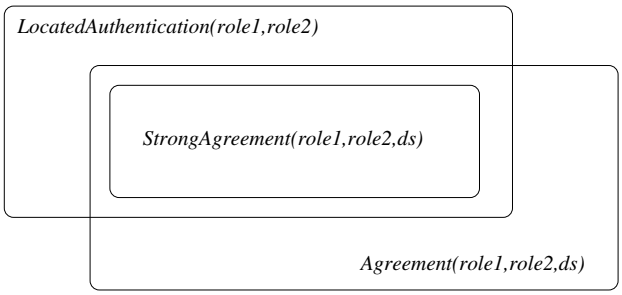


Figure 2. Inclusion diagram. It holds for all sets of data *ds*.

It is interesting to observe that *LocatedAuthentication* and *Agreement* are incomparable. In particular, we have seen that *LocatedAuthentication* requires that addresses are explicitly transmitted in the encrypted part of the messages. Moreover, we have observed that *Agreement* implies liveness which is not guaranteed by *LocatedAuthentication*. We find a correlation between these two properties in *StrongAgreement* which implies both of them (see Figure 2). The fact that $StrongAgreement(role1, role2, ds)$ is stronger that $Agreement(role1, role2, ds)$ derives from the monotonicity of *Agreement* with respect to the set of relevant data: *Agreement* over a certain set $S$ is strictly stronger than *Agreement* over a set $S' \subset S$ [3]. Moreover, the following holds (see [12] for the proof):

**Proposition 3.4** *For all sets of data ds we have that:*

$$StrongAgreement(role1, role2, ds) \Rightarrow LocatedAuthentication(role1, role2).$$

■

Note that in *LocatedAuthentication* we do not have to choose a set of relevant data (which could be a non trivial task). Moreover it is very simple to check. If we have a transition system as the semantics of the specifications (like in CCS language), we can simply check if the protocol can reach a state from which it is possible to execute an inconsistent commitment event. For finite state specifications, this can be done by simply exploring the state space.

## 4 Conclusion

In literature there are a number of proposals for a formal definition of entity authentication. However they do not approach directly the problem of defining authentication. Instead, they try to characterize authentication through

other properties, e.g., liveness. We have shown that the introduction of explicit information about addresses can help to find a direct definition of authentication. We have extended the *Agreement* property in order to consider addresses as relevant. Properties similar to this one, i.e., that take care of the addresses, can be defined also through other formalisms. It is basically a matter of how many details we put in the specification/model. For example, in [4], the author uses a model which considers the entity name and the addresses on the net as the same thing. It could probably be extended in order to handle entities which are not related to a fixed address. In [13,14] it is presented a technique to analyze security protocols using properties based on Non-Interference [15,16]. In this case, simply by refining the specification, it is possible to obtain an authentication property which guarantees that an enemy is not able to interfere on the "correct" commitments of the two parties. This means that it is not possible that an entity at the end of the protocol believes that another entity is connected at the wrong address on the net. A comparison among all these properties can be found in [9].

Explicit entity addresses have allowed us to define a very simple property which seems to guarantee a form of entity authentication which is in a sense, orthogonal to the classic definitions. In fact it is not based on concepts similar to liveness, but it simply requires that the protocol is able to give out only correct mappings of addresses into entities. A mapping of this kind can be seen as a confirmation that a certain entity is logged at a certain address. In this way we obtain a formalization of entity authentication in a very simple and intuitive way.

We have shown how this simple property can detect the classic attack to the Needham-Schroeder protocol, and we have seen that it can be verified automatically in linear time w.r.t. the number of states.

As a side effect of considering addresses as relevant we obtain that the protocols must take care of addresses too. For example for the Needham-Schroeder protocol we should include the addresses in the encrypted part of the messages. This can be seen as part of a verification methodology and deserves to be further investigated. Moreover we want to test this location based property on more examples. The check can be automatically done using the CoSeC tool [14] with slight modifications. It is also our intention to work on formal comparisons with other existing definitions.

## References

1. C. Boyd. "Extensional Goals in Authentication Protocols". In *Proceeding of the DIMACS Workshop on Design and Formal Verification of Security*

*Protocols*, September 1997.
2. D. Gollmann. "What do we mean by Entity Authentication". In *Proceedings of the 1996 Symposium on Security and Privacy*, pages 46–54. IEEE Computer Society Press, 1996.
3. G. Lowe. "A Hierarchy of Authentication Specification". In *Proceedings of the 10th Computer Security Foundation Workshop*. IEEE press, 1997.
4. S. Schneider. "Verifying authentication protocols with CSP". In *Proceedings of the 10th Computer Security Foundation Workshop*. IEEE press, 1997.
5. R. Needham and M. Schroeder. "Using encryption for authentication in large networks of computers". *Communications of the ACM*, 21(12):993–999, 1978.
6. G. Lowe. "An attack on the Needham-Schroeder public-key authentication protocol". *Information Processing Letters*, 53:97–101, 1995.
7. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
8. A. W. Roscoe. "Model Checking CSP". In A. W. Roscoe (ed) *A Classical Mind*. Prentice Hall, 1994.
9. R. Focardi and R. Gorrieri. "Analysis of Authentication Protocols using Non-Interference". Sumbitted for Publication.
10. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
11. R. Cleaveland, J. Parrow, and B. Steffen. "The Concurrency Workbench: a Semantics Based Tool for the Verification of Concurrent Systems". *ACM Transactions on Programming Languages and Systems*, Vol. 15 No. 1:36–72, January 1993.
12. R. Focardi. "Located Entity Authentication". Technical Report CS98-5, University of Venice, 1998.
13. R. Focardi, A. Ghelli, and R. Gorrieri. "Using Non Interference for the Analysis of Security Protocols ". In *Proceeding of the DIMACS Workshop on Design and Formal Verification of Security Protocols (H. Orman and C. Meadows Ed.)*, DIMACS Center, Rutgers University, September 1997.
14. R. Focardi and R. Gorrieri. "The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties". *IEEE Transactions on Software Engineering*, 23(9), September 1997.
15. J. A. Goguen and J. Meseguer. "Security Policy and Security Models". In *Proceedings of the 1982 Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society Press, April 1982.
16. R. Focardi and R. Gorrieri. "A Classification of Security Properties for Process Algebras". *Journal of Computer Security*, 3(1):5–33, 1994/1995.