UNIVERSITÀ CA' FOSCARI DI VENEZIA
Dip. di Matematica Applicata ed Informatica
Technical Report Series on Computer Science

Technical Report CS-98-5

20 March 1998

Riccardo Focardi

Located Entity Authentication

Corso di Laurea in Informatica
Via Torino, 155 – 30173 – Venezia Mestre

Located Entity Authentication *

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

20 March 1998

Abstract

Entity authentication should allow the verification of an entity's claimed identity, by another entity [4]. In the last years a number of definitions of entity authentication have been proposed (see, e.g., [1, 5, 8]). We show that with these notions of authentication it is not possible to verify the identity of an entity connected in a particular point of the net. We show that also the fixed Needham-Schroeder protocol is not able to deal with this kind of verification. For this reason we introduce the idea of *located entity authentication*, i.e., we make addresses of entities explicit and we consider them as relevant for authentication. We give two formalization of this idea: one is a natural extension (considering addresses) of the *Agreement* property [8]. The other is based on the simple idea of reducing 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 the latter 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

Entity authentication should allow the verification of an entity's claimed identity, by another entity [4]. In the last years a number of 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 to the protocol or at least wished to communicate with A (see, e.g., [1, 5]). Others require some agreement on exchanged data like nonces [8]. However they do not seem to guarantee a complete verification of entity identity. With these notions of authentication it is not possible to verify the identity of an entity connected in a particular point of the net. In fact, they can guarantee that both A and B wish to communicate one with the other, that they participated to the protocol and that they agree on some exchanged data, but A and B could be everywhere on the net. This seems to be in contrast with the idea of entity authentication. As an example, suppose that someone connected at a certain address on the net is sending a message of the form "Hello Alice! I'm Bob" to Alice and Alice wants to verify that this entity who claims to be Bob is indeed Bob. We have that Alice in not interested in knowing if Bob is somewhere on the net, but she would like to discover if Bob is indeed the entity connected at that particular address.

For this reason, we introduce the idea of *located entity authentication*: we refine the specification model by making explicit the addresses of the entities involved in the protocol and we consider them as relevant

*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".

for authentication. We give a first formalization of this idea as a natural extension (considering addresses as relevant) of the *Agreement* property [8]. Then we propose a new simple definition which is not based on the *Agreement* property but that simply exploits the presence of the addresses in the specification. We call this property *LocatedAuthentication*. It is based on the following very simple intuition: we reduce entity authentication to a mapping of addresses into entities. For example, consider someone connected at a certain address *Addr* who sends to A a message claiming to be B. A wants to verify the identity of such an entity. We can see this as the problem of mapping address *Addr* into the entity which is presently connected there. In order to have a protocol which guarantees *LocatedAuthentication*, we then require that it never gives out incorrect mappings, even in the presence of an enemy.

LocatedAuthentication is very simple and can be easily checked with an automatic tool. It seems to guarantee the verification of an entity's claimed identity. Moreover it can be used to detect interesting known flaws: as an 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 explain why we should make addresses explicit in the specification of entity authentication protocols. In Section 3.2, we present a natural extension of the *Agreement* property [8] which exploits explicit addresses considering them as relevant data. We give a version of the Lowe-Needham-Schroeder protocol which satisfies this new property. 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 [9] is to establish mutual authentication between two users *Alice* (A) and *Bob* (B). It uses public key cryptography.

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:

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

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 α and β to indicate the two runs and $E(A)$ to represent the intruder that claims to be A, the sequence that allows to cheat is:

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

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 massage ($\alpha 2$). 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:

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

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 why we should make addresses explicit in the specification of entity authentication protocols. In Section 3.2, we present a natural extension of the *Agreement* property [8] which exploits explicit addresses considering them as relevant data. We give a version of the Lowe-Needham-Schroeder protocol which satisfies this new property. 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 following situation:

$$
\begin{array}{llcll}
Message\ \gamma 1. & A & \rightarrow & E(B) & : & PK_B(N_A, A) \\
Message\ \delta 1. & E(A) & \rightarrow & B & : & PK_B(N_A, A) \\
Message\ \delta 2. & B & \rightarrow & E(A) & : & PK_A(N_A, N_B, B) \\
Message\ \gamma 2. & E(B) & \rightarrow & A & : & PK_A(N_A, N_B, B) \\
Message\ \gamma 3. & A & \rightarrow & E(B) & : & PK_B(N_B) \\
Message\ \delta 3. & E(A) & \rightarrow & B & : & PK_B(N_B)
\end{array}
$$

We have that A starts the authentication protocol with E thinking that E is B. Then, E impersonates A and starts another session with B. The communication proceeds with E forwarding every message

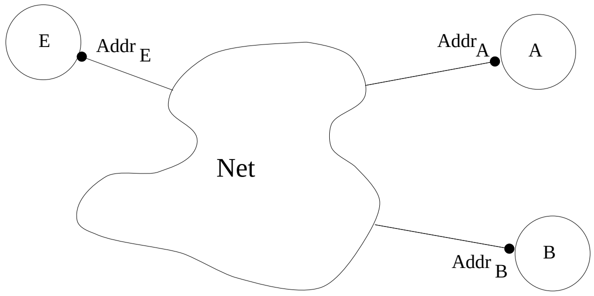

Figure 1: Mapping of entities to addresses

in both directions without learning any information. This execution is not considered as an attack to entity authentication by a number of existent definitions [1, 5, 8]. The basic point is that A and B are indeed communicating although in an indirect way. Most of the existent definitions consider entity authentication as something strictly related to liveness: at the end of the protocol, A can conclude that somewhere in the net B is alive or that B has participated to the protocol or at least that wished to communicate with A. In [8] it is also required that they agree on some information (i.e., N_A and N_B). According to these definitions, the situation above is not dangerous. In fact, at the end of the execution, A can conclude that B is alive, that B has participated to the protocol and that wished to communicate with A. Moreover A and B agree on the exchanged nonces. However A cannot conclude that B is the entity she is sending the messages to, i.e., the addressee of the messages. In fact, A is sending messages to E (and the same holds for B). Note that A sends her first message $\gamma 1$ to the address where E is connected. This is presumably done in order to verify that the identity of the entity connected there, is B. A could be unsatisfied by simply learning that B is somewhere on the net. As an (extreme) example, suppose that A is a bank server that is going to give out some cash through an automatic machine corresponding to E's address. In this case it is relevant to know who indeed is connected at that address. The bank server would be obviously unsatisfied of learning that B is connected at "some" automatic machine.

In order to better analyze this point it is useful to make explicit the address fileds in the messages. Sometimes, this is done by adding the names of the entities at the beginning of the messages (see, e.g., [7]). For example, Message 1 of the protocol becomes:

$$A \to B : A, B, PK_B(N_A, A)$$

However in this way it is not clear if there is a difference between the address and the name of an entity. Perhaps, in some situation a certain address on the net always corresponds to the same entity. This is true, for example, if we have a server connected at a certain fixed address (i.e., a fixed port of some machine). However we can also think of a user that accesses the net everywhere trying to authenticate himself with a remote server. In this case the entity (user) can be everywhere and is not related to a particular address. So it is less ambiguous if we choose different names for entities and addresses in the net. We can think that at a certain instant (when the protocol is executed) we have a particular mapping of the entities into the addresses. We call $Addr_E$ the current address for a generic entity E (see Figure 1). We can write:

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

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

$$A \to E(B) : Addr_A, Addr_E, PK_B(N_A, A)$$

So, it might be possible that a message addressed to a certain entity (e.g., B) is sent to a wrong address ($Addr_E$ in this case). In the example above, A has no way of relating B with $Addr_B$ at the beginning of the protocol. We can say that $E(B)$ represents entity E (connected at $Addr_E$) behaving like or pretending to be B. Note that part of the message above is encrypted with B's public key since it was addressed to B. For example, this message could be generated if E pretends to be B and A decides to start the protocol in order to verify $E(B)$'s identity. Similarly for Message $\delta 1$ 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 making explicit some data that are implicit in the notation $A \rightarrow B : Msg$.

3.2 Strong Agreement

We can re-write the whole execution as follows:

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

Note that, the subsequence of messages $\gamma 1$, $\delta 2$ and $\gamma 3$ shows a normal execution of the protocol where both A and B send messages to the wrong (E's) address.

So both A and B cannot conclude they are directly communicating one with the other. In other words they cannot conclude that the other party is connected to the net exactly where they are sending their messages. As we already stated above, this point is clear in some of the existing definitions of entity authentication but it is not considered as relevant. However, if E calling from $Addr_E$ says to A "I am B", and A wants to verify if this is true, A cannot use the Lowe-Needham-Schroeder protocol, because of the situation we have shown above. So, the Lowe-Needham-Schroeder protocol does not allow the verification of an entity's (E's) claimed identity by A. Consequently, it does not guarantee entity authentication as defined in [4]:

> "Entity authentication mechanisms allow the verification of an entity's claimed identity, by another entity. The authenticity of the entity can be ascertained only for the instance of the authentication exchange".

In the following we want to show how explicit addresses can be used to give a formal definition of entity authentication which seems to satisfy the informal requirement above. The idea is to consider the *Agreement* property defined in [8] and add entity addresses in the set of 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 pretend to be B, terminates correctly. This 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 [8] and 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 in [8], we call this property $Agreement(Resp, Init, ds)$. Basically it authenticates the responder to the initiator. In order to have mutual authentication we need also the symmetric property $Agreement(Init, Resp, ds)$. 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) [8]. As we said above, we add the addresses in the set of relevant data. For example, $Agreement(Init, Resp, \{na, nb\})$ (one of the property verified on the Lowe-Needham-Schroeder protocol) becomes $Agreement(Init, Resp, \{addra, addrb, na, nb\})$, where $addra, addrb, na, nb$ 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.

What is technically done in the *Agreement* property is to have 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. Formally $Agreement(Init, Resp, \{addra, addrb, na, nb\})$ requires that when B executes:

$$signal.commit.res.B.A.\{Addr_E, Addr_B, N_A, N_B\}$$

then A has previously executed:

$$signal.running.ini.A.B.\{Addr_E, Addr_B, N_A, N_B\}$$

This means that every time B completes the protocol convinced that addresses and nonces are $\{Addr_E, Addr_B, N_A, N_B\}$, 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.

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*.

Definition 3.1 *A protocol satisfies* $StrongAgreement(role1, role2, ds)$ *if and only if it satisfies property* $Agreement(role1, role2, ds \cup \{addra, addrb\})$*, where* $addra$ *and* $addrb$ *represent the addresses of the two parties.* ■

The Lowe-Needham-Schroeder protocol does not satisfy $StrongAgreement(Init, Resp, \{na, nb\})$ as we have shown above.

Before presenting a version of the Needham-Schroeder protocol which satisfies *StrongAgreement* we want to show a more subtle attack to the protocol:

$$\begin{array}{llcll}
Message\ \epsilon 1. & A & \rightarrow & B & : \ Addr_A, Addr_B, PK_B(N_A, A) \\
\multicolumn{5}{c}{-\ E \text{ intercepts message } \epsilon 1\ -} \\
Message\ \mu 1. & E(A) & \rightarrow & B & : \ Addr_E, Addr_B, PK_B(N_A, A) \\
Message\ \mu 2. & B & \rightarrow & A & : \ Addr_B, Addr_E, PK_A(N_A, N_B, B) \\
Message\ \epsilon 2. & E[B] & \rightarrow & A & : \ Addr_B, Addr_A, PK_A(N_A, N_B, B) \\
Message\ \epsilon 3. & A & \rightarrow & B & : \ Addr_A, Addr_B, PK_B(N_B) \\
\multicolumn{5}{c}{-\ E \text{ intercepts message } \epsilon 3\ -} \\
Message\ \mu 3. & E(A) & \rightarrow & B & : \ Addr_E, Addr_B, PK_B(N_B)
\end{array}$$

where $E[B]$ means that E is not only acting like it was B (as in $E(B)$) but is also using $Addr_B$ as his address. Note that here we assume that the enemy is able to fake his address when sending a message and to intercept and subtract messages from the net. This could require a quite powerful enemy, e.g., located on a gateway node between the two users.

This situation is really similar to the one presented above in this section. However, here A starts the protocol on the correct B's address but the enemy intercepts every messages of A. In particular messages $\epsilon 1$ and $\epsilon 3$ are directed to $Addr_B$ and are intercepted by the enemy. Moreover message $\epsilon 2$ is sent by the enemy using B's address $Addr_B$. In this attack it is not necessary that A starts the protocol with the enemy. Basically A is not aware of E's presence.

We obtain a version of the Needham-Schroeder protocol which satisfies the *StrongAgreement* property by simply including the addresses in the encrypted part of the messages:

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

It is easy to see that this version satisfies *StrongAgreement* on $\{N_A, N_B\}$. In fact, since we send the addresses together with the identities we are guaranteed that in the *running* and *commit* events we will always have the correct addresses and consequently we will have agreement.

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 use the *Agreement* property. The definition can be given as follows:

Definition 3.2 *A protocol guarantees LocatedAuthentication*$(role1, role2)$ *iff it never generates inconsistent commitments, i.e., iff every time a* $signal.commit.role2.A2.A1.\{addr1, addr2\}$ *is generated, we have that* $addr1 = Addr_{A1}$ *and* $addr2 = Addr_{A2}$*, where* $A1$*,* $A2$*, represent the entities playing the roles* $role1$ *and* $role2$*.* ■

Intuitively, every time the entity with role $role2$ completes the protocol and is convinced that the addresses of the two parties are $addr1$ and $addr2$ 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* [8]. In fact we are here simply guaranteed that if A completes the protocol then she will know the right address of B. However B could never have run the protocol. So this property does not guarantee that the two parties have participated to the protocol. However it is very simple and can detect the basic attack found in [6]. Consider again the attack reported in Section 2 with explicit addresses:

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

At the end of this execution we obtain from A a $signal.commit.ini.A.E.\{Addr_A, Addr_E\}$ 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 in 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

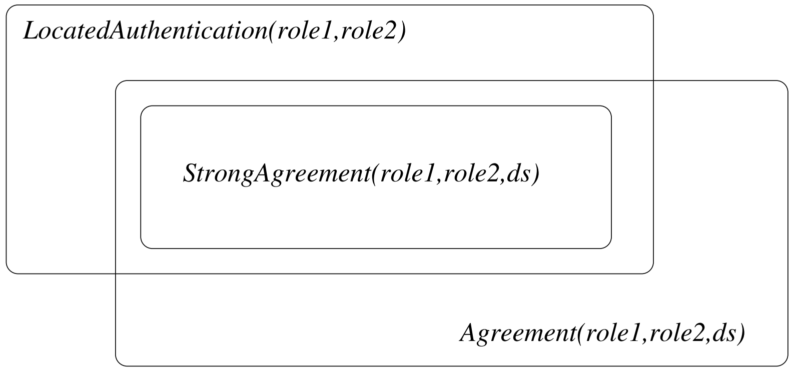

Figure 2: Inclusion diagram. It holds for all sets of data ds.

$Addr_E$) is A. So *LocatedAuthentication* is able to detect this attack as well as *Agreement* on nonces. Note that they consider completely different data. Here we do not consider nonces. We only look at addresses.

Note that *LocatedAuthentication* and *Agreement* on nonces 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 $S' \subset S$ [8]. The following holds:

Proposition 3.3 *For all sets of data ds we have that:*

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

PROOF. Because of the monotonicity of *Agreement* we have that $StrongAgreement(role1, role2, ds) \Rightarrow StrongAgreement(role1, role2, \emptyset)$. So it is sufficient to prove that:

$$StrongAgreement(role1, role2, \emptyset) \Rightarrow LocatedAuthentication(role1, role2)$$

which is equivalent to:

$$\neg LocatedAuthentication(role1, role2) \Rightarrow \neg StrongAgreement(role1, role2, \emptyset)$$

Consider a protocol which does not satisfy $LocatedAuthentication(role1, role2)$. It will lead to some inconsistent commitments. In particular, we will have a $signal.commit.role2.A2.A1.\{Addr_{A2}, Addr_Y\}$ with $A2 \neq Y$. Note that $Addr_{A2}$ is correct since $A2$ will certainly know his address. Now, we have that $StrongAgreement(role1, role2, \emptyset)$ requires that $signal.running.role1.A1.A2.\{Addr_{A2}, Addr_Y\}$ has been previously executed, but this is not possible since $A1$ will never commit on the wrong $Addr_Y$ for himself. Hence $StrongAgreement(role1, role2, \emptyset)$ is not satisfied by the protocol and we obtain the thesis. ■

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 some kind of transition system as the semantics of the specifications, 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. In some situations, they seem to be in contrast with the idea of entity authentication. In particular, they do not always allow the verification of an entity's claimed identity because they do not consider the location of entities on the net as relevant.

We have shown how it is possible to strengthen existing definitions (e.g., *Agreement*) by introducing explicit addresses for entities. 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 [10], 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 [2, 3] it is presented a technique to analyze security protocols using Non-Interference. 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. It would be really useful to have a formal comparison among all these different definitions.

Moreover, 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 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 future work we will check this property on more examples. We think that the check can be automatically done using the CoSeC tool [3] with slight modifications. We will also work on 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] 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.

[3] 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.

[4] International Organization for Standardization. Information technology - Security techniques - Entity authentication machanism; Part 1: General model. ISO/IEC 9798-1, Second Edition, September 1991.

[5] 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.

[6] G. Lowe. "An attack on the Needham-Schroeder public-key authentication protocol". *Information Processing Letters*, 53:97–101, 1995.

[7] G. Lowe. "Breaking and Fixing the Needham-Schroeder Public-key Protocol using FDR". In *Proceedings of Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, pages 146–166, Passau (Germany), March 1996. Springer-Verlag, LNCS 1055.

[8] G. Lowe. "A Hierarchy of Authentication Specification". In *Proceedings of the 10th Computer Security Foundation Workshop*. IEEE press, 1997.

[9] R. Needham and M. Schroeder. "Using encryption for authentication in large networks of computers". *Communications of the ACM*, 21(12):993–999, 1978.

[10] S. Schneider. "Verifying authentication protocols with CSP". In *Proceedings of the 10th Computer Security Foundation Workshop*. IEEE press, 1997.