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

Technical Report CS-98-18

October 26, 1998

Riccardo Focardi and Roberto Gorrieri

Analysis of Authentication Protocols using Non-Interference

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

Analysis of Authentication Protocols using Non-Interference *

Riccardo Focardi
Dipartimento di Informatica – Università Ca' Foscari di Venezia
via Torino, 155, I - 30173 Venezia Mestre, Italy
e-mail: focardi@dsi.unive.it, Phone: +39 41 2908438, Fax: +39 41 2908419.
Roberto Gorrieri
Dipartimento di Scienze dell'Informazione – Università di Bologna
Mura Anteo Zamboni 7, I - 40127 Bologna, Italy
e-mail: gorrieri@cs.unibo.it, Phone: +39 51 354509, Fax: +39 51 354510.

October 26, 1998

Abstract

Non interference [GM82] was originally proposed as a means for analyzing the security of computer systems. In this paper we show that its theory can be profitably exploited also for the analysis of security protocols, in particular for authentication. As a simple case study, we consider the paradigmatic example of the Needham-Schroeder public-key protocol, that was already shown incorrect and fixed in [Low96]. As expected, our analysis discovers the same bug; what is really new is the methodology, based on the idea of proving that potential external attackers cannot interfere on the correct execution of the protocol. We show that a number of existing definitions of authentication can be obtained as particular instances of our methodology, thus proving the generality of our method.

1 Introduction

Non-Interference (NI for short) [GM82] was introduced in order to circumvent difficulties in classic approaches to computer security. In order to limit, and possibly avoid, the damages produced by malicious programs (often called *Trojan Horses*) which try to leak secret information, it was suggested to impose some access control rules which limit the action of these programs. However, it is often difficult to detect the so-called "covert channels", i.e., indirect ways for transmitting information with no explicit reading of confidential information. This kind of difficulties induced some researchers to re-think from scratch the whole problem of guaranteeing security in computer systems.

NI is indeed an alternative, more general approach to computer security. It requires to control directly the whole flow of information, rather than the accesses of subjects to objects (as in classical access control rules). In general, NI-like properties are based on a formal model of system behaviour and on a definition of information flow on such a model. By imposing some information flow rules, we can control both the direct and indirect transmission channels.

Although the motivation for NI properties was the analysis of information flows inside systems, the basic ideas behind them are very general and potentially applicable to other related security areas. The aim of this paper is precisely to show that the theory of NI properties can be profitably applied in typical problems of network security. As a simple case study, we consider the paradigmatic example of

*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".

the Needham-Schroeder public-key protocol, that was already shown incorrect and fixed in [Low96]. As expected, our analysis discovers the same bug; what is really new is the methodology, based on the idea of proving that potential external attackers cannot interfere on the correct execution of the protocol. We have successfully applied this methodology to the verification of other authentication protocols. In Section 7 we briefly report some simple examples. Moreover, the methodology is general and is not tailored for authentication protocols only. In [FG97] the same technique presented here is used to check a simple session key distribution protocol based on public-key (see Section 7 for more details on this issue).

The basic idea which permits to express authentication requirements using NI-based properties is very simple: we want to check if an enemy is able to interfere with the correct execution of the protocol, i.e., with those actions which characterize the successful authentication. As an example, consider the attack reported in [Low96] where B is convinced to communicate with A but A has never sent an authentication request to B. This anomalous behaviour is caused by the enemy and can be observed through some particular actions specifically introduced to model correct executions of the protocol. In a correct execution, every run of B corresponds to a run of A. So we always observe an interleaving of the actions representing A's and B's activity. In the attack reported above, we have a successful execution of B but no execution of A, thus only the actions representing B's activity are observable. This difference in the executions is caused by the presence of an attack and can be detected through NI-based properties.

The method is general: instead of giving a definition of authentication, we simply select particular events that we want to observe and which represent the "correct" execution of the protocol. One typical choice is: one action for every message sent and received in the correct execution between the two partners. Then, we check if the enemy is able to interfere with the execution of those actions.

The formal verification of these properties is fully automatized thanks to the CoSeC tool [FG97] that accepts finite-state protocol specifications and checks over them the NI-based information flow properties.

In order to show that authentication is indeed guaranteed by NI-based verification, we prove that a number of existing definitions of authentication can be obtained as particular instances of our methodology. In particular we obtain that the *Agreement* property [Low97] can be seen as a NI property where we observe two particular couples of actions. Moreover, the message-oriented authentication defined in [Sch97, Sch96] is equivalent to a NI check over a particular protocol where we do not allow the execution of the authenticated actions and we observe only the authenticating ones.

The paper is organized as follows. In Section 2 we present the model and the NI-based properties we use in the protocol specification and analysis. In Section 3 we recall the Needham-Schroeder public-key protocol and we specify it using our language. Section 4 is about the attacker specification. In Section 5 we analyze the protocol using NI-based properties. Section 6 is about the relation between NI verification and existing authentication properties. Finally, in Section 7 a few additional examples are reported.

2 The Model

In this Section we introduce the model and the language we use for the specification of protocols and we also present the generalization of NI we use in the protocol analysis (see [FG95, FG97] for more details).

2.1 SPA and Semantic Equivalences

In the following, protocols will be specified using the *Security Process Algebra*, an extension of CCS [Mil89]. SPA has one additional operator, namely the hiding operator E/L of CSP [Hoa85], which is useful in characterizing some security properties in an algebraic style. Moreover the set of visible actions is partitioned into high and low level actions in order to specify multilevel systems.

SPA syntax is based on the following elements: a set $I = \{a, b, \ldots\}$ of *input* actions, a set $O = \{\bar{a}, \bar{b}, \ldots\}$ of *output* actions, a set $\mathcal{L} = I \cup O$ of *visible* actions, ranged over by α, and the usual function $\bar{\cdot} : \mathcal{L} \to \mathcal{L}$ such that if $a \in I$ then $\bar{a} \in O$ and if $\bar{a} \in O$ then $\bar{\bar{a}} = a \in I$; two sets Act_H and Act_L of high and low level actions such that $\overline{Act_H} = Act_H$, $\overline{Act_L} = Act_L$, $Act_H \cup Act_L = \mathcal{L}$ and $Act_H \cap Act_L = \emptyset$ where

$\bar{L} \stackrel{\text{def}}{=} \{\bar{a} : a \in L\}$; a set $Act = \mathcal{L} \cup \{\tau\}$ of actions (τ is the internal, invisible action), ranged over by μ; a set K of constants, ranged over by Z. The syntax of SPA *agents* is defined as follows:

$$E \quad ::= \quad \underline{0} \mid \mu.E \mid E + E \mid E|E \mid E \setminus L \mid E/L \mid E[f] \mid Z$$

where $L \subseteq \mathcal{L}$ and $f : Act \to Act$ is such that $f(\bar{\alpha}) = \overline{f(\alpha)}$ and $f(\tau) = \tau$. Moreover, for every constant Z there must be the corresponding definition: $Z \stackrel{\text{def}}{=} E$. The meaning of $\underline{0}$, $\mu.E$, $E+E$, $E|E$, $E \setminus L$, $E[f]$ and $Z \stackrel{\text{def}}{=} E$ is as for CCS [Mil89]. Intuitively, $\underline{0}$ is the empty process, which cannot do any action; $\mu.E$ can do an action μ and then behaves like E; $E_1 + E_2$ can alternatively choose [1] to behave like E_1 or E_2; $E_1|E_2$ is the parallel composition of E_1 and E_2, where the executions of the two systems are interleaved, possibly synchronized on complementary input/output actions, producing an internal τ. $E \setminus L$ can execute all the actions E is able to do, provided that they do not belong to $L \cup \bar{L}$; E/L turns all the actions in L into internal τ's; if E can execute action μ, then $E[f]$ performs $f(\mu)$; finally, Z does what E does, when $Z \stackrel{\text{def}}{=} E$.

As we said above, all the operators, except hiding, are standard operators of CCS. The hiding operator can be defined as $E[f_L]$, where f_L is the function that relabels with τ all the actions in L, leaving all the other actions unchanged. So we do not need to introduce a new semantic rule for it.

Let $\mathcal{E}$ be the set of SPA agents, ranged over by E, F. Let $\mathcal{L}(E)$ denote the *sort* of E, i.e., the set of the (possibly executable) actions occurring syntactically in E. The sets of high level agents and low level ones are defined as $\mathcal{E}_H \stackrel{\text{def}}{=} \{E \in \mathcal{E} \mid \mathcal{L}(E) \subseteq Act_H \cup \{\tau\}\}$ and $\mathcal{E}_L \stackrel{\text{def}}{=} \{E \in \mathcal{E} \mid \mathcal{L}(E) \subseteq Act_L \cup \{\tau\}\}$, respectively. The operational semantics of SPA is given (as usual) associating to each agent a particular state of the labelled transition system $(\mathcal{E}, Act, \to)$ where $\to \subseteq \mathcal{E} \times Act \times \mathcal{E}$ and, intuitively, $E \stackrel{\mu}{\to} E'$ means that agent E can execute μ moving to E'. An *isomorphism* between labelled transition systems is defined as a labelled transition preserving bijection between states. We will use the notation $E \stackrel{\text{lts}}{=} F$ to indicate that E and F generate isomorphic labelled transition systems, where the isomorphism maps E to F.

In the following the expression $E \stackrel{\mu}{\Longrightarrow} E'$ is a shorthand for $E(\stackrel{\tau}{\to})^* E_1 \stackrel{\mu}{\to} E_2(\stackrel{\tau}{\to})^* E'$, where $(\stackrel{\tau}{\to})^*$ denotes a (possibly empty) sequence of τ labelled transitions. We also extend the '$\Longrightarrow$' notation to sequences of actions; $E \stackrel{\gamma}{\Longrightarrow} E'$ with $\gamma \in \mathcal{L}^+, \gamma = \gamma_1\gamma_2\ldots\gamma_n$ means that $\exists E_1, E_2, \ldots, E_{n-1}$ such that $E \stackrel{\gamma_1}{\Longrightarrow} E_1 \stackrel{\gamma_2}{\Longrightarrow} \ldots \stackrel{\gamma_{n-1}}{\Longrightarrow} E_{n-1} \stackrel{\gamma_n}{\Longrightarrow} E'$. For the empty sequence ϵ we have that $E \stackrel{\epsilon}{\Longrightarrow} E'$ stands for $E(\stackrel{\tau}{\to})^* E'$.

We recall here the definition of trace equivalence

Definition 2.1 *A trace of a process E is a sequence of actions that E can execute. The set of traces is defined as follows: $T(E) \stackrel{\text{def}}{=} \{\gamma \in \mathcal{L}^* \mid \exists E' : E \stackrel{\gamma}{\Longrightarrow} E'\}$. E and F are* trace equivalent *(notation $E \approx_T F$) if and only if $T(E) = T(F)$.* ■

2.2 Value-Passing SPA

In this section we briefly present a value-passing extension of "pure" SPA (VSPA, for short). In all the examples reported in this paper we will use such value-passing calculus, because it leads to more readable specifications than those written in pure SPA.

As an example, consider the following buffer cell [Mil89]:

$$\begin{aligned} C &\stackrel{\text{def}}{=} in(x).C'(x) \\ C'(x) &\stackrel{\text{def}}{=} \overline{out}(x).C \end{aligned}$$

where x is a variable that can assume values in $\mathbf{N}$ (we usually write $x \in \mathbf{N}$). C reads a natural value n through action in and stores it in variable x. Then this value is passed to agent C' which can give n as

[1] For notational convenience, we use sometimes the $\sum$ operator to represent a general n-ary (or even infinitary) sum operator, meaning $\underline{0}$ if $n = 0$.

output through action *out* moving again to C. So C represents a buffer which may hold a single data item.

Now we show how C can be translated into an SPA agent. The parametrized constant C' becomes a family of constants C'_v one for each value $v \in \mathbf{N}$. Similarly $\overline{out}(x)$ becomes a family $\overline{out}_v$ of prefixes. So the single definition for C' becomes the following family of definitions:

$$C'_v \stackrel{\text{def}}{=} \overline{out}_v.C \quad (v \in \mathbf{N})$$

Now consider the prefix $in(x)$. To reflect the fact that it can accept any input value, binding variable x, we translate it into $\sum_{v \in \mathbf{N}} in_v$. So the definition becomes:

$$C \stackrel{\text{def}}{=} \sum_{v \in \mathbf{N}} in_v.C'_v$$

VSPA is very similar to the value-passing CCS introduced in [Mil89]. The main difference is that in VSPA we can have more than one parameter for actions and parameters are multi-sorted. The syntax of VSPA agents is defined as follows:

$$\begin{aligned} E \quad ::= \quad & \underline{0} \mid a(x_1, \ldots, x_n).E \mid \bar{a}(e_1, \ldots, e_n).E \mid \tau.E \mid E + E \mid E|E \mid E \setminus L \mid \\ & \mid E/L \mid E[f] \mid A(e'_1, \ldots, e'_n) \mid \textbf{if } b \textbf{ then } E \mid \textbf{if } b \textbf{ then } E \textbf{ else } E \end{aligned}$$

where the variables $x_1, \ldots, x_n$, the value expressions $e_1, \ldots, e_n$ and $e'_1, \ldots, e'_n$ must be consistent with the arity of the action a and constant A, respectively (the arity specifies the sorts of the parameters), and b is a boolean expression.

It is also necessary to define constants as follows: $A(x_1, \ldots, x_m) \stackrel{\text{def}}{=} E$ where E is a VSPA agent which may contain no free variables except $x_1, \ldots, x_m$, which must be distinct. As in [Mil89] the semantics of the value-passing calculus is given as a translation into the pure calculus. The translation rests upon the idea that a single label a of VSPA, with n parameters with sorts $S_1 \ldots S_n$, becomes the set of labels $\{a_{v_1 \ldots v_n} : v_i \in S_i, \forall i \in [1, n]\}$ in SPA. We consider only agents without free variables because if an agent has a free variable then it becomes a family of agents, one for each value of the variable. The translation can be given inductively on the structure of agents (see [Mil89] for more details about this kind of translation).

Note that we do not partition the set of actions into two levels; we directly refer to the partition in the pure calculus. In this way it is possible for a certain action in VSPA to correspond, in the translation, to actions at different levels in SPA. This can be useful if we want a parameter representing the level of a certain action.

2.3 NI generalization

In this section we present the *Non Deducibility on Compositions* (*NDC*, for short) property, proposed in [FG95] as a generalization of NI in a process algebraic setting. It is simply based on the idea of checking the system against all high level potential interactions. A system E is *NDC* if for every high level process Π a low level user cannot distinguish E from $(E|\Pi) \setminus Act_H$. In other words, a system E is *NDC* if what a low level user sees of the system is not modified by composing any high level process Π to E.

Definition 2.2 *$E \in NDC$ if and only if* $\forall \Pi \in \mathcal{E}_H$, $E/Act_H \approx_T (E \mid \Pi) \setminus Act_H$. ■

This definition of *NDC* is difficult to use in practice, because of the universal quantification on high level processes. We need an alternative formulation of *BNDC* which avoids universal quantification, exploiting local information only. It is possible to prove [FG95] that *NDC* is equivalent to *Strong Non-deterministic Non Interference* (*SNNI*, for short) which is defined as follows:

Definition 2.3 *$E \in SNNI$ if and only if* $E \setminus Act_H \approx_T E/Act_H$. ■

This property is a natural generalization of NI [GM82] to non-deterministic systems. Intuitively, the high level does not interfere with the low level if the effects of high level actions are not visible by a low level user. Consider every trace γ of the system containing some high level actions; if there exists another trace γ' with the same subsequence of low level actions and without any high actions, then a low level user cannot conclude if the high level actions in γ' have been executed since it is not able to distinguish between the execution of γ and γ' (it can only observe low level actions). As both γ and γ' are legal traces, the possible execution of the high level actions in γ has no effect on the low level view of the system. As a matter of fact, in E/Act_H all the high level actions are hidden, hence giving the low level view of the system; $E \setminus Act_H$ instead removes all the traces containing high level actions. So, if the two terms are equivalent, then for every trace with high level actions we can find another trace without such actions but with the same subsequence of low level actions.

We have the following result [FG95]:

Proposition 2.4 *NDC = SNNI* ■

We have the following monotonicity property for *SNNI*: if a system E is *SNNI* then also E/L with $L \subseteq Act_L$ is *SNNI*. In other words, if Act_H does not interfere with Act_L then Act_H does not interfere with every subset of Act_L.

Proposition 2.5 *if $E \in$ SNNI then $E/L \in$ SNNI, $\forall L \subseteq Act_L$.*

Proof. We have that $E \setminus Act_H \approx_T E/Act_H$. It is easy to see that $\approx_T$ is a congruence for $-/L$ and so $E \setminus Act_H/L \approx_T E/Act_H/L$. Since $L \cap Act_H = \emptyset$ then $E \setminus Act_H/L \stackrel{\text{lts}}{=} E/L \setminus Act_H$ and $E/Act_H/L \stackrel{\text{lts}}{=} E/L/Act_H$. We obtain that $E/L \setminus Act_H \approx_T E/L/Act_H$. ■

3 Protocol specification in SPA

In this section we present a formalization of the Needham-Schroeder protocol using VSPA. The aim of the Needham-Schroeder protocol [NS78] 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 of seven steps, but four of them can be avoided if A and B have local caches of commonly used public keys. In this way we obtain the "core" of the protocol:

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

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 she is really talking to B since only B could have decrypted the first message. In the last message A replies to B, sending back nonce N_B. Following the same reasoning as above, B will conclude that is indeed talking with A.

The formalization we give is similar to the one in [Low96]. The protocol is modeled through two VSPA agents: the *Initiator* and the *Responder*. The *Initiator* accepts a user connection request, then starts the messages exchange with the *Responder*.

Protocol messages are all of the form:

$$msg_i(mitt, dest, enc(key, p_1, p_2, \ldots, p_n))$$

where:

- i is the message number (it ranges in [1 . . 3]);

- *mitt* is the identifier of the message sender;

- *dest* is the identifier of the message addressee;

- $enc(\ldots)$ is a generic public key encryption algorithm[2];

- *key* is an encryption key;

- $p_1, p_2, \ldots, p_n$ represents the information tuple conveyed by the (encrypted) message.

Messages "send" and "receive" are prefixed by the strings *send* and *receive*, respectively. Besides protocol messages, there are some actions that allow users to interact with the protocol. All the names of these actions are prefixed by the string *signal*.

User *ini_id* can make a connection request to *res_id* through action *signal_request(ini_id,res_id)*. To inform that the connection is established, the *Initiator* and the *Responder* give as output *signal_commit_ini(ini_id,res_id)* and *signal_commit_res(res_id,ini_id)*, respectively. In the case of a protocol failure the messages become: *signal_fail_ini(ini_id,res_id)* and *signal_fail_res(res_id,ini_id)*, respectively.

Let I_{id} and R_{id} be the sets of all the possible initiators and responders, respectively. Let *Nonces* be the set of all nonces used in the protocol execution. *Initiator* is defined using VSPA as follows:

$$
\begin{array}{lcl}
Initiator(ini_id, ini_nonce) & \stackrel{\text{def}}{=} & signal_request(ini_id, res_id). \\
 & & \overline{send_msg_1}(ini_id, res_id, enc(PK_{res_id}, ini_nonce, ini_id)). \\
 & & receive_msg_2(res_id, ini_id, enc(PK_{ini_id}, ini_nonce', res_nonce)). \\
 & & \quad \textbf{if } ini_nonce' = ini_nonce \textbf{ then} \\
 & & \quad\quad \overline{send_msg_3}(ini_id, res_id, enc(PK_{res_id}, res_nonce)). \\
 & & \quad\quad \overline{signal_commit_ini}(ini_id, res_id).0 \\
 & & \quad \textbf{else} \\
 & & \quad\quad \overline{signal_fail_ini}(ini_id, res_id).0
\end{array}
$$

where $ini_id \in I_{id}$, $res_id \in R_{id}$ and ini_nonce, ini_nonce', $res_nonce \in Nonces$.

Initiator receives, through action $signal_request(ini_id, res_id)$, a request from user *ini_id* for a connection with user *res_id*. Then it starts the protocol by sending message 1 containing the sender identifier and its nonce, both encrypted with receiver's public key. When it receives message 2 containing the two nonces *ini_nonce'*,*res_nonce* encrypted with the *Initiator* public key, it checks if *ini_nonce'* is equal to the nonce sent in the first message (*ini_nonce*). If this is the case, it sends the other nonce *res_nonce* to user *res_id* and informs the user *ini_id* that the connection with *res_id* is established (action $\overline{signal_commit_ini}(ini_id, res_id)$). If the nonce *ini_nonce'* is different from *ini_nonce*, the protocol fails and this is notified through action $\overline{signal_fail_ini}(ini_id, res_id)$.

Similarly to *Initiator*, the *Responder* is defined in VSPA as follows:

$$
\begin{array}{lcl}
Responder(res_id, res_nonce) & \stackrel{\text{def}}{=} & receive_msg_1(ini_id, res_id, enc(PK_{res_id}, ini_nonce, ini_id')). \\
 & & \quad \textbf{if } ini_id = ini_id' \textbf{ then} \\
 & & \quad\quad \overline{send_msg_2}(res_id, ini_id, enc(PK_{ini_id}, ini_nonce, res_nonce)). \\
 & & \quad\quad receive_msg_3(ini_id, res_id, enc(PK_{res_id}, res_nonce')). \\
 & & \quad\quad \textbf{if } res_nonce = res_nonce' \textbf{ then} \\
 & & \quad\quad\quad \overline{signal_commit_res}(res_id, ini_id).0 \\
 & & \quad\quad \textbf{else} \\
 & & \quad\quad\quad \overline{signal_fail_res}(res_id, ini_id).0 \\
 & & \quad \textbf{else} \\
 & & \quad\quad \overline{signal_fail_res}(res_id, ini_id).0
\end{array}
$$

where $ini_id, ini_id' \in I_{id}$, $res_id \in R_{id}$ and res_nonce, $ini_nonce \in Nonces$.

[2]We are not interested in analyzing possible attacks to this algorithm, so we do not specify it.

Responder receives message 1, containing sender's nonce and identity (*ini_nonce,ini_id'*). It checks if the identifier contained in the plaintext part of the message (*ini_id*) is equal to that contained in the ciphertext part. If it is so, it sends message 2 containing the nonce received and its own nonce, encrypted with sender's public key. Then, it receives message 3 and checks if the nonce received is equal to its own; if the check is successful, it informs that the connection is established through the action $\overline{signal_commit_res}(res_id, ini_id)$, otherwise it gives as output $\overline{signal_fail_res}(res_id, ini_id)$ which represents an unsuccessful termination of the protocol.

In order to complete the protocol specification we define an agent representing the communication channel. This agent is not strictly necessary (we could make communicate the Initiator and the Responder directly), however we will see that it gives a more realistic setting for the attacker.

The channel is able to get all the messages sent by the two parties and forward them appropriately. It can be defined as follows:

$$\begin{aligned} Channel \stackrel{\text{def}}{=}\ & send_msg_1(id1, id2, enc(PK_{id2}, n, id1)). \\ & \overline{receive_msg_1}(id1, id2, enc(PK_{id2}, n, id1)). \\ & Channel \\ & + \\ & send_msg_2(id2, id1, enc(PK_{id2}, n1, n2)). \\ & \overline{receive_msg_2}(id2, id1, enc(PK_{id2}, n1, n2)). \\ & Channel \\ & + \\ & send_msg_3(id1, id2, enc(PK_{id2}, n)). \\ & \overline{receive_msg_3}(id1, id2, enc(PK_{id2}, n)). \\ & Channel \end{aligned}$$

where $id1 \in I_{id}$, $id2 \in R_{id}$ and $n, n1, n2 \in Nonces$.

The complete protocol specification for A as initiator with nonce na and B as responder with nonce nb, is defined as parallel composition of the agents described above:

$$Protocol \stackrel{\text{def}}{=} (Initiator(A, na) \mid Channel \mid Responder(B, nb)) \setminus L$$

where $L = \{send_msg_1, receive_msg_1, send_msg_2, receive_msg_2, send_msg_3, receive_msg_3\}$ is the set of messages sent and received during the protocol. Moreover, $I_{id} = \{A\}$, $R_{id} = \{B\}$ and $Nonces = \{na, nb\}$. For more clarity, in the definition of the set L we have used only the name of a certain action (with no parameters) as a shorthand for the action where its parameters can assume all the possible values in the corresponding ranges. For example, $send_msg_1$ stands for $send_msg_1(id1, id2, enc(PK_{id2}, n, id1))$ where $id1 \in I_{id}$, $id2 \in R_{id}$ and $n \in Nonces$. We will often use this notation in the following.

4 The Attacker

In order to analyze the protocol it is necessary to study its behaviour when it is attacked by a malicious user. This intruder can be a normal user of the computer network that has decided to assume an incorrect behaviour. In particular, the intruder should be able to:

- intercept any sent message;
- store the information contained in intercepted message for later uses (we assume that the encryption algorithm used is secure: the enemy is able to decrypt a message only if he knows the key that though can be his own private key or one that he has learned from an intercepted message);
- introduce new messages into the system using information he knows.

Although the intruder is not able to break the encryption algorithm, it is still possible to have dangerous "man-in-the-middle" attacks as we will see in the following. In order to simplify the specification we do

not consider the possibility of storing intercepted information for later uses. In particular the enemy is able to use only the information gained in the last intercepted message when he introduces new messages in the system. This is different from what is done in, e.g. [Low96], where the enemy has a local memory where he can store stolen information and reuse it later. We could define this more complex enemy also in our model but we prefer to use the simpler version with no memory which is sufficient for our analysis.

First, we present the enemy activity for message 1 and then we will see the whole *Enemy* agent. The enemy basically intercepts messages from the channel (action prefixed by *receive*), and sends faked messages on the channel (actions prefixed by *send*). We have inserted C-like comments in order to better explain the process behaviour.

$Enemy_1(e_id, e_nonce) \stackrel{\text{def}}{=}$
`/* the Enemy intercepts msg_1 */`
$receive_msg_1(id1, id2, enc(PK_{id2}, nonce, id1)).$
(
`/* The Enemy can decide to do nothing with the intercepted message: */`
$Enemy_1(e_id, e_nonce)$
$+$
`/* or it can forward the message possibly changing some parts */`
if $PK_{id2} = PK_{e_id}$ **then**
`/* The message is encrypted with the Enemy public key so it can reuse the encrypted information */`
$\sum_{\substack{ini_id, id' \in I_{id} \\ res_id \in R_{id} \\ nonce' \in \{nonce, e_nonce\}}} \overline{send_msg_1}(ini_id, res_id, enc(PK_{res_id}, nonce', id')).Enemy_1(e_id, e_nonce)$
else
`/* The Enemy is not able to decrypt the message: it can only change the plaintext part */`
$\sum_{\substack{ini_id \in I_{id} \\ res_id \in R_{id}}} \overline{send_msg_1}(ini_id, res_id, enc(PK_{id2}, nonce, id1)).Enemy_1(e_id, e_nonce)$
)

where $id1 \in I_{id}$, $id2 \in R_{id}$ and $nonce, e_nonce \in Nonces$. We have an analogous situation for messages 2 and 3. So the VSPA specification for the *Enemy* is given by the agent in Table 1. The aim of the action $fake$ will be explained later. As for the *Initiator* and the *Responder*, the *Enemy* parameters are its own identifier and nonce.

5 Protocol Analysis

We want to reduce the problem of testing if the protocol correctly authenticates the two users to a problem of information flow control between two levels. For this reason the actions will be divided into high and low level actions (sets Act_H and Act_L). Since we want to check the influence that an intruder can have on the protocol execution, Act_H is composed by the action *fake*, which is executed at every *Enemy* iteration and so can be seen as an abstraction of the *Enemy* activity. All the other actions belong to the low level.

As we are considering the network as insecure, the right way to compose the *Enemy* with the partners of the protocol is exactly on the communication channel between the *Initiator* and the *Responder*. [3] This can be done in different ways that correspond to different kinds of control the enemy has on the channel. For example consider the protocol defined in Section 3 and a generic enemy E (not the particular one defined in the previous section). We can consider the parallel composition of E with the other processes,

[3] The semantics of the restriction operator on a channel name a ensures that such a channel is "perfectly" secure, i.e., no other process outside its scope can communicate through channel a with processes syntactically located inside its scope.

$$
\begin{array}{l}
Enemy(e_id, e_nonce) \stackrel{\text{def}}{=} \\
\quad fake.(\\
\quad\quad receive_msg_1(id1, id2, enc(PK_{id2}, nonce, id1)).(\\
\quad\quad\quad Enemy(e_id, e_nonce) \; + \\
\quad\quad\quad \textbf{if} \;\; PK_{id2} = PK_{e_id} \;\; \textbf{then} \\
\quad\quad\quad\quad \sum_{\substack{ini_id, id' \in I_{id} \\ res_id \in R_{id} \\ nonce' \in \{nonce, e_nonce\}}} \overline{send_msg_1}(ini_id, res_id, enc(PK_{res_id}, nonce', id')). Enemy(e_id, e_nonce) \\
\quad\quad\quad \textbf{else} \\
\quad\quad\quad\quad \sum_{\substack{ini_id \in I_{id} \\ res_id \in R_{id}}} \overline{send_msg_1}(ini_id, res_id, enc(PK_{id2}, nonce, id1)).Enemy(e_id, e_nonce) \\
\quad\quad\quad) \\
\quad\quad + \\
\quad\quad receive_msg_2(id2, id1, enc(PK_{id2}, nonce1, nonce2)).(\\
\quad\quad\quad Enemy(e_id, e_nonce) \; + \\
\quad\quad\quad \textbf{if} \;\; PK_{id2} = PK_{e_id} \;\; \textbf{then} \\
\quad\quad\quad\quad \sum_{\substack{ini_id \in I_{id} \\ res_id \in R_{id} \\ any_nonce1 \in \{nonce1, nonce2, e_nonce\} \\ any_nonce2 \in \{nonce1, nonce2, e_nonce\}}} \overline{send_msg_2}(res_id, ini_id, enc(PK_{ini_id}, any_nonce1, any_nonce2)). Enemy(e_id, e_nonce) \\
\quad\quad\quad \textbf{else} \\
\quad\quad\quad\quad \sum_{\substack{ini_id \in I_{id} \\ res_id \in R_{id}}} \overline{send_msg_2}(res_id, ini_id, enc(PK_{id2}, nonce1, nonce2)).Enemy(e_id, e_nonce) \\
\quad\quad\quad) \\
\quad\quad + \\
\quad\quad receive_msg_3(id1, id2, enc(PK_{id2}, nonce)).(\\
\quad\quad\quad Enemy(e_id, e_nonce) \; + \\
\quad\quad\quad \textbf{if} \;\; PK_{id2} = PK_{e_id} \;\; \textbf{then} \\
\quad\quad\quad\quad \sum_{\substack{ini_id \in I_{id} \\ res_id \in R_{id} \\ res_nonce \in \{nonce, e_nonce\}}} \overline{send_msg_3}(ini_id, res_id, enc(PK_{res_id}, res_nonce)).Enemy(e_id, e_nonce) \\
\quad\quad\quad \textbf{else} \\
\quad\quad\quad\quad \sum_{\substack{ini_id \in I_{id} \\ res_id \in R_{id}}} \overline{send_msg_3}(ini_id, res_id, enc(PK_{id2}, nonce)).Enemy(e_id, e_nonce) \\
\quad\quad\quad) \\
\quad\quad)
\end{array}
$$

Table 1: The VSPA specification of the *Enemy*

inside the scope of the restriction operator:

$$(Initiator(A, na) \mid Channel \mid Responder(B, nb) \mid E) \setminus L \tag{1}$$

we obtain that the enemy E is able to intercept and fake every message but a correct execution of the protocol is always possible. In fact, we still have the execution trace in which the enemy is idle and the parties can correctly execute the protocol. In order to model a more powerful attack we can, for example, replace the channel with one generic enemy E as follows:

$$(Initiator(A, na) \mid \ E \mid Responder(B, nb)\ ) \setminus L$$

Here the enemy E is getting a complete control of the communication. In particular it can simulate the correct channel behaviour, but it can also avoid any possible communication between the two parties obtaining a complete denial_of_service attack.

Since we have in principle infinitely many different ways of defining an enemy and "inserting it on the communication channel", then we need a methodology for doing that. Note that the detection of denial_of_service attacks is not interesting, as it is always possible to have such attacks if the enemy is sufficiently powerful whatever protocol we are analysing. In the following we define the notion of *correct attack*. We will see that if we consider only *correct attacks* we basically filter the possibility of a denial_of_service attack. We will also show that this filtering is useful since a denial_of_service attack could "hide" the presence of an interesting attack.

Definition 5.1 *Let $P \in \mathcal{E}_L$ and $P_E \in \mathcal{E}$ be two agents. We say that P_E/Act_H represents a* correct attack *to P iff $P_E \setminus Act_H \approx_T P$.* ■

Basically, we say that an agent P_E is a correct model of an attack on P if the high level actions represent all the enemy activity. We must be able to turn off the enemy by simply restricting the execution of P_E over all the high level actions. Instead of specifying how to construct P_E from P, we simply give a condition on the behaviour of P_E. This permits to apply SNNI in the analysis of protocols. As a matter of fact, for correct attacks, we can check if the enemy is able to interfere on the protocol execution by simply checking SNNI on P_E:

$$P_E \setminus Act_H \approx_T P_E/Act_H$$

This is equivalent to:

$$P \approx_T P_E/Act_H \tag{2}$$

So we check if the original protocol behaves like the protocol under attack (in P_E/Act_H we leave the enemy act indiscriminately).

Note that we could directly check the latter equivalence $P \approx_T P_E/Act_H$ without requiring that P_E/Act_H is a *correct attack*. In this way we would have no restrictions on the shape of P_E. In order to understand why it is useful to focus on the *correct attacks* only, we prove the following lemma which gives a simple characterization of all possible *correct attacks*:

Lemma 5.2 *The following hold:*

(i) if $P_E \setminus Act_H \approx_T P$ then $T(P) \subseteq T(P_E/Act_H)$;

(ii) if $T(P) \subseteq T(P')$ then $\exists P_E$ such that $P_E/Act_H \approx_T P'$ and $P_E \setminus Act_H \approx_T P$.

Proof. *(i)* It is trivial since $T(P) = T(P_E \setminus Act_H) \subseteq T(P_E/Act_H)$. *(ii)* Let $T(P) \subseteq T(P')$. Consider $P_E = P + fake.P'$ with $fake \in Act_H$. We have that $P_E/Act_H \approx_T P'$ and also that $P_E \setminus Act_H \approx_T P$. ■

So all the *correct attacks* P_E/Act_H on a protocol P correspond, in the trace model, to all the processes that extend the set of traces of process P. Hence we do not consider attacks where the enemy is able to forbid the execution of some traces of P, as in the complete denial_of_service attack mentioned above. In

general, if the enemy is able to prevent the execution of a certain trace, it will be able to forbid one of the possible correct executions of the protocol and so we will have a denial_of_service attack. Moreover these attacks are problematic in the protocol analysis since they could "hide" all the other (interesting) attacks. In particular, if we have a denial_of_service attack in P_E/Act_H we will have that $P \not\approx_T P_E/Act_H$ for sure. So other possibly interesting attack might be "hidden" by the presence of such an attack.

In Section 6.1, we will show that, in a particular case, the verification of $P \approx_T P_E/Act_H$ without requiring that P_E/Act_H is a *correct attack* corresponds to the verification of SNNI on *correct attacks*, plus the verification of the absence of denial_of_service attacks.

We will construct $Protocol_E$ as the parallel composition of all the parties with the channel and the enemy (as done in (1) above in this Section). Before doing this we insert some additional low level actions which are useful to observe the behaviour of the protocol in more detail. In particular we add a *signal_sent* action after every message "send" and a *signal_rcvd* one after every message "receive". This is not necessary for the last message where we already have the *signal_commit* action. For example, the Initiator becomes as follows:

$$
\begin{array}{rcl}
Initiator'(ini_id, ini_nonce) & \stackrel{\text{def}}{=} & signal_request(ini_id, res_id) \\
 & & \overline{send_msg_1}(ini_id, res_id, enc(PK_{res_id}, ini_nonce, ini_id)). \\
 & & \overline{signal_sent}(1, ini_id, res_id). \\
 & & receive_msg_2(res_id, ini_id, enc(PK_{ini_id}, ini_nonce', res_nonce)). \\
 & & \overline{signal_rcvd}(2, ini_id, res_id). \\
 & & \textbf{if } ini_nonce' = ini_nonce \textbf{ then} \\
 & & \quad \overline{send_msg_3}(ini_id, res_id, enc(PK_{res_id}, res_nonce)). \\
 & & \quad \overline{signal_commit_ini}(ini_id, res_id).0 \\
 & & \textbf{else} \\
 & & \quad \overline{signal_fail_ini}(ini_id, res_id).0
\end{array}
$$

Similarly, we obtain process $Responder'(res_id, nb)$ by adding a $\overline{signal_rcvd}(1, res_id, ini_id)$ and a $\overline{signal_sent}(2, res_id, ini_id)$ after the reception of message 1 and the sending of message 2, respectively. Since we want to focus on the "correct" execution of the protocol, we consider, for the analysis, only the actions representing the correct execution and termination of a session between A and B:

$$
\begin{array}{l}
signal_sent(1, A, B),\ signal_sent(2, B, A),\ signal_rcvd(2, A, B), \\
signal_rcvd(1, B, A),\ signal_commit_ini(A, B), signal_commit_res(B, A).
\end{array}
$$

All the other actions are hidden. We redefine *Protocol* by substituting the Initiator and the Responder with the new processes and by hiding all the actions except the ones mentioned above:

$$Protocol' \stackrel{\text{def}}{=} (Initiator'(A, na) \mid Channel \mid Responder'(B, nb)) \setminus L \ / \ M$$

where $M = \{$ *signal_request*, *signal_fail_ini*, *signal_fail_res* $\}$ and, as above, $L = \{$ *send_msg_1*, *receive_msg_1*, *send_msg_2*, *receive_msg_2*, *send_msg_3*, *receive_msg_3* $\}$ is the set of messages sent and received during the protocol. Moreover, $I_{id} = \{A\}$, $R_{id} = \{B\}$ and $Nonces = \{na, nb\}$.

Now we consider the following process representing the protocol with the enemy on the channel (we hide some actions in order to maintain the same visible actions as in *Protocol'*):

$$Protocol'_E \stackrel{\text{def}}{=} (Initiator'(A, na) \mid Channel \mid Responder'(B, nb) \mid Enemy(E, ne)) \setminus L \ / \ M'$$

where:

- $M' = M \cup \{$ *signal_sent(m,any_id, E), signal_rcvd(m,any_id, E), signal_commit_ini(A, E), signal_commit_res(B, E)* $\mid m \in \{1, 2\}$, *any_id* $\in \{A, B, E\}$ $\}$.
- *fake* $\in Act_H$

- $\{$*send_msg_1, receive_msg_1, send_msg_2, receive_msg_2, send_msg_3, receive_msg_3, signal_request, signal_sent, signal_rcvd, signal_commit_ini, signal_commit_res, signal_fail_ini, signal_fail_res*$\} \subseteq Act_L$

It is easy to see that $Protocol'_E \setminus Act_H \approx_T Protocol'$. So $Protocol'_E / Act_H$ is a *correct attack* for $Protocol'$.

The security property we automatically check is *SNNI*. The tool used for this analysis is the *Compositional Security Checker* (CoSeC), a semantic-based tool for the automatic verification of some compositional information flow properties [FG97]. In particular, using CoSeC, we check if the following holds:

$$Protocol'_E \setminus Act_H \approx_T Protocol'_E / Act_H$$

As we said above, high level actions represent the *Enemy* activity. So we can see $Protocol'_E \setminus Act_H$ as the protocol on a completely secure channel (i.e., with no enemies), and $Protocol'_E / Act_H$ as the protocol under the *Enemy* attack. If these two processes are not equivalent we can conclude that the *Enemy* is able to interfere with the correct execution of the protocol.

The automatic check gives the following answer:

```
false
Agent Protocol'_E/fake
can perform action sequence signal_rcvd(1,B,A), signal_sent(2,B,A),
        signal_commit_res(B,A)
which agent Protocol'_E\fake
cannot
```

This answer means that the high level action $fake$ (and so the enemy) interferes with the low level actions, i.e., with the correct execution of the protocol. We see that when we hide the fake action, and so we permit the enemy to operate indiscriminately, we obtain an execution sequence where B is convinced to communicate to A but A has never started the protocol with B. In particular B receives message 1 apparently from A, sends message 2 to A and finally terminates the protocol convinced that A is the other party. However, we have no messages sent or received by A. If we study the labelled transition system generated by the initial process $Protocol'$ looking for a trace with $\overline{signal_rcvd}(1,B,A)$, $\overline{signal_sent}(2,B,A)$, $\overline{signal_commit_res}(B,A)$ and without $signal_sent(m,A,B)$ and $signal_rcvd(m,A,B)$ we find out the following sequence corresponding to a specific attack:

$$\begin{array}{l} signal_request(A,E), \overline{signal_sent}(1,A,E), \overline{signal_rcvd}(1,B,A), \overline{signal_sent}(2,B,A), \\ \overline{signal_rcvd}(2,A,E), \overline{signal_commit_ini}(A,E), \overline{signal_commit_res}(B,A) \end{array}$$

We observe that A starts the protocol with E while B receives the first message apparently from A. When the execution terminates A is (correctly) convinced to communicate with E while B thinks to communicate with A. To realize the cheat, two runs have to be active at the same time: one between the users A and E, and one between the users E (the intruder that claims to be A) and B. Using α and β to indicate the two runs, the sequence that allows the attack is the following (we report at the right side the corresponding VSPA actions):

$Message\ \alpha 1.$	$A \to E$	:	$PK_E(N_A, A)$	$\overline{signal_sent}(1,A,E)$
$Message\ \beta 1.$	$E(A) \to B$	:	$PK_B(N_A, A)$	$\overline{signal_rcvd}(1,B,A)$
$Message\ \beta 2.$	$B \to E(A)$	:	$PK_A(N_A, N_B)$	$\overline{signal_sent}(2,B,A)$
$Message\ \alpha 2.$	$E \to A$	:	$PK_A(N_A, N_B)$	$\overline{signal_rcvd}(2,A,E)$
$Message\ \alpha 3.$	$A \to E$	:	$PK_E(N_B)$	$\overline{signal_commit_ini}(A,E)$
$Message\ \beta 3.$	$E(A) \to B$	:	$PK_B(N_B)$	$\overline{signal_commit_res}(B,A)$

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. A corrected version of the protocol is [Low95]:

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

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

Using again SC to analyze the corrected formalization, we can see that now the agent $Protocol'_E$ satisfies the SNNI property, so we can conclude that $Enemy$ is not able to interfere with the protocol.

6 Guaranteeing Authentication

In the previous section we have seen how it is possible to apply Non-Interference properties to detect attacks on authentication. However we have not given a definition of authentication. What we are guaranteed when applying NI properties to the analysis of protocols is simply that the behaviour of the protocol with or without the enemy is always the same (with respect to some suitable observable actions). This can be read as "the enemy is able to do nothing (good or evil) to the protocol". For this reason the NI-based analysis we propose is very general and can be applied to the verification of very different security properties. For example in [FG97] we apply this technique to the verification of the correctness of a simple key-exchange protocol. What NI-based analysis guarantees is some kind of "property preservation": all the properties based on the observable actions (low level) will be preserved also in the presence of the enemy.

In this section we want to show that this "NI guarantee" implies some of the existing authentication properties. Moreover we also show that, under some conditions, it may become exactly equivalent to such properties.

6.1 The Agreement Property

The first property we consider is *Agreement*. We quote from [Low97] an informal definition of it:

> "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 [Low97], 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) in [Low97].

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. Here we choose $\overline{signal_rcvd}(2, ini_id, res_id)$ and $\overline{signal_rcvd}(1, res_id, ini_id)$ actions to represent the running of the

protocol for the initiator and the responder respectively. [4] Such actions must be extended with the data that the two parties have to agree on. If d represent a value in the set of relevant data, we have $\overline{signal_rcvd}(2, ini_id, res_id, d)$ and $\overline{signal_rcvd}(1, res_id, ini_id, d)$ respectively. For more clarity, we will call these two actions $signal_running_ini(ini_id, res_id, d)$ and $signal_running_res(res_id, ini_id, d)$. For example, if we want to observe the running of A as initiator with B using nonces N_A and N_B, we will have the action

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

Also the commit actions must be extended with d as follows: $signal_commit_ini(ini_id, res_id, d)$, $signal_commit_res(res_id, ini_id, d)$. For example, $signal_commit_res(B, A, \{N_A, N_B\})$ represents the fact that B is convinced to communicate with A using nonces N_A and N_B.

After specifying the run and commit actions, we can define the agreement property as follows: $Agreement(ini, res, ds)$ requires that when B executes:

$$signal_commit_res(B, A, d)$$

with $d \in ds$, then A has previously executed:

$$signal_running_ini(A, B, d)$$

This means that every time B completes the protocol with A convinced that the relevant data are the ones represented by d, then A must have been running the protocol with B using exactly the data in d. In $Agreement(res, ini, ds)$ we have the symmetric situation: the commit of the initiator (A) must be preceded by the running of the responder (B). Note that in ds we specify the "range" of the relevant data. What that data represent depends on how they are inserted in actions commit and running. For example, if we want to consider the initiator and the responder nonces as relevant, we have to insert them in the running and commit actions as follows:

$$\begin{array}{l} signal_running_ini(ini_id, res_id, \{ini_nonce, res_nonce\}) \\ signal_running_res(res_id, ini_id, \{ini_nonce, res_nonce\}) \\ signal_commit_ini(ini_id, res_id, \{ini_nonce, res_nonce\}) \\ signal_commit_res(res_id, ini_id, \{ini_nonce, res_nonce\}) \end{array}$$

Note that variables ini_nonce and res_nonce are instantiated in processes $Initiator'$ and $Responder'$ exactly with the values of the initiator's and responder's nonces.

Then we must consider $ds = Nonces \times Nonces$ which represents the set in which such data should range. $Agreement(ini, res, Nonces \times Nonces)$ and $Agreement(res, ini, Nonces \times Nonces)$ have been successfully verified on the Lowe-Needham-Schroeder protocol in [Low97].

We can formalize the *Agreement* property as follows:

Definition 6.1 *A protocol P guarantees $Agreement(role1, role2, ds)$ iff $\forall D \in ds$:*

$$\begin{array}{c} P \setminus \{signal_running_role1(A1, A2, D)\} \\ \approx_T \\ P \setminus \{signal_running_role1(A1, A2, D), signal_commit_role2(A2, A1, D)\} \end{array}$$

where ($role1 = ini$, $role2 = res$) or ($role1 = res$, $role2 = ini$) and $A1$ and $B1$ are the two entities playing the roles $role1$ and $role2$, respectively. ■

We require that, if we restrict P over $signal_running_role1(A1, A2, D)$, we obtain a process which cannot execute $signal_commit_role2(A2, A1, D)$. This means that $signal_commit_role2(A2, A1, D)$ is always executed after a $signal_running_role1(A1, A2, D)$.

[4] We cannot use $\overline{signal_sent}(1, ini_id, res_id)$ to represent the running of the protocol for the initiator if we want to observe the values of the nonces, as we will do in the following. Indeed, the initiator learns the value of the responder's nonce after the reception of the second message.

Note that the agreement property does not compare the execution of the protocol with and without the presence of the enemy. We can check if a certain protocol with a certain enemy guarantees *Agreement* without considering the execution of the original protocol (with no enemy). This is the basic difference between this approach and the NI based one. However, it is implicit that the protocol without enemy guarantees *Agreement*. So what we do is to check if the protocol in the presence of a certain enemy still guarantees *Agreement*. This permits to find a relation between the two approaches. In particular we observe that, if P is a protocol and P_E/Act_H is a correct attack to P such that $P_E \in SNNI$, then all the properties of P which are based on the observation of actions in Act_L still hold for P_E. This is because $P_E \in SNNI$ implies that the enemy is not able to change the behaviour of the protocol from a low level point of view. Since $Agreement(role1, role2, ds)$ is based only on the observation of actions $signal_running_role1(A1, A2, D)$ and $signal_commit_role2(A2, A1, D)$ for all $D \in ds$, then we will focus on this pair of actions. In the following we will use the notation $P/_{(role1,role2,D)}$ to represent process P where all the low level actions except $signal_running_role1(A1, A2, D)$ and $signal_commit_role2(A2, A1, D)$ are hidden. Formally:

$$P/_{(role1,role2,D)} \stackrel{\text{def}}{=} P/(Act_L \setminus \{signal_running_role1(A1, A2, D), signal_commit_role2(A2, A1, D)\})$$

Since we have to deal with processes of the form $P/_{(role1,role2,D)}$ with $D \in ds$, for more clarity we give the following definitions:

Definition 6.2 *Let $P \in \mathcal{E}_L$ and $P_E \in \mathcal{E}$ be two agents. $(P, P_E) \in LowEq(role1, role2, ds)$ if and only if $P/_{(role1,role2,D)} \approx_T P_E/_{(role1,role2,D)}/Act_H$, for all $D \in ds$.* ■

$LowEq(role1, role2, ds)$ requires that $P/_{(role1,role2,D)}$ and $P_E/_{(role1,role2,D)}$ are equivalent from a low level point of view for all $D \in ds$ (note that $P/_{(role1,role2,D)}$ is a low level process). It corresponds to the property (2) of Section 5 ($P \approx_T P_E/Act_H$). We also extend $SNNI$ as follows:

Definition 6.3 *Let $P \in \mathcal{E}_L$ and $P_E \in \mathcal{E}$ be two agents. $P_E \in SNNI(role1, role2, ds)$ if and only if $P_E/_{(role1,role2,D)} \in SNNI$, for all $D \in ds$.* ■

We will also need a notion of correct attack which refers only to the visible actions in processes $P/_{(role1,role2,D)}$ and $P_E/_{(role1,role2,D)}$:

Definition 6.4 *Let $P \in \mathcal{E}_L$ and $P_E \in \mathcal{E}$ be two agents. We say that P_E/Act_H represents a ds-*correct attack *to P iff for all $D \in ds$, $P_E/_{(role1,role2,D)}/Act_H$ is a correct attack to $P/_{(role1,role2,D)}$, i.e., $P_E/_{(role1,role2,D)} \setminus Act_H \approx_T P/_{(role1,role2,D)}$.* ■

Note that if P_E is a *correct attack* to P then it is also a *ds-correct attack*.

We will prove a general result on the property $LowEq(role1, role2, ds)$ where we do not require that P_E/Act_H is a correct attack to P. We will give a similar result for $SNNI(role1, role2, ds)$ and correct attacks as a corollary. This general result will be useful to see that the property *LowEq* as well as *Agreement* can be misled by denial_of_service attacks.

The following holds:

Proposition 6.5 *Let Let $P \in \mathcal{E}_L$ and $P_E \in \mathcal{E}$ be two agents. If $P \in Agreement(role1, role2, ds)$ and $(P, P_E) \in LowEq(role1, role2, ds)$, then:*

$$P_E \in Agreement(role1, role2, ds)$$

PROOF. We will use *run* and *commit* to represent actions $signal_running_role1(A1, A2, D)$ and $signal_commit_role2(A2, A1, D)$ respectively.

For a generic $D \in ds$, we will prove that $P/_{(role1,role2,D)} \approx_T P_E/_{(role1,role2,D)}/Act_H$ implies $P_E \setminus \{run\} \approx_T P \setminus \{run, commit\}$.

If $P \in Agreement(role1, role2, ds)$ then we have that $P \setminus \{run\} \approx_T P \setminus \{run, commit\}$. So $P \setminus \{run\}$ is not able to execute a *commit* action. By definition of $P/_{(role1,role2,D)}$ we have that

$P_E/_{(role1,role2,D)}/Act_H \stackrel{lts}{=} P_E/(Act \setminus \{run, commit\})$. Moreover, $P/_{(role1,role2,D)} \stackrel{def}{=} P/(Act_L \setminus \{run, commit\})$. Now, since $P/_{(role1,role2,D)} \approx_T P_E/_{(role1,role2,D)}/Act_H$ then $P_E/(Act \setminus \{run, commit\}) \approx_T P/(Act_L \setminus \{run, commit\})$. Note that $P/(Act_L \setminus \{run, commit\}) \approx_T P/(Act \setminus \{run, commit\})$ and so $P_E/(Act \setminus \{run, commit\}) \approx_T P/(Act \setminus \{run, commit\})$. Since $P \setminus \{run\} \approx_T P \setminus \{run, commit\}$, we also have that $P \setminus \{run\}/(Act \setminus \{run, commit\}) \approx_T P \setminus \{run, commit\}/(Act \setminus \{run, commit\})$. Note that $P \setminus \{run\}/(Act \setminus \{run, commit\}) \stackrel{lts}{=} P/(Act \setminus \{run, commit\}) \setminus \{run\}$, since $(Act \setminus \{run, commit\})$ and $\{run\}$ are disjoint. Moreover $P \setminus \{run, commit\}/(Act \setminus \{run, commit\}) \stackrel{lts}{=} \underline{0}$. We obtain that $P/(Act \setminus \{run, commit\}) \setminus \{run\} \approx_T \underline{0}$ and so $P_E/(Act \setminus \{run, commit\}) \setminus \{run\} \approx_T \underline{0}$. This means that also in process P_E no *commit* action is possible when we restrict the *run* one. Hence $P_E \setminus \{run\} \approx_T P \setminus \{run, commit\}$, i.e., $P_E \in Agreement(role1, role2, ds)$. ■

For ds-correct attacks we have the following result:

Corollary 6.6 *Let $P \in \mathcal{E}_L$ and $P_E \in \mathcal{E}$ such that P_E/Act_H is a ds-correct attack for P. If $P \in Agreement(role1, role2, ds)$ and $P_E \in SNNI(role1, role2, ds)$ then*

$$P_E \in Agreement(role1, role2, ds)$$

PROOF. Trivial by Proposition 6.5 since $(P_E/_{(role1,role2,D)}) \setminus Act_H \approx_T P/_{(role1,role2,D)}$. ■

This Corollary states that for ds-correct attacks, $SNNI$ guarantees the preservation of the *Agreement* property, i.e., it guarantees that if the original protocol satisfies *Agreement* then also the ds-correct attack does.

Note that in the protocol analysis of Section 5 we had

$$signal_sent(1, A, B),\ signal_sent(2, B, A),\ signal_rcvd(2, A, B),$$
$$signal_rcvd(1, B, A),\ signal_commit_ini(A, B), signal_commit_res(B, A)$$

as observable actions. If we assume that $signal_running_ini(A, B) = signal_sent(1, A, B)$ and $signal_running_res(B, A) = signal_rcvd(1, B, A)$ then by Corollary 6.6 and Proposition 2.5 we obtain that $Protocol'_E$ satisfies both $Agreement(init, resp)$ and $Agreement(resp, init)$. [5] We do not observe the value of the nonces and so we are not guaranteed that the two parties agree on them. In order to obtain agreement on nonces we have to refine the specification by adding such values in the run and commit actions. If we call $Protocol''_E$ this new specification we obtain again that $Protocol''_E \in SNNI$ and so $Protocol''_E \in Agreement(init, resp, Nonces \times Nonces) \cap Agreement(resp, init, Nonces \times Nonces)$.

The above shows that *SNNI* can be used to prove *Agreement*. Now we want to understand what happens when *SNNI* does not hold. In that case we can say that the enemy is able to modify the behaviour of the protocol with respect to the actions we are observing (Act_L), even if we cannot in general conclude that we have an attack to the protocol as the modification of the behaviour could be not dangerous. However, we will see that under some conditions $SNNI(role1, role2, ds)$ becomes equivalent to *Agreement*. Again, we start with a general result on $LowEq(role1, role2, ds)$ property (with no conditions on P_E).

Proposition 6.7 *Let $P \in \mathcal{E}_L$ and $P_E \in \mathcal{E}$ be two agents. If the following three hold*

(*i*) $P, P_E \in Agreement(role1, role2, ds)$,

(*ii*) $\forall D \in ds : P \not\approx_T P \setminus \{signal_commit_role2(A2, A1, D)\}$,

(*iii*) $\forall D \in ds : P_E/Act_H \not\approx_T (P_E/Act_H) \setminus \{signal_commit_role2(A2, A1, D)\}$

[5] These are particular forms of *Agreement* where we do not have a set of relevant data. We simply check that $signal_commit_role2(A2, A1)$ is always executed after a $signal_running_role1(A1, A2)$. Note that such actions are not extended with the relevant data D.

then

$$(P, P_E) \in LowEq(role1, role2, ds)$$

PROOF. Again, we use *run* and *commit* as shorthands for $signal_running_role1(A1, A2, D)$ and $signal_commit_role2(A2, A1, D)$ respectively.

If $P, P_E \in Agreement(role1, role2, ds)$ then we have that $P \setminus \{run\} \approx_T P \setminus \{run, commit\}$ and $P_E \setminus \{run\} \approx_T P_E \setminus \{run, commit\}$. Hence, $T(P/(Act_L \setminus \{run, commit\})), T(P_E/(Act \setminus \{run, commit\})) \subseteq T(run.commit.0)$.

Moreover we have that $P \not\approx_T P \setminus \{commit\}$ and $P_E \not\approx_T P_E \setminus \{commit\}$. This means that both in P and in P_E the action *commit* can be executed. Hence $T(P/(Act_L \setminus \{run, commit\})) = T(P_E/(Act \setminus \{run, commit\})) = T(run.commit.0)$.

So $P/(Act_L \setminus \{run, commit\}) \approx_T P_E/(Act \setminus \{run, commit\})$. ■

So if $(P, P_E) \notin LowEq(role1, role2, ds)$, we can have different situations: (i) is not true, so either P or P_E does not satisfy $Agreement(role1, role2, ds)$. The former case is quite unnatural since we certainly want that the protocol with no enemies is completed correctly only if the other party has run the protocol. Probably we have not specified correctly the protocol. The latter should represent an attack of the enemy to the protocol. If (ii) does not hold then P is not able to execute some $signal_commit_role2(A2, A1, D)$ actions with $D \in ds' \subseteq ds$. In fact, we could have some value in ds for which the protocol is able to commit only in the presence of the enemy (condition (iii)). In this case, however, we can consider a process

$$P' \stackrel{\text{def}}{=} P + \sum_{D \in ds'} signal_running_role1(A1, A2, D).signal_commit_role2(A2, A1, D)$$

instead of P where we add a "correct" behaviour for the missing commits. In the last case (iii) we have that P_E is not able to terminate correctly for $A2$ for some values in ds. We have two sub-cases: $(iii).a$ If also the original protocol P is not able to commit for such values then we have considered a set ds which is too large. It makes no sense that we consider values for which also the original protocol P is not able to commit. $(iii).b$ Otherwise, this represents a denial_of_service attack by the enemy since the original protocol was indeed able to commit on such values.

Note that if a protocol P_E cannot execute a $signal_commit_role2(A2, A1, D)$ then it will certainly satisfy $Agreement(role1, role2, \{D\})$. In other words the fact that *Agreement* property holds for a certain protocol P_E is not interesting if we do not verify that the enemy is not able of a complete denial_of_service attack.

This confirms the fact that if $(P, P_E) \notin LowEq(role1, role2, ds)$ then we do not necessarily have an "interesting" attack. In this case it could also mean that we have a denial_of_service attack which is misleading the analysis. We have the following characterization of *Agreement*:

Corollary 6.8 *Let $P \in \mathcal{E}_L$ and $P_E \in \mathcal{E}$ be two agents. If the following three hold*

(i) $P \in Agreement(role1, role2, ds)$,

(ii) $\forall D \in ds : P \not\approx_T P \setminus \{signal_commit_role2(A2, A1, D)\}$,

(iii) $\forall D \in ds : P_E/Act_H \not\approx_T (P_E/Act_H) \setminus \{signal_commit_role2(A2, A1, D)\}$

then

$$(P, P_E) \in LowEq(role1, role2, ds) \Leftrightarrow P_E \in Agreement(role1, role2, ds)$$

PROOF. Trivial by Propositions 6.5 and 6.7. ■

We want to give a similar result for correct attacks. We need the following lemma:

Lemma 6.9 *Let $P \in \mathcal{E}_L$ and $P_E \in \mathcal{E}$ such that P_E/Act_H is a ds-correct attack for P. For all $D \in ds$ if $P \not\approx_T P \setminus \{signal_commit_role2(A2, A1, D)\}$ then*

$$P_E/Act_H \not\approx_T (P_E/Act_H) \setminus \{signal_commit_role2(A2, A1, D)\}$$

PROOF. Again, we use *run* and *commit* as shorthands for $signal_running_role1(A1, A2, D)$ and $signal_commit_role2(A2, A1, D)$ respectively.

Since P_E/Act_H is a ds-correct attack for P then $P_E/_{(role1,role2,D)} \setminus Act_H \approx_T P/_{(role1,role2,D)}$. This implies $T(P/_{(role1,role2,D)}) \subseteq T(P_E/_{(role1,role2,D)}/Act_H)$. So if P can execute *commit* then also P_E/Act_H can. ■

This lemma basically states that ds-correct attacks are not denial_of_service attacks. We give the following characterization of *Agreement* with respect to correct attacks:

Corollary 6.10 *Let $P \in \mathcal{E}_L$ and $P_E \in \mathcal{E}$ such that P_E/Act_H is a ds-correct attack for P. If*

(i) $P \in Agreement(role1, role2, ds)$, *and*

(ii) $\forall D \in ds : P \not\approx_T P \setminus \{signal_commit_role2(A2, A1, D)\}$,

then

$$P_E \in SNNI(role1, role2, ds) \Leftrightarrow P_E \in Agreement(role1, role2, ds)$$

PROOF. Trivial by Lemma 6.9 and Corollary 6.8. ■

We finally prove that the exclusion of "non-correct" attacks corresponds exactly to the exclusion of denial_of_service attacks. This allows us to focus only on correct attacks and so to use Corollary 6.10. We first observe that we could have "non-correct" attacks for which there exists an equivalent correct attack. For example, consider $P_E \stackrel{\text{def}}{=} fake.P$ with $P \not\approx_T \underline{0}$. We have that $P_E \setminus Act_H \approx_T \underline{0}$ and so P_E/Act_H is not a correct attack. However we can define a process $P'_E \stackrel{\text{def}}{=} P + fake.P$ which is such that P'_E/Act_H is correct (since $P'_E \setminus Act_H \approx_T P$) and $P'/Act_H \approx_T P/Act_H$. Since P_E/Act_H is not correct we cannot apply directly SNNI, we must instead check if $P \approx_T P_E/Act_H$. However this is equivalent to check if $P'_E \in SNNI$. So P_E is implicitly checked when we consider only correct attacks.

In order to consider also these processes which are equivalent to correct attacks, we simply have to consider the closure of the set of all correct attacks with respect to $\approx_T$. By Lemma 5.2 we obtain that such closure is exactly equal to the set of all processes P' such that $T(P) \subseteq T(P')$. For this reason, we define a "non-correct" attack as follows:

Definition 6.11 *Let $P, P' \in \mathcal{E}_L$ be two agents. We say that P' represents a* non-correct attack *to P iff $T(P') \subset T(P)$. We also say that P' represents a* non-ds-correct attack *to P iff $\exists D \in ds$ such that $P'/_{(role1,role2,D)}$ is a non-correct attack to $P/_{(role1,role2,D)}$, i.e., $T(P'/_{(role1,role2,D)}) \subset T(P/_{(role1,role2,D)})$.* ■

Now we prove that non-correct attacks correspond to denial_of_service attacks:

Proposition 6.12 *Let $P \in \mathcal{E}_L$ and $P_E \in \mathcal{E}$ such that P_E/Act_H is a non-ds-correct attack for P. If*

(i) $P \in Agreement(role1, role2, ds)$, *and*

(ii) $\forall D \in ds : P \not\approx_T P \setminus \{signal_commit_role2(A2, A1, D)\}$,

then

$$\exists D \in ds : P_E/Act_H \approx_T (P_E/Act_H) \setminus \{signal_commit_role2(A2, A1, D)\}$$

PROOF. Again, we use *run* and *commit* as shorthands for $signal_running_role1(A1, A2, D)$ and $signal_commit_role2(A2, A1, D)$ respectively.

If P_E/Act_H is a *non-ds-correct attack* to P then $\exists D \in ds$ such that $T(P_E/_{(role1,role2,D)}/Act_H) \subset T(P/_{(role1,role2,D)})$. By (i) and (ii) we obtain that process $P/_{(role1,role2,D)}$ is equivalent to process $run.\underline{0} + run.commit.\underline{0}$. Since $T(P_E/Act_H/_{(role1,role2,D)}) \subset T(P/_{(role1,role2,D)})$ then we have that process $P_E/_{(role1,role2,D)}/Act_H$ cannot execute *commit* action. Indeed, if $P_E/_{(role1,role2,D)}/Act_H$ would be able to execute *commit* then it will also be able to execute *run* and *run.commit* and we will obtain that $T(P_E/Act_H/_{(role1,role2,D)}) = T(P/_{(role1,role2,D)})$. Hence we finally obtain that $P_E/Act_H \approx_T P_E/Act_H \setminus \{signal_commit_role2(A2, A1, D)\}$. ■

6.2 Message-Oriented Authentication

Now, we consider the message-based approach to authentication defined in [Sch97, Sch96]. The idea is to observe when a set of messages T authenticates another set of messages R. Informally, T authenticates R if the occurrence of some element of T implies the occurrence of some element of R. T and R have to be disjoint. We formalize this property as follows:

Definition 6.13 *Let $P \in \mathcal{E}_L$. P satisfies T* **authenticates** *R iff $P \setminus R \approx_T P \setminus R \setminus T$.* ■

In other words, when we do not allow the execution of actions in R we obtain a system which is not able to execute actions in T. So, actions in T must be preceded by actions in R.

Note that $Agreement(role1, role2, \{D\})$ is equivalent to T **authenticates** R when we choose $R = \{signal_running_role1(A1, A2, D)\}$ and $T = \{signal_commit_role2(A2, A1, D)\}$. So, if T and R are single actions all the results obtained in the section above still hold.

We have a first result, which is similar to Corollary 6.6, based on the fact that if a protocol satisfies *SNNI* then all the properties based on low level actions are preserved in the presence of the enemy.

In the following we will use the notation $P/_{R \cup T}$ to represent process P where all the low level actions except the ones in $R \cup T$ are hidden. Formally:

$$P/_{R \cup T} \stackrel{\text{def}}{=} P/(Act_L \setminus (R \cup T))$$

Proposition 6.14 *Let $R \cup T \subseteq Act_L$, $P \in \mathcal{E}_L$ and $P_E \in \mathcal{E}$ such that $P_E/_{R \cup T}/Act_H$ is a correct attack for $P/_{R \cup T}$. If P satisfies T* **authenticates** *R then*

$$P_E/_{R \cup T} \in SNNI \text{ implies } P_E \text{ satisfies } T \textbf{ authenticates } R$$

Proof. We have that $P \setminus R \approx_T P \setminus R \setminus T$. Since $\approx_T$ is a congruence w.r.t. hiding we obtain that $P \setminus R/(Act \setminus R \cup T) \approx_T P \setminus R \setminus T/(Act \setminus R \cup T) \approx_T \underline{0}$. Since $Act \setminus R \cup T$ and R are disjoint we have that $P/(Act \setminus R \cup T) \setminus R \approx_T P \setminus R/(Act \setminus R \cup T) \approx_T \underline{0}$. Since $P_E/_{R \cup T} \in SNNI$ and $P_E/_{R \cup T}/Act_H$ is a correct attack for $P/_{R \cup T}$, then we have that $P_E/(Act \setminus R \cup T) \approx_T P/(Act \setminus R \cup T)$. So we obtain $P_E/(Act \setminus R \cup T) \setminus R \approx_T \underline{0}$. This means that in process P_E, if we restrict over R then we cannot execute any action in T, i.e., $P_E \setminus R$ is not able to execute actions in T. As a matter of fact, if $P_E \setminus R$ can execute an action $t \in T$ then also $P_E/(Act \setminus R \cup T) \setminus R$ can. So we obtain the thesis: $P_E \setminus R \approx_T P_E \setminus R \setminus T$ ■

So, we can use *SNNI* to prove that P_E satisfies T **authenticates** R. For the *Agreement* property we have found some conditions which make it equivalent to *SNNI*. Here we cannot find a similar result. First of all, we have a technical problem. Suppose that both P and P_E satisfy T **authenticates** R. So, both in P and in P_E the actions in T are not executable when we do not allow the execution of the actions in R. However this does not imply that the execution traces composed by actions in $R \cup T \subseteq Act_L$ are the same for the two processes P and P_E. For example the enemy could cause a new "correct" (with all the actions in T preceded by some action in R) behaviour of the protocol which is not possible in P. In this case we have that P and P_E are not equivalent from a low level point of view and so *SNNI* does not hold.

There is also a more crucial point. When we consider $R \cup T \subseteq Act_L$ we loose the information about which actions are in R and which in T. So we do not know which actions should authenticate the others.

This definitely leads us to look for a different approach. We need to consider R and T as distinct sets with different roles. In fact, we want that, when we do not allow the execution of actions in R, something on the execution of actions in T is verified. In particular we want that no actions in T can be executed anymore. We have the following result:

Proposition 6.15 *Let $P \in \mathcal{E}_L$, $P_E \in \mathcal{E}$ and $R \cup T \subseteq Act_L$. Then*

$$P_E \textit{ satisfies } T \textbf{ authenticates } R \textit{ implies } P_E/_{R \cup T} \setminus R \in SNNI$$

PROOF. If P_E satisfies T **authenticates** R then $P_E \setminus R \approx_T P_E \setminus R \setminus T$. Hence $P_E \setminus R$ is not able to execute actions in T. So $P_E/_{R \cup T} \setminus R \in \mathcal{E}_H$ which is trivially in $SNNI$. ■

Note that we restrict $P_E/_{R \cup T}$ over set R. In this way we observe the effect of the enemy on the actions in T when we do not allow the execution of the actions in R. We also have the opposite implication with the additional condition that P satisfies T **authenticates** R and that $P_E/_{R \cup T}/Act_H$ is a correct attack for P:

Proposition 6.16 *Let $R \cup T \subseteq Act_L$, $P \in \mathcal{E}_L$ and $P_E \in \mathcal{E}$ such that $P_E/_{R \cup T}/Act_H$ is a correct attack for $P/_{R \cup T}$. If P satisfies T **authenticates** R then*

$$P_E/_{R \cup T} \setminus R \in SNNI \textit{ implies } P_E \textit{ satisfies } T \textbf{ authenticates } R$$

PROOF. If P satisfies T **authenticates** R then $P \setminus R \approx_T P \setminus R \setminus T$. Moreover, $P_E/_{R \cup T} \setminus R \in SNNI$ means that $P_E/(Act_L \setminus (R \cup T)) \setminus R \setminus Act_H \approx_T P_E/(Act_L \setminus (R \cup T)) \setminus R/Act_H$. Note that $P_E/(Act_L \setminus (R \cup T)) \setminus R \setminus Act_H \stackrel{\text{lts}}{=} P_E/(Act_L \setminus (R \cup T)) \setminus Act_H \setminus R \approx_T P/(Act_L \setminus (R \cup T)) \setminus R$ since $P_E/_{R \cup T}/Act_H$ is a correct attack for $P/_{R \cup T}$. Now, $P/(Act_L \setminus (R \cup T)) \setminus R \stackrel{\text{lts}}{=} P \setminus R/(Act_L \setminus (R \cup T)) \approx_T P \setminus R \setminus T/(Act_L \setminus (R \cup T)) \approx_T \underline{0}$ since $P \setminus R \approx_T P \setminus R \setminus T$ and $\approx_T$ is a congruence with respect to hiding. So we obtain that $P_E/(Act_L \setminus (R \cup T)) \setminus R/Act_H \approx_T \underline{0}$. Since $P_E/(Act_L \setminus (R \cup T)) \setminus R/Act_H \stackrel{\text{lts}}{=} P_E \setminus R/(Act \setminus (R \cup T))$ we have that $P_E \setminus R/(Act \setminus (R \cup T)) \approx_T \underline{0}$. This means that when we restrict set R in P_E no actions in T can be executed. Hence $P_E \setminus R \approx_T P_E \setminus R \setminus T$. ■

So, we have the following characterization:

Corollary 6.17 *Let $R \cup T \subseteq Act_L$, $P \in \mathcal{E}_L$ and $P_E \in \mathcal{E}$ such that $P_E/_{R \cup T}/Act_H$ is a correct attack for $P/_{R \cup T}$. If P satisfies T **authenticates** R then*

$$P_E/_{R \cup T} \setminus R \in SNNI \Leftrightarrow P_E \textit{ satisfies } T \textbf{ authenticates } R$$

■

6.3 Strand Spaces Technique

In [FHG98] it is defined an interesting technique to analyze authentication protocols based on the idea of *strand spaces*. A *strand* is a sequence of events that a participant may be engaged in, i.e., is a sequence of message sent and received. So the model used is not very different from ours. We have that each participant is represented by the set of traces (s)he can execute too. However, in [FHG98] it is developed a proof technique that permits to find the exact conditions under which the protocol is correct.

For example, one of the interesting results presented is that the Lowe-Needham-Schroeder protocol guarantees authentication of the initiator to the responder even if the key of the responder has been compromised. We could deal with this kind of verification by considering a more powerful enemy that knows the responder key. Then we could automatically check that the protocol is secure also in this case, but we have no way of automatically find the minimal conditions for the protocol to be correct. In [FHG98], the authors write a correctness proof and so they can find out which hypothesis are needed for the protocol correctness, i.e., for completing the proof.

The property they prove is similar to the *Agreement* one, but, for the responder, they obtain a result that is in general stronger than the guarantees given by *Agreement*. In particular they prove that if the responder has correctly terminated the protocol then also the initiator has. This is different from proving that when the responder terminates then the initiator has at least run the protocol. However this depends on how we observe that the initiator is running the protocol. For example in the analysis we have done, the $\overline{signal_rcvd}(2, A, B)$ is executed by the initiator before the test on the received nonce. So we do not know if A will commit after the execution of that signal. However if we move that action after the test (exactly before the third message is sent), we will be assured that, if A executes such an action, then she will correctly terminate the protocol. Indeed, she cannot fail sending the last message. In the original protocol (with no enemy) such an action always precedes the commit of the responder; thus, whenever the responder commits eventually the initiator does the same. This behaviour is obviously preserved in every *correct attack* which is *SNNI*.

7 Conclusion

In this paper we have shown how NI properties can be profitably applied in the verification of an authentication protocol. In general, what we are guaranteed when applying NI properties to the analysis of protocols is simply that the behaviour of the protocol with and without the enemy is always the same (with respect to some suitable observable actions). This can be read as "the enemy is able to do nothing (good or evil) to the protocol". What NI-based analysis guarantees is some kind of "property preservation": all the properties based on the observable actions (low level) will be preserved also in the presence of the enemy.

We have successfully applied the same methodology presented in this paper to the verification of other authentication protocols. As an example, consider the following simple symmetric-key authentication protocol studied in [BGH+93]:

$$\begin{array}{llcl} Message\ 1. & A \rightarrow B & : & E(N1) \\ Message\ 2. & B \rightarrow A & : & N1, E(N2) \\ Message\ 3. & A \rightarrow B & : & N2 \end{array}$$

where $E(M)$ represents the encryption of M through a key which is shared between A and B. A sends a nonce $N1$ encrypted with E. Since only A and B are able to decrypt $E(N1)$, when A receives $N1$ she will believe to talk with B. The same happens in the third message when A sends $N2$ back to B.

Suppose to model this protocol in SPA allowing A and B to act both as initiator and as responder. We also add the same enemy as for the Needham-Schroeder protocol and the same visible actions. Then, we obtain that the resulting process in not SNNI. The automatic check gives exactly the same answer as for the Needham-Schroeder protocol: B is running the protocol as a responder apparently with A but A has never sent any messages to B as initiator. We find out the following specific trace which represent an attack:

$$\overline{signal_rcvd}(1, B, A), \overline{signal_sent}(2, B, A), \overline{signal_rcvd}(1, A, B),$$
$$\overline{signal_sent}(2, A, B), \overline{signal_commit_res}(B, A)$$

We see that A is acting as a responder in a session apparently with B. We can rewrite the attack as follows:

$$\begin{array}{llcll} Message\ \alpha 1. & E(A) \rightarrow B & : & E(N1) & \overline{signal_rcvd}(1, B, A) \\ Message\ \alpha 2. & B \rightarrow E(A) & : & N1, E(N2) & \overline{signal_sent}(2, B, A) \\ Message\ \beta 1. & E(B) \rightarrow A & : & E(N2) & \overline{signal_rcvd}(1, A, B) \\ Message\ \beta 2. & A \rightarrow E(B) & : & N2, E(N3) & \overline{signal_sent}(2, A, B) \\ Message\ \alpha 3. & E(A) \rightarrow B & : & N2 & \overline{signal_commit_res}(B, A) \end{array}$$

The enemy starts a session α with B where he pretends to be A. [6] Then he pretends to be B and starts

[6] The encrypted nonce $E(N1)$ could have been intercepted in an old session between A and B. It could also be a generic sequence of bit, since its decryption ($N1$) is not used anymore in the protocol.

another session β with A using her as an oracle to obtain $N2$. In this way E is able to conclude the session α. At the end B is convinced to communicate with initiator A, while A never started a session as initiator with B.

A protocol similar to this one is the ISO SC27 [ISO90]:

$$\begin{array}{lllll} Message\ 1. & A \rightarrow B & : & N1 \\ Message\ 2. & B \rightarrow A & : & E(N1), E(N2) \\ Message\ 3. & A \rightarrow B & : & N2 \end{array}$$

Here we find out the following more subtle attack [ISO90]:

$$\begin{array}{lllll|l} Message\ \alpha 1. & A \rightarrow E(B) & : & N1 & \overline{signal_sent}(1, A, B) \\ Message\ \beta 1. & E(B) \rightarrow A & : & N1 & \overline{signal_rcvd}(1, A, B) \\ Message\ \beta 2. & A \rightarrow E(B) & : & E(N1), E(N2) & \overline{signal_sent}(2, A, B) \\ Message\ \alpha 2. & E(B) \rightarrow A & : & E(N1), E(N2) & \overline{signal_rcvd}(2, A, B) \\ Message\ \alpha 3. & A \rightarrow E(B) & : & N2 & \overline{signal_commit_ini}(A, B) \\ Message\ \beta 3. & E(B) \rightarrow A & : & N2 & \overline{signal_commit_res}(A, B) \end{array}$$

A starts a session α with the enemy, believing E is B. Then the enemy start a parallel session with A where he pretends again to be B. In this way we obtain two parallel sessions between A and the enemy which pretends to be B. In the two sessions the roles of A and $E(B)$ are different. in α, A is the initiator and $E(B)$ is the responder. In β we have the opposite situation. In this "parallel session" attack the enemy is able to re-send all the values received by A. At the end, both the sessions are completed successfully and so we have a double connection between A and E who pretends to be B (corresponding to the two commit actions).

It is clear that the sequence of actions reported on the right side is not possible without an enemy (there are no messages from B).

Since NI technique is very general, it can be applied to the verification of very different security properties. In [FG97], we show how to use exactly the same NI-based property and the same kind of analysis to check a simple session key distribution protocol based on public-key.

$$\begin{array}{lllll} Message\ 1. & A \rightarrow KDC & : & A, B \\ Message\ 2. & KDC \rightarrow A & : & A, SK_{KDC}(PK_B, B) \\ Message\ 3. & A \rightarrow B & : & A, B, PK_B(K_{sess}) \end{array}$$

When a user A needs to send some secret information to another user B, A asks KDC (Key Distribution Center) for B's public key. KDC replies with B's public key and identifier both signed with its secret key. Then, A sends to B a session-key encoded with the public key of B. Such session key will be used for every further communication, until the two users decide to establish a new session with a new session-key

In [FG97], we show how NI-based properties are able to detect the following two attacks:

- if the KDC does not sign the requested public keys (the second message is PK_B, B instead of $SK_{KDC}(PK_B, B)$), then an attacker can substitute those keys with its own key;
- if the KDC forgets to include in the signed message the identity of the owner of the key (the second message is $SK_{KDC}(PK_B)$ instead of $SK_{KDC}(PK_B, B)$), then an attacker can fake the request from a user to KDC in such a way that the user receives the enemy key.

Acknowledgements

We would like to thank Anna Ghelli for her preliminary experimentations of the Needham-Schroeder protocol with CoSeC and Celso Belli for useful comments.

References

[BGH+93] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. "Systematic Design of a Family of Attack Resistant Authentication Protocols". *IEEE Journal on Selected Areas in Communications*, 11(5):679–693, 1993.

[FG95] R. Focardi and R. Gorrieri. "A Classification of Security Properties for Process Algebras". *Journal of Computer Security*, 3(1):5–33, 1994/1995.

[FG97] 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.

[FHG98] F. J. T. Fabrega, J. C. Herzog, and J. D. Guttman. "Strand Spaces: Why is a Security Protocol Correct?". In *Proceedings of the 1998 Symposium on Security and Privacy*. IEEE Computer Society Press, May 1998.

[GM82] 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.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[ISO90] "Entity Authentication Using Symmetric Techniques". Report ISO/IEC JTC1.27.02.2 (20.03.1.2), June 1990.

[Low95] G. Lowe. "An attack on the Needham-Schroeder public-key authentication protocol". *Information Processing Letters*, 53:97–101, 1995.

[Low96] 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.

[Low97] G. Lowe. "A Hierarchy of Authentication Specification". In *Proceedings of the 10th Computer Security Foundation Workshop*. IEEE press, 1997.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[NS78] R. Needham and M. Schroeder. "Using encryption for authentication in large networks of computers". *Communications of the ACM*, 21(12):993–999, 1978.

[Sch96] S. Schneider. "Security Properties and CSP". In *Proceedings of the 1996 Symposium on Security and Privacy*. IEEE Computer Society Press, May 1996.

[Sch97] S. Schneider. "Verifying authentication protocols with CSP". In *Proceedings of the 10th Computer Security Foundation Workshop*. IEEE press, 1997.