URL: http://www.elsevier.nl/locate/entcs/volume32.html 12 pages

Secrecy in Security Protocols as Non Interference

R. Focardi [1]

Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italy

R. Gorrieri [2]

Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy.

F. Martinelli [3]

Dipartimento di Informatica, Università di Pisa, Italy.

Abstract

Non Interference [8] has been proposed for modelling and analysing information flow in systems. In [4,7] we have indeed shown that the Non Interference property called NDC can be applied also in the area of network security, for the analysis of typical cryptographic protocol properties (e.g., authentication, non-repudiation). In this paper we extend the results of [4,7] by showing that NDC can be also easily adapted to detect secrecy attacks over networks.

1 Introduction

Secrecy is one of the main issues in security. In general, a system (or a protocol) preserves the *secrecy* of a set of data if it guarantees that *non-authorized* users/entities never gain access to such data.

In [5,6] a non interference property called *Non Deducibility on Compositions* (NDC) has been proposed for the detection of information flows inside systems. It is a strong property which guarantees that no information flow is possible from a set of *high level* users (representing the authorized users) to the set of *low level* ones (who are not authorized to access secret data).

[1] Partially supported by MURST Progetto "Certificazione automatica di programmi mediante interpretazione astratta"

[2] Partially supported by MURST Progetto "Teoria della Concorrenza, Linguaggi di Ordine Superiore e Strutture di Tipi."

[3] Partially supported by CSP Progetto "ISA: Internet Security trAnsactions".

In this paper we show how NDC can be easily adapted for analysing secrecy in networks. Indeed, the main aim of our current research is to find a uniform approach for defining the many variants of security properties in such a way that they can be all seen as specific instances of a general NDC-based scheme (called GNDC [7]). This is badly needed in order to compare, classify and evaluate the merits of the various definitions and, possibly, to provide general, effective proof techniques that can be applied suitably for all properties.

To this aim, in [7] we have presented a process algebra, called CryptoSPA (in turn an improvement of SPA [6] which borrows some concepts from the language defined in [13]), that is expressive enough to model a large class of systems, e.g., (non mobile) security protocols. CryptoSPA has been chosen as the common model for comparing the various properties through the general, unifying scheme. The idea behind is essentially *non interference*, proposed many years ago [8] in a completely different context to study information flow in computer systems and widely studied in [5,6,14]. Roughly, a system is secure if its behaviour cannot be significantly altered (hence, no interference is possible) when executed in a hostile environment. This property is a direct generalization of NDC [5,6].

Some security properties (e.g., authentication as in [10,17] and denial of service as in [16]) have been shown as instances of our general scheme in [7]. The main goal of this paper is to show that also secrecy can be easily defined in our NDC-based framework. This is interesting as it strengthens our claim that non interference plays an important role in the specification and analysis of security protocols. Indeed, non interference seems the strongest property that can be defined for cryptographic protocols (see also [7]). Moreover, it shows that non interference, originally proposed for detecting information flows in systems, is also profitable for revealing secrecy attacks in network security.

The paper is organized as follows: in Section 2 we define the model; in Section 3 we define secrecy as a NDC-based property; Section 4 reports a simple example; finally, Section 5 is about future work.

2 The Model

In this section we report from [7] the language we use for the specification of authentication properties and protocols. It is called *Cryptographic Security Process Algebra* (CryptoSPA for short), and it is basically a variant of value-passing CCS [15], where the processes are provided with some primitives for manipulating messages. In particular, processes can perform message encryption and decryption, and also construct complex messages by composing together simpler ones.

2.1 The CryptoSPA Syntax

CryptoSPA syntax is based on the following elements:

- A set $I = \{a, b, \ldots\}$ of *input* channels, a set $O = \{\bar{a}, \bar{b}, \ldots\}$ of *output* ones;
- A set M of basic messages and a set K of encryption keys with a function $\cdot^{-1} : K \to K$ such that $(k^{-1})^{-1} = k$. The set $\mathcal{M}$ of all messages is defined as the least set such that $M \cup K \in \mathcal{M}$ and $\forall m \in \mathcal{M}$, $\forall k \in K$ we have that (m, m') and $\{m\}_k$ also belong to $\mathcal{M}$;
- A family $\mathcal{U}$ of sets of messages and a function $Msg(c) : I \cup O \longrightarrow \mathcal{U}$ which maps every channel c into the set of possible messages that can be sent and received on such channel. Msg is such that $Msg(c) = Msg(\bar{c})$.
- A set C of *public* channels; these channels represent the insecure network where the enemy can intercept and fake messages;
- A set $Act = \{c(m) \mid c \in I, m \in Msg(c)\} \cup \{\overline{c}\, m \mid c \in O, m \in Msg(c)\} \cup \{\tau\}$ of actions (τ is the internal, invisible action), ranged over by a; we also have a function $chan(a)$ which returns c if a is either $c(m)$ or $\overline{c}\, m$, and the special channel $void$ when $a = \tau$; we assume that $void$ is never used within a restriction operator (see below).
- A set $Const$ of constants, ranged over by A.

The syntax of CryptoSPA agents is defined as follows:

$$\begin{aligned} E ::= \underline{0} \;\big|\; c(x).E \;\big|\; \overline{c}\, e.E \;\big|\; \tau.E \;\big|\; E + E \;\big|\; E \parallel E \;\big|\; E \setminus L \;\big| \\ \big|\; A(m_1, \ldots, m_n) \;\big|\; [e = e']E; E \;\big|\; [\langle e_1 \ldots e_r \rangle \vdash_{rule} x]E; E \end{aligned}$$

where x is a variable, $m_1, \ldots, m_n$ are messages, $e, e_1, \ldots, e_r$ are messages (possibly containing variables) and L is a set of input channels. Both the operators $c(x).E$ and $[\langle e_1 \ldots e_r \rangle \vdash_{rule} x]E; E'$ bind the variable x in E. It is also necessary to define constants as follows: $A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} E$ where E is a CryptoSPA agent which may contain no free variables except $x_1, \ldots, x_n$, which must be distinct.

We basically have all the standard operators of value-passing CCS. In particular, the synchronization between parallel processes allows to exchange a value through the following simple mechanism: a system $c(x).E_1 \parallel \overline{c}\, m.E_2$ can execute an internal τ action moving to $E_1[m/x] \parallel E_2$, where $E_1[m/x]$ is the process E_1 with all the occurrences of x replaced by m. Thus, process $c(x).E_1$, is indeed receiving in variable x the value m sent out by process $\overline{c}\, m.E_2$.

Besides the standard value-passing CCS operators, we have an additional one that has been introduced in order to model message handling and cryptography. Informally, the $[\langle m_1 \ldots m_r \rangle \vdash_{rule} x]E_1; E_2$ process tries to deduce an information z from the tuple of messages $\langle m_1 \ldots m_r \rangle$ through one application of rule $\vdash_{rule}$; if it succeeds then it behaves like $E_1[z/x]$, otherwise it behaves like E_2; for example, given a rule $\vdash_{dec}$ for decryption, process $[\langle \{m\}_k, k^{-1} \rangle \vdash_{dec} x]E_1; E_2$ decrypts message $\{m\}_k$ through key k^{-1} and behaves like $E_1[m/x]$ while $[\langle \{m\}_k, k' \rangle \vdash_{dec} x]E_1; E_2$ (with $k' \neq k^{-1}$) tries to decrypt the same message with the wrong inverse key k' and (since it is not permitted by $\vdash_{dec}$) it behaves like E_2.

$$\frac{m \quad m'}{(m,m')}\ (\vdash_{pair}) \qquad \frac{(m,m')}{m}\ (\vdash_{fst}) \qquad \frac{(m,m')}{m'}\ (\vdash_{snd})$$

$$\frac{m \quad k}{\{m\}_k}\ (\vdash_{enc}) \qquad \frac{\{m\}_k \quad k^{-1}}{m}\ (\vdash_{dec})$$

Fig. 1. Message manipulation, where $m, m' \in \mathcal{M}$ and $k, k^{-1} \in K$.

We call $\mathcal{E}$ the set of all the CryptoSPA terms, and we define $sort(E)$ to be the set of all the channels syntactically occurring in the term E.

2.2 *The Operational Semantics of CryptoSPA*

The semantics of CryptoSPA is given through *labelled transition systems.* A labelled transition system (LTS) is essentially an automaton with possibly infinitely many states. It is defined as a triple $(S, T, \rightarrow)$ such that S is a set of states, T is a set of labels (actions) and $\rightarrow \subseteq S \times T \times S$ is a set of labelled transitions. $(S_1, \alpha, S_2) \in \rightarrow$ (or equivalently $S_1 \xrightarrow{\alpha} S_2$) means that the system can move from the state S_1 to the state S_2 through the action α.

In order to model message handling and cryptography, in Figure 1 we define an inference system which formalizes the way messages may be manipulated by processes. It is indeed quite similar to those used by many authors (see, e.g., [9,12]). In particular it can combine two messages obtaining a pair (rule $\vdash_{pair}$); it can extract one message from a pair (rules $\vdash_{fst}$ and $\vdash_{snd}$); it can encrypt a message m with a key k obtaining $\{m\}_k$ and finally decrypt a message of the form $\{m\}_k$ only if it has the corresponding (inverse) key k^{-1} (rules $\vdash_{enc}$ and $\vdash_{dec}$). We denote with $\mathcal{D}(\phi)$ the set of messages that can be deduced by applying the inference rules on the messages in ϕ. Note that we are assuming encryption as completely reliable. Indeed we do not allow any kind of cryptographic attack, e.g., the guessing of secret keys. This permits to observe the attacks that can be carried out even if cryptography is completely reliable.

The formal behaviour of a CryptoSPA term is described by means of the LTS $< \mathcal{E}, Act, \{\xrightarrow{a}\}_{a \in Act} >$, where $\xrightarrow{a}$ is the least relation between CryptoSPA terms induced by axioms and inference rules of Figure 2 (where symmetric rules for $+_1$, $\|_1$ and $\|_2$ are omitted for the sake of readability). The operational semantics for a term E is the subpart of the CryptoSPA LTS reachable from the initial state E.

Example 2.1 We present a very simple example of a protocol where A sends a message m_A to B encrypted with a key k_{AB} shared between A and B. [4]

[4] For the sake of readability, we omit the termination $\underline{0}$ at the end of every agent specifications, e.g., we write a in place of $a.\underline{0}$. We also write $[m = m']E$ in place of $[m = m']E; \underline{0}$ and analogously for $[\langle m_1 \dots m_r \rangle \vdash_{rule} x]E; \underline{0}$.

$$(input)\frac{m \in Msg(c)}{c(x).E \xrightarrow{c(m)} E[m/x]} \qquad (output)\frac{m \in Msg(c)}{\overline{c}\, m.E \xrightarrow{\overline{c}\, m} E} \qquad (internal)\frac{}{\tau.E \xrightarrow{\tau} E}$$

$$(\|_1)\frac{E \xrightarrow{a} E'}{E \,\|\, E_1 \xrightarrow{a} E' \,\|\, E_1} \qquad (\|_2)\frac{E \xrightarrow{c(m)} E' \quad E_1 \xrightarrow{\overline{c}\, m} E_1'}{E \,\|\, E_1 \xrightarrow{\tau} E' \,\|\, E_1'} \qquad (+_1)\frac{E \xrightarrow{a} E'}{E + E_1 \xrightarrow{a} E'}$$

$$(=_1)\frac{m \neq m' \quad E_2 \xrightarrow{a} E_2'}{[m = m']E_1; E_2 \xrightarrow{a} E_2'} \qquad (=_2)\frac{m = m' \quad E_1 \xrightarrow{a} E_1'}{[m = m']E_1; E_2 \xrightarrow{a} E_1'}$$

$$(\backslash L)\frac{E \xrightarrow{a} E' \quad chan(a) \notin L}{E\backslash L \xrightarrow{a} E'\backslash L} \quad (def)\frac{E[m_1/x_1, \ldots, m_n/x_n] \xrightarrow{a} E' \quad A(x_1, \ldots, x_n) \overset{\text{def}}{=} E}{A(m_1, \ldots, m_n) \xrightarrow{a} E'}$$

$$(\mathcal{D}_1)\frac{\langle m_1 \ldots m_r\rangle \vdash_{rule} m \qquad E_1[m/x] \xrightarrow{a} E_1'}{[\langle m_1 \ldots m_r\rangle \vdash_{rule} x]E_1; E_2 \xrightarrow{a} E_1'}$$

$$(\mathcal{D}_2)\frac{\nexists m : \langle m_1 \ldots m_r\rangle \vdash_{rule} m \qquad E_2 \xrightarrow{a} E_2'}{[\langle m_1 \ldots m_r\rangle \vdash_{rule} x]E_1; E_2 \xrightarrow{a} E_2'}$$

Fig. 2. Operational semantics (symmetric rules omitted).

$$A(m,k) \overset{\text{def}}{=} [\langle m, k\rangle \vdash_{enc} x]\overline{c}\, x$$
$$B(k) \overset{\text{def}}{=} c(y).[\langle y, k\rangle \vdash_{dec} z]\overline{out}\, z$$
$$P \overset{\text{def}}{=} A(m_A, k_{AB}) \,\|\, B(k_{AB})$$

where $k_{AB}^{-1} = k_{AB}$, that models a symmetric encryption, and $Msg(c) = \{\{m\}_k \mid m \in M, k \in K\}$ that declares the type of messages sent over c. We want to analyze the execution of P with no intrusions, we thus consider $P \setminus \{c\}$, since the restriction guarantees that c is a completely secure channel. We obtain a system which can only execute action $\overline{out}\, m_A$ that represents the correct transmission of m_A from A to B. In particular, we have that $A(m_A, k_{AB}) \xrightarrow{\overline{c}\{m_A\}_{k_{AB}}} \underline{0}$ and $B(k_{AB})$ can synchronize on that action by executing a $B(k_{AB}) \xrightarrow{c(\{m_A\}_{k_{AB}})} [\langle\{m_A\}_{k_{AB}}, k_{AB}\rangle \vdash_{dec} z]\overline{out}\, z$. So

$$P \setminus \{c\} \xrightarrow{\tau} (\underline{0} \;\|\; [\langle\{m_A\}_{k_{AB}}, k_{AB}\rangle \vdash_{dec} z]\overline{out}\, z) \setminus \{c\} \xrightarrow{\overline{out}\, m_A} (\underline{0} \,\|\, \underline{0}) \setminus \{c\}$$

In the next section we analyze the execution of this simple protocol in a hostile environment. ■

2.3 Hostile Environments

In this section we report the characterization of hostile environments as given in [7]. Such a characterization is necessary to analyze protocols where some information is assumed to be secret, as it always happens in cryptographic protocols. Basically, a *hostile environment* is an agent which tries to attack a protocol by stealing and faking the information which is transmitted on the CryptoSPA *public* channels in set C. In principle, such an agent could be modeled as a generic process X which can communicate only through the channels belonging to C, i.e., $X \in \mathcal{E}_C$ where $\mathcal{E}_C \overset{\text{def}}{=} \{E \in \mathcal{E} \mid sort(E) \subseteq C\}$. However, in this way we obtain that X is a completely powerful attacker which is able to "guess" every secret information (e.g., cryptographic keys, nonces, private messages) and is thus not suitable when analyzing cryptographic protocols. We show this crucial point through a simple example.

Example 2.2 Consider again the protocol P of Example 2.1. Since only A and B know k_{AB}, this protocol should guarantee the secrecy of m_A even in the presence of a hostile environment. We assume that $c, pub \in C$ are public channels and we consider the following process:

$$X(k) \stackrel{\text{def}}{=} c(x).[\langle x, k\rangle \vdash_{dec} m]\ \overline{pub}\, y$$

It intercepts a message sent over channel c and tries to decrypt it using key k. If it succeeds, then it makes the message public by sending it as plaintext over channel *pub*. Note that $X(k)$ belongs to $\mathcal{E}_C$ since $sort(X(m,k)) = \{c, pub\}$. Consider now $X(k_{AB})$ and the following protocol "under attack" (note that we put X inside the scope of restriction):

$$(P \,\|\, X(k_{AB})) \setminus C$$

After one τ step, $X(k_{AB})$ will be able to execute $\overline{pub}\, m_A$, representing the fact that m_A is not secret anymore. [5] This happens since $X(k_{AB})$ is able to guess k_{AB}, but we would like to forbid such behaviour since, as mentioned above, we are interested in attacks that can be carried out even when cryptography is completely reliable. ■

This problem can be solved by imposing some constraints on the initial data that are known by the intruders. Given a process E, we call $ID(E)$ the set of messages that syntactically appear in E. Intuitively, this set contains all the messages that are initially known by E. Now, let $\phi_I \subseteq \mathcal{M}$ be the initial knowledge that we would like to give to the intruder X, i.e., the public information such as the names of the entities and the public keys, plus some possible private data of the intruder (e.g., its private key or nonces). For a certain intruder X, we want that all the messages in $ID(X)$ are deducible from ϕ_I.

The set $\mathcal{E}_C^{\phi_I}$ of processes which can communicate on a subset of C and have an initial knowledge bound by ϕ_I can be thus defined as follows:

$$\mathcal{E}_C^{\phi_I} = \{X \mid X \in \mathcal{E}_C \text{ and } ID(X) \subseteq \mathcal{D}(\phi_I)\}$$

We consider as hostile processes only those belonging to this particular set. In the example above, if we require that k_{AB} is not deducible from ϕ_I (i.e., it is not public) we can easily see that the behavior of $X(k_{AB})$ cannot be simulated by any process in $\mathcal{E}_C^{\phi_I}$.

[5] Indeed, such an event is not directly observable since $pub \in C$. In Section 3 we will show how to solve this.

2.4 *Trace Equivalence*

Most of the security properties that have been proposed for the analysis of security protocols are based on the simple notion of *trace*: two processes are equivalent if they exactly show the same execution sequences (called *traces*). In order to formally define it, we need a transition relation which does not consider internal τ moves.

Definition 2.3 The expression $E \stackrel{\alpha}{\Longrightarrow} E'$ is a shorthand for $E(\stackrel{\tau}{\longrightarrow})^* E_1 \stackrel{\alpha}{\longrightarrow} E_2(\stackrel{\tau}{\longrightarrow})^* E'$, where $(\stackrel{\tau}{\longrightarrow})^*$ denotes a (possibly empty) sequence of τ labelled transitions. Let $\gamma = \alpha_1 \ldots \alpha_n \in (Act \setminus \{\tau\})^*$ be a trace; then $E \stackrel{\gamma}{\Longrightarrow} E'$ if there exist $E_1, E_2, \ldots, E_{n-1} \in \mathcal{E}$ such that $E \stackrel{\alpha}{\Longrightarrow} E_1 \stackrel{\alpha_2}{\Longrightarrow} \cdots \stackrel{\alpha_{n-1}}{\Longrightarrow} E_{n-1} \stackrel{\alpha_n}{\Longrightarrow} E'$. ■

We thus define *trace preorder* ($\leq_{trace}$) and *trace equivalence* ($\approx_{trace}$):

Definition 2.4 For any $E \in \mathcal{E}$ the set $T(E)$ of *traces associated with* E is $T(E) = \{\gamma \in (Act \setminus \{\tau\})^* \mid \exists E' : E \stackrel{\gamma}{\Longrightarrow} E'\}$. F can execute all the traces of E (notation $E \leq_{trace} F$) iff $T(E) \subseteq T(F)$. E and F are *trace equivalent* (notation $E \approx_{trace} F$) iff $E \leq_{trace} F$ and $F \leq_{trace} E$, i.e., iff $T(E) = T(F)$. ■

3 Secrecy in Protocols through Non Interference

In this section we show that NDC can be easily adapted for analysing secrecy in networks. NDC is defined as follows:

Definition 3.1 A process S is NDC iff

$$\forall X \in \mathcal{E}_C^{\phi_I} \quad (S \,||\, X) \setminus C \approx_{trace} S \setminus C$$ ■

In other words S is NDC if every possible enemy X which has an initial knowledge limited by ϕ_I is not able to significantly change the behaviour of the system. Note that $S \setminus C$ is the system S where the public channels C are made private, i.e., where no enemy can intercept or introduce fake messages.

Consider now a protocol $P(M)$ and assume that we want to verify if $P(M)$ preserves the secrecy of message M. This can be done by proving that every enemy which does not know message M, cannot learn it by interacting with $P(M)$. Thus, we need a mechanism that notifies whenever an enemy is learning M. We implement it through a simple process called *knowledge notifier* which reads from a public channel $c_k \in C \setminus sort(P(M))$ not used in $P(M)$ and executes a $\overline{learnt}\, M$ action if the read value is exactly equal to M. For a generic message m, it can be defined as follows:

$$KN(m) \stackrel{\mathrm{def}}{=} c_k(y).[m = y]\overline{learnt}\, m$$

We assume that *learnt* is a special channel that is never used by protocols and is not public, i.e., $learnt \notin sort(P) \cup C$. We now consider $P'(m) \stackrel{\mathrm{def}}{=} P(m) \,||\, KN(m)$, i.e., a modified protocol where the learning of M is now notified. A very intuitive definition of secrecy can be thus given as follows:

Definition 3.2 $P(m)$ preserves the secrecy of m iff for all (secret) messages $M \in \mathcal{M} \setminus \mathcal{D}(\phi_I)$ and for all enemies $X \in \mathcal{E}_C^{\phi_I}$ there exist no trace $\mu_1 \dots \mu_n$ such that $\mu_1 \dots \mu_n \overline{learnt}\, M \in T((P'(M) \| X) \setminus C)$. ■

In other words, we require that for every secret M and for every enemy X, process $(P'(M) \| X) \setminus C$ never executes an $\overline{learnt}\, M$ action.

This definition models very well the notion of secrecy. On the one hand, if $(P'(M) \| X) \setminus C$ executes $\overline{learnt}\, M$ then M has been sent over the public channel c_k by either $P(M)$ or X. In both cases the message is not secret anymore. In the former situation $P(M)$ is making M public, while in the latter X has for sure learned M before sending it over c_k. On the other hand, if an enemy X is able to learn message M then there also exists an enemy X' that will send such a message over channel c_k and thus $(P'(M) \| X) \setminus C$ will eventually execute $\overline{learnt}\, M$.

We want now to use NDC to perform this check. Note that NDC already contains the quantification over all the possible enemies. The following holds:

Proposition 3.3 *Consider a protocol $P(m)$ such that $sort(P(m)) \subseteq C \setminus \{c_k\}$. Then, $P(m)$ preserves the secrecy of m in the sense of Definition 3.2 iff*

$$\forall M \in \mathcal{M} \setminus \mathcal{D}(\phi_I) \qquad P'(M) \textit{ is NDC}$$

Proof. ($\Rightarrow$) Note that $P'(M) \setminus C \approx_{trace} \underline{0}$, since $sort(P(m)) \subseteq C \setminus \{c_k\}$. As a matter of fact, $\overline{learnt}\, M$ is the only action that $P'(M) \setminus C$ could execute (it is the only one which is not captured by the restriction over C) but $P(M)$ cannot send messages over c_k and $learnt$. Moreover, we have that $sort(X) \subseteq C$, thus the only action that is executable by $(P'(M) \| X) \setminus C$ is again $\overline{learnt}\, M$. Now if, by Definition 3.2, $(P'(M) \| X) \setminus C$ never executes an $\overline{learnt}\, M$ action, then we obtain $(P'(M) \| X) \setminus C \approx_{trace} \underline{0} \approx_{trace} P'(M) \setminus C$ for every X, i.e., $P'(M)$ is NDC.
($\Leftarrow$) If $P'(M)$ is NDC then for all enemies X we have $(P'(M) \| X) \setminus C \approx_{trace} P'(M) \setminus C \approx_{trace} \underline{0}$. This of course means that $(P'(M) \| X) \setminus C$ cannot execute $\overline{learnt}\, M$ and holds for every possible M. ■

Intuitively, this result means that NDC corresponds to secrecy when (i) the only action we observe is exactly $\overline{learnt}\, M$ and (ii) channel c_k is a special one that cannot be used in $P(m)$.

These requirements are both captured by the condition $sort(P(m)) \subseteq C \setminus \{c_k\}$, i.e., the specification $P(m)$ can use neither c_k nor channels that are not public, thus not restricted in the composition with the enemy. Usually these particular channels are called observable (e.g. *learnt* is an observable channel). Note that such a condition is not restrictive as it only requires a particular form of the specification. Moreover, it is consistent with the idea of NDC-based verifications (see, e.g., [4,7]): we fix the property (NDC) and we capture different properties by just defining different observable actions.

4 An Example

In this section we show through a simple example how the NDC-based secrecy verification works. We consider a simplified version of the Wide Mouthed Frog Protocol [2].

Consider two processes A and B respectively sharing keys k_{AS} and k_{BS} with a trusted server S. In order to establish a secure channel with B, A sends a fresh key k_{AB} encrypted with k_{AS} to the server S. Then, the server decrypts the key and forwards it to B, this time encrypted with k_{BS}. Now B has the key k_{AB} and A can send a message m_A encrypted with k_{AB} to B. The protocol is composed of the following three messages:

$$\begin{array}{l}
\text{Message 1 } A \rightarrow S \;:\; A, B, \{k_{AB}\}_{k_{AS}} \\
\text{Message 2 } S \rightarrow B \;:\; \{A, k_{AB}\}_{k_{BS}} \\
\text{Message 3 } A \rightarrow B \;:\; \{m_A\}_{k_{AB}}
\end{array}$$

The main differences with respect to the original protocol is that here messages 1 and 2 do not contain timestamps (as studied in [1] for authentication). Moreover, in message 1 the identifier B is sent as plaintext while in the original protocol it is encrypted with the session key (this modification generates, as we will show, a secrecy attack). We specify the protocol as follows: [6]

$$\begin{array}{rl}
A(m,k) \overset{\text{def}}{=} & \overline{c_1}\,((A,B),\{k\}_{k_{AS}})\;.\;\overline{c_3}\,\{m\}_k \\
B \overset{\text{def}}{=} & c_2(y)\;.\;[\langle y, k_{BS}\rangle \vdash_{dec} z]\;[z \vdash_{snd} s]\;c_3(t)\;.\;[\langle t, s\rangle \vdash_{dec} w] \\
S \overset{\text{def}}{=} & c_1(x)\;.\;[x \vdash_{fst} i]\;[i \vdash_{fst} s]\;[i \vdash_{snd} r] \\
& \quad [x \vdash_{snd} c]\;[\langle c, K(s)\rangle \vdash_{dec} z]\;\overline{c_2}\,\{(s,z)\}_{K(r)}\;.\;S \\
P(n) \overset{\text{def}}{=} & A(n, k_{AB}) \parallel B \parallel S
\end{array}$$

where $K(id)$ is a function that returns the key shared between the server and entity id (e.g., $K(A)$ returns k_{AS}). Moreover we have that $\{c_1, c_2, c_3\} \subseteq C$.

Since $P(n)$ only uses channels c_1, c_2, c_3 we have that $sort(P(n)) \subseteq C \setminus \{c_k\}$. The condition of Proposition 3.3 is then satisfied and we can prove, through NDC, that $P(m)$ does not preserve the secrecy of m if $A, E, K_{ES} \in \phi_I$. Consider the following enemy:

$$\begin{array}{rll}
\tilde{X} \overset{\text{def}}{=} & c_1(x)\;[x \vdash_{snd} y] & \texttt{\% intercepts message 1} \\
& \overline{c_1}\,((A,E),y). & \texttt{\% replaces B with E, sends it} \\
& c_2(z)\;[\langle z, k_{ES}\rangle \vdash_{dec} w]\;[w \vdash_{snd} k]. & \texttt{\% intercepts msg 2, obtains k} \\
& c_3(j)[\langle j, k\rangle \vdash_{dec} m] & \texttt{\% decrypts msg 3} \\
& \overline{c_k}\,m & \texttt{\% sends message to KN}
\end{array}$$

[6] We encode tuples through a left associative canonical form, e.g., the first message $A, B, \{k_{AB}\}_{k_{AS}}$ becomes $((A,B),\{k_{AB}\}_{k_{AS}})$

It is easy to see that process $(P'(M) \,\|\, \tilde{X}) \setminus C \not\approx_{trace} P'(M) \setminus C$ as the former process can execute $\overline{learnt}\, M$. The attack performed by $\tilde{X}$ is the following:

$$\begin{array}{lll} \text{Message 1} & A \to E(S) & : \ A, B, \{K_{AB}\}_{K_{AS}} \\ \text{Message 1}' & E(A) \to S & : \ A, E, \{K_{AB}\}_{K_{AS}} \\ \text{Message 2}' & S \to E & : \ \{A, K_{AB}\}_{K_{ES}} \\ \text{Message 3} & A \to E(B) & : \ \{M\}_{K_{AB}} \end{array}$$

By message $2'$ the enemy learns K_{AB} and, by message 3 which is addressed to B, it finally learns M.

Consider now the protocol where, in the first message, B is encrypted with the session key:

$$\text{Message 1 } A \to S : \ A, \{B, K_{AB}\}_{K_{AS}}$$

Here the secrecy attack is not possible anymore. We can prove this automatically through the CoSeC/CVS tools [6,4], as discussed in the next section.

5 Future Work

In order to compare various formalizations of security properties, we have defined in [7] a general scheme that permits to capture a number of properties (e.g., authentication as in [10] and denial of service as in [16]) as particular instances of the scheme itself. The results presented in this paper have allowed us to extend the set of properties defined in the scheme. Our main issue is now to find comparison results in order to obtain a complete classifications in the style of [5,10], which could help in evaluating the relative merits of all such properties.

Another aim of our current research is to provide general, effective proof techniques that can be suitably applied to a set of security properties. Indeed, the definition of security properties as instances of the GNDC scheme [7] allows us to use a uniform analysis technique in order to check all of them. For example in [4] we show how NDC can be used to check authentication properties. Moreover, in this paper we have seen that secrecy can be easily defined as NDC. This permits the reuse of automatic checking techniques for NDC in order to check secrecy over CryptoSPA protocols; indeed if ϕ_I is finite then it is possible to find a most-general intruder Top such that NDC is reduced to just one check $(P'(M) \,\|\, Top) \setminus C \approx_{trace} P'(M) \setminus C$ (see [7] for more details) that can be verified using the tool in [4].

This kind of verification can be, in principle, applied to all the properties we have defined in our scheme. Moreover, the flexibility of the GNDC-scheme makes it possible to verify different properties in just one (NDC) check [3]. An alternative way of analysing security properties with the GNDC scheme is to apply compositional analysis techniques as done in [13,14]. These can be used even when the most-general intruder approach described above is not applicable.

An automatic verification can be carried out only if we fix in advance the maximum number of instances of A and B (as done in the example of Section 4). Some recent results (see, e.g., [11]) show that the verification of a fixed number of sessions of a protocol can be, in some cases, sufficient to prove the general correctness of such a protocol. It would be interesting to have similar results for GNDC, since they could be applied to all the properties defined in the scheme.

References

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[2] M. Burrows, M. Abadi, and R. Needham. “A Logic of Authentication”. *Proceedings of the Royal Society of London*, 426:233–271, 1989.

[3] A. Durante, R. Focardi, and R. Gorrieri. A compiler for analysing cryptographic protocols using non-interference. Submitted for publication.

[4] A. Durante, R. Focardi, and R. Gorrieri. CVS: A compiler for the analysis of cryptographic protocols. In *Proceedings of CSFW'99*, pages 203–212. IEEE press, 1999.

[5] R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–33, 1994/1995.

[6] 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):550–571, 1997.

[7] R. Focardi and F. Martinelli. A uniform approach for the definition of security properties. In *Proceedings of World Congress on Formal Methods (FM'99)*, pages 794–813. Springer, LNCS 1708, 1999.

[8] J. A. Goguen and J. Meseguer. Security policy and security models. In *Proc. of the 1982 Symposium on Security and Privacy*, pages 11–20. IEEE Press, 1982.

[9] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS'96*, pages 146–166. LNCS 1055, 1996.

[10] G. Lowe. A hierarchy of authentication specification. In *Proceedings of the 10th Computer Security Foundation Workshop*, pages 31–43. IEEE press, 1997.

[11] G. Lowe. “Towards a Completeness Result for Model Checking of Security Protocols”. In *Proceedings Eleventh IEEE Computer Security Foundation Workshop, (CSFW'98)*, Rockport Massachusetts (USA), June 1998. IEEE Press.

[12] W. Marrero, E. Clarke, and S. Jha. A model checker for authentication protocols. In *Proc. of DIMACS Workshop on Design and Formal Verification of Security Protocols.* Rutgers University, Sep. 1997.

[13] F. Martinelli. Languages for description and analysis of authentication protocols. In *Proceedings of ICTCS'98*, pages 304–315. World Scientific, 1998.

[14] F. Martinelli. Partial model checking and theorem proving for ensuring security properties. In *Proceedings of CSFW'98*, pages 44–52. IEEE press, 1998.

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

[16] S. Schneider. Formal analysis of a non-repudiation protocol. In *Proceedings of CSFW'98*, pages 54–65. IEEE Press, 1998.

[17] S. Schneider. Verifying authentication protocols in CSP. *IEEE Transactions on Software Engineering*, 24(9), September 1998.