

# Security protocols verification in Abductive Logic Programming: a case study

Marco Alberti<sup>1</sup>, Federico Chesani<sup>2</sup>, Marco Gavanelli<sup>1</sup>, Evelina Lamma<sup>1</sup>, Paola Mello<sup>2</sup>, and Paolo Torroni<sup>2</sup>

<sup>1</sup> ENDIF, Università di Ferrara - Via Saragat, 1 - 44100 Ferrara (Italy).  
Email: {malberti|mgavanelli|elamma}@ing.unife.it

<sup>2</sup> DEIS, Università di Bologna - Viale Risorgimento 2 - 40126 Bologna (Italy).  
Email: {fchesani|pmello|ptorroni}@deis.unibo.it

**Abstract.** In this paper we present by a case study an approach to the verification of security protocols based on Abductive Logic Programming.

We start from the perspective of open multi-agent systems, where the internal architecture of the individual system's components may not be completely specified, but it is important to infer and prove properties about the overall system behaviour. We take a formal approach based on Computational Logic, to address verification at two orthogonal levels: 'static' verification of protocol properties (which can guarantee, at design time, that some properties are a logical consequence of the protocol), and 'dynamic' verification of compliance of agent communication (which checks, at runtime, that the agents do actually follow the protocol).

We adopt as a running example the well-known Needham-Schroeder protocol. We first show how the protocol can be specified in our previously developed *SOCS-SI* framework, and then demonstrate the two types of verification.

## 1 Introduction

The recent and fast growth of network infrastructures, such as the Internet, is allowing for a new range of scenarios and styles of business-making and transaction-management. In this context, the use of security protocols has become common practice in a community of users who often operate in the hope (and sometimes in the trust) that they can rely on a technology which protects their private information and makes their communications secure and reliable. A large number of formal methods and tools have been developed to analyse security protocols, achieving notable results in determining their strengths (by showing their security properties) and their weaknesses (by identifying attacks on them).

The need for well defined protocols is even more apparent in the context of multi-agent systems, where agents are abstraction for autonomous computational entities which can act on behalf of their users. By well defined, we mean that protocols should be specified so as to guarantee that, provided that the

agents follow them, the resulting interaction will exhibit some desirable properties. In order to achieve reliability and users' trust, formal proofs of such properties need to be provided. We call the generation of such formal proofs *static verification of protocol properties*. A tool performing this activity automatically, rather than by hand, is an obvious request.

Open agent societies are defined as dynamic groups of agents, where new agents can join the society at any time, without disclosing their internals or specifications, nor providing any formal credential of being "well behaved". Open agent societies are a useful setting for heterogeneous agent to interact; but, since no assumptions can be made about the agents and their behaviour, it cannot be assumed that the agents will follow the protocols. Therefore, at run-time, the resulting agent interaction may not exhibit the protocol properties that were verified statically at design time. However, one possible way to tackle (at least partly) this problem is to be able to guarantee that, if one agent misbehaves, its violation will be detected (and, possibly, sanctioned). Following Guerin and Pitt [GP02], we call this *on-the-fly verification of compliance*. This kind of verification should be performed by a trusted entity, external to the agents.

In previous work, and in the context of the EU-funded SOCS project [SOCa] we developed a Computational Logic-based framework, called *SOCS-SI* (where *SI* stands for *Social Infrastructure*), for the specification of agent interaction. In order to make *SOCS-SI* applicable to open agent societies, the specifications refer to the *observable* agent behaviour, rather than to the agents' internals or policies, and do not overconstrain the agent behaviour. We have shown in previous work that *SOCS-SI* is suitable for semantic specification of agent communication languages [ACG<sup>+</sup>03], and that it lends itself to the definition of a range of agent interaction protocols [AGL<sup>+</sup>03b]. A repository of protocols is available on the web [SOCb].

In this paper, we demonstrate by a case study on the well known Needham-Schroeder security protocol [NS78] how the *SOCS-SI* framework supports both static verification of protocol properties and on-the-fly verification of compliance. The two kinds of verifications are achieved by means of the operational counterpart of the *SOCS-SI* framework, consisting of two abductive proof procedures (*SCIFF* and *g-SCIFF*). Notably, the same specification of the protocol in our language is used for both kinds of verification: in this way, the protocol designer is relieved from a time consuming (and, what is worse, possibly erroneous) translation.

*SOCS-SI*, together with its proof procedures, can thus be seen as a tool which lets the protocol designer automatically verify: (i) at design time, that a protocol enjoys some desirable properties, and (ii) at runtime, that the agents follow the protocol, so making the interaction indeed exhibit the properties.

The paper is structured as follows. In Sect. 2, we describe an implementation of the well-known Needham Schroeder Public Key authentication protocol in our framework, and in Sect. 3 we show how we perform on-the-fly verification of compliance and static verification of properties of the protocol. Related work and conclusions follow.

## 2 Specifying the Needham-Schroeder Public Key encryption protocol

In this section, we show how our framework can be used to represent the well-known Needham-Schroeder security protocol [NS78].

- (1)  $A \rightarrow B : \langle N_A, A \rangle_{pub\_key(B)}$
- (2)  $B \rightarrow A : \langle N_A, N_B \rangle_{pub\_key(A)}$
- (3)  $A \rightarrow B : \langle N_B \rangle_{pub\_key(B)}$

**Fig. 1.** The Needham-Schroeder protocol (simplified version)

The protocol consists of seven steps, but, as other authors do, we focus on a simplified version consisting of three steps, where we assume that the agents know the public key of the other agents. A protocol run can be represented as in Figure 1, where  $A \rightarrow B : \langle M \rangle_{PK}$  means that  $A$  has sent to  $B$  a message  $M$ , encrypted with the key  $PK$ .

- (1)  $a \rightarrow i : \langle N_a, a \rangle_{pub\_key(i)}$
- (2)  $i \rightarrow b : \langle N_a, a \rangle_{pub\_key(b)}$
- (3)  $b \rightarrow i : \langle N_a, N_b \rangle_{pub\_key(a)}$
- (4)  $i \rightarrow a : \langle N_a, N_b \rangle_{pub\_key(a)}$
- (5)  $a \rightarrow i : \langle N_b \rangle_{pub\_key(i)}$
- (6)  $i \rightarrow b : \langle N_b \rangle_{pub\_key(b)}$

**Fig. 2.** Lowe's attack on the Needham-Schroeder protocol

**Lowe's attack on the protocol.** Eighteen years after the publication of the Needham-Schroeder protocol, Lowe [Low96] proved it to be prone to a security attack. Lowe's attack on the protocol is presented in Figure 2, where a third agent  $i$  (standing for *intruder*) manages to successfully authenticate itself as agent  $a$  with a third agent  $b$ , by exploiting the information obtained in a legitimate dialogue with  $a$ .

### 2.1 The social model

In this section we give a brief summary of the *SOCS-SI* social framework developed within the EU-funded SOCS project [SOCa]<sup>3</sup> to specify interaction protocols for open societies of agents in a declarative way.

The agent interaction is observed and recorded by the social infrastructure in a set **HAP** (called *history*), of *events*. Events are represented as ground atoms

$$\mathbf{H}(\text{Event}[, \text{Time}])$$

<sup>3</sup> The reader can refer to [AGL<sup>+</sup>03a] for a more detailed description.

The term *Event* describes the event that has happened, according to application-specific conventions (e.g., a message sent or a payment issued); *Time* (optional) is a number, meant to represent the time at which the event has happened.

For example,

$$\mathbf{H}(\text{send}(a, b, \text{content}(\text{key}(k_b), \text{agent}(a), \text{nonce}(n_a))), 1)$$

could represent the fact that agent  $a$  sent to agent  $b$  a message consisting its own identifier ( $a$ ) and a nonce ( $n_a$ ), encrypted with the key  $k_b$ , at time 1.

While events represent the actual agent behaviour, the desired agent behaviour is represented by *expectations*. Expectations are “positive” when they refer to events that are expected to happen, and “negative” when they refer to events that are expected *not* to happen. The following syntax is adopted

$$\mathbf{E}(\text{Event}[, \text{Time}]) \quad \mathbf{EN}(\text{Event}[, \text{Time}])$$

for, respectively, positive and negative expectations. Differently from events, expectations can contain variables (we follow the Prolog convention of representing variables with capitalized identifiers) and CLP [JM94] constraints can be imposed on the variables. This is because the desired agent behaviour may be under-specified (hence variables), yet subject to restriction (hence CLP constraints).

For instance,

$$e(\text{send}(a, b, \text{content}(\text{key}(k_b), \text{nonce}(n_b), \text{empty}(0))), T)$$

could represent the expectation for agent  $a$  to send to agent  $b$  a message consisting of a nonce ( $n_b$ ) and an empty part ( $\text{empty}(0)$ ), encrypted with a key  $k_b$ , at time  $T$ . A CLP constraint such as  $T \leq 10$  can be imposed on the time variable, to express a deadline.

Explicit negation can be applied to expectations ( $\neg\mathbf{E}$  and  $\neg\mathbf{EN}$ ).

A protocol specification  $\mathcal{S} = \langle KB_S, \mathcal{IC}_S \rangle$  is composed of:

- the *Social Knowledge Base* ( $KB_S$ ) is a logic program whose clauses can have expectations and CLP constraints in their bodies. It can be used to express domain-specific knowledge (such as, for instance, deadlines);
- a set  $\mathcal{IC}_S$  of *Social Integrity Constraints* (also SICs, for short, in the following): rules of the form  $Body \rightarrow Head$ . SICs are used to express how the actual agent behaviour generates expectations on their behaviour; examples can be found in the following sections.

In abductive frameworks [KKT93], *abducibles* represent hypotheses, and abduction is used to select a set of hypotheses consistent with some specification. In our (abductive) framework, expectations are abducibles, and the abductive semantics is used to select a desired behaviour consistent with the instance in question.

In particular, we say that a history **HAP** is *compliant* to a specification  $\mathcal{S} = \langle KB_S, \mathcal{IC}_S \rangle$  iff there existss a set **EXP** of expectations that is

- $\mathcal{IC}_S$ -consistent: it must entail  $\mathcal{IC}_S$ , for the given  $\mathcal{S}_{\mathbf{HAP}}$ ;
- $\neg$ -consistent: for any  $p$ ,  $\mathbf{EXP}$  cannot include  $\{\mathbf{E}(P), \neg\mathbf{E}(p)\}$  or  $\{\mathbf{EN}(p), \neg\mathbf{EN}(p)\}$ ;
- $\mathbf{E}$ -consistent: for any  $p$ ,  $\mathbf{EXP}$  cannot include  $\{\mathbf{E}(p), \mathbf{EN}(p)\}$  (an event cannot be both expected to happen and expected not to happen);
- fulfilled:  $\mathbf{EXP}$  cannot contain  $\mathbf{EN}(p)$  if  $\mathbf{HAP}$  contains  $\mathbf{H}(p)$ , and  $\mathbf{EXP}$  cannot contain  $\mathbf{E}(p)$  if  $\mathbf{HAP}$  does not contain  $\mathbf{H}(p)$ .

In order to support goal-oriented societies,  $\mathbf{EXP}$  is also required to entail, together with  $KB_S$ , a goal  $\mathcal{G}$  which is defined as a conjunction of literals.

## 2.2 Representing the Needham-Schroeder protocol in the *SOCS-SI* social model

With the atom:

$$\mathbf{H}(\text{send}(X, Y, \text{content}(\text{key}(K), \text{Term}_1, \text{Term}_2)), T_1)$$

we mean that a message is sent by an agent  $X$  to an agent  $Y$ ; the content of the message consists of the two terms  $\text{Term}_1$  and  $\text{Term}_2$  and has been encrypted with the key  $K$ .  $T_1$  is the time at which  $Y$  receives the message.

The interaction of Fig. 1, for instance, can be expressed as follows:

$$\begin{aligned} &\mathbf{H}(\text{send}(a, b, \text{content}(\text{key}(k_b), \text{agent}(a), \text{nonce}(n_a))), 1) \\ &\mathbf{H}(\text{send}(b, a, \text{content}(\text{key}(k_a), \text{nonce}(n_a), \text{nonce}(n_b))), 2) \\ &\mathbf{H}(\text{send}(a, b, \text{content}(\text{key}(k_b), \text{nonce}(n_b), \text{empty}(0))), 3) \end{aligned}$$

A first group of SICs, depicted in Fig. 3, defines the protocol itself, i.e, the expected sequence of messages.

```

H( send( X, B, content( key( KB), agent( A), nonce( NA))), T1)
---->
E( send( B, X, content( key( KA), nonce( NA), nonce( NB))), T2)
/\ NA!=NB /\ T2 > T1.

H( send( X, B, content( key( KB), agent( A), nonce( NA))), T1)
/\ H( send( B, X, content( key( KA), nonce( NA), nonce( NB))), T2)
/\ T2 > T1
---->
E( send( X, B, content( key( KB), nonce( NB), empty( 0))), T3)
/\ T3 > T2.

```

**Fig. 3.** Social Integrity Constraints defining the Needham-Schroeder protocol.

The first SIC of Fig. 3 states that, whenever an agent  $B$  receives a message from agent  $X$ , and this message contains the name of some agent  $A$  (possibly the name of  $X$  himself), some nonce  $N_A$ , encrypted with  $B$ 's public key  $K_B$ , then a message is expected to be sent at a later time from  $B$  to  $X$ , containing the original nonce  $N_A$  and a new nonce  $N_B$ , encrypted with the public key of  $A$ .

The second SIC of Fig. 3 expresses that if two messages have been sent, with the characteristics that: *a*) the first message has been sent at the instant  $T_1$ , from  $X$  to  $B$ , containing the name of some agent  $A$  and some nonce  $N_A$ , encrypted with some public key  $K_B$ ; and *b*) the second message has been sent at a later instant  $T_2$ , from  $B$  to  $X$ , containing the original nonce  $N_A$  and a new nonce  $N_B$ , encrypted with the public key of  $A$ ; then a third message is expected to be sent from  $X$  to  $B$ , containing  $N_B$ , and encrypted with the public key of  $B$ .

```

H( send( X, Y, content( key( KY), Term1, Term2)), T0)
  /\ one_of(NX, Term1, Term2) /\ not isNonce( X, NX)
--->
E( send( V, X, content( key( KX), Term3, Term4)), T1)
  /\ X!=V /\ isPublicKey( X, KX) /\ T1 < T0
  /\ one_of( nonce(NX), Term1, Term2)
∨
E( send( V, X, content( key( KY), Term1, Term2)), T2)
  /\ T2 < T0

```

**Fig. 4.** Social Integrity Constraint expressing that an agent cannot guess a *nonce* generated by another agent (after Dolev-Yao [DY83]).

The second group of SICs consists of the one in Fig. 4, which expresses the condition that an agent is not able to guess another agent's *nonce*. The predicate  $one\_of(A, B, C)$ , defined in the  $KB_S$ , is true when  $A$  unifies with at least one of  $B$  and  $C$ . The SIC says that, if agent  $X$  sends to another agent  $Y$  a message containing a nonce that  $X$  did not create, then  $X$  must have received  $N_X$  previously in a message encrypted with  $X$ 's public key, or  $X$  must be forwarding a message that it has received.

### 3 Verification of security protocols

In this section we show the application of the *SOCS-SI* social framework to on-the-fly verification of compliance and static verification of protocol properties, adopting the well-known Needham-Schroeder security protocol as a case study.

In our approach, both types of verification are applied to the same specification of the protocol, without the need for a translation: the protocol designer, in this way, can be sure that the protocol for which he or she has verified formal properties will be the same that the agents will be required to follow.

The two types of verification are achieved by means of two abductive proof procedures, *SCIFF* and *g-SCIFF*, which are closely related. In fact, the proof procedure used for the static verification of protocol properties (*g-SCIFF*) is defined as an extension of the one used for on-the-fly verification of compliance (*SCIFF*): for this reason, we first present on-the-fly verification, although, in the intended use of *SOCS-SI*, static verification would come first.

```

h(send( a, b, content( key( kb), agent( a), nonce( na))), 1).
h(send( b, a, content( key( ka), nonce( na), nonce( nb))), 2).
h(send( a, b, content( key( kb), nonce( nb), empty( 0))), 3).

```

**Fig. 5.** A compliant history.

```

h(send( a, b, content( key( kb), agent( a), nonce( na))), 1).
h(send( b, a, content( key( ka), nonce( na), nonce( nb))), 2).

```

**Fig. 6.** A non-compliant history (the third message is missing).

### 3.1 On-the-fly verification of compliance

In this section, we show examples where the *SCIFF* proof procedure is used as a tool for verifying that the agent interaction is *compliant* (see Sect. 2.1) to a protocol.

*SCIFF* verifies compliance by trying to generate a set **EXP** which fulfils the four conditions defined in Section 2.1.

The *SCIFF* proof procedure [AGL<sup>+</sup>04] is an extension of the IFF proof procedure<sup>4</sup> [FK97]. Operationally, if the agent interaction has been compliant to the protocol, *SCIFF* reports success and the required set **EXP** of expectations (see sect. 2.1); otherwise, it reports failure. The proof procedure has been proven sound and complete with respect to the declarative semantics. A result of *termination* has also been proved, when the knowledge of the society is *acyclic*.

The following examples can be verified by means of *SCIFF*. Fig. 5 shows an example of a history compliant to the SICs of Fig. 3 and Fig. 4.

Fig. 6 instead shows an example of a history that is not compliant to such SICs. The reason is that the protocol has not been completed. In fact, the two events in the history propagate the second integrity constraints of Fig. 3 and impose an expectation

```
e(send( a, b, content( key( kb), nonce( nb), empty( 0))), T3)
```

(with the CLP constraint  $T3 > 2$ ), not fulfilled by any event in the history.

The history in Fig. 7, instead, while containing a complete protocol run, is found by *SCIFF* to violate the integrity constraints because agent **a** has used a nonce (**nc**) that it cannot know, because is not one of **a**'s nonces (as defined in the  $KB_S$ ) and **a** has not received it in any previous message. In terms of integrity constraints, the history satisfies those in Fig. 3, but it violates the one in Fig. 4.

Fig. 8 depicts Lowe's attack, which *SCIFF* finds compliant both to the protocol and to the SICs in Fig. 4.

<sup>4</sup> Extended because, unlike IFF, it copes with (i) universally quantified variables in abducibles, (ii) dynamically incoming events, (iii) consistency, fulfillment and violations, and (iv) CLP-like constraints.

```

h(send( a, b, content( key( kb), agent( a), nonce( nc))), 1).
h(send( b, a, content( key( ka), nonce( nc), nonce( nb))), 2).
h(send( a, b, content( key( kb), nonce( nb), empty( 0))), 3).

```

**Fig. 7.** A non-compliant history (agent *a* has used a nonce that it cannot hold).

```

h(send( a, i, content( key( ki), agent( a), nonce( na))), 1).
h(send( i, b, content( key( kb), agent( a), nonce( na))), 2).
h(send( b, i, content( key( ka), nonce( na), nonce( nb))), 3).
h(send( i, a, content( key( ka), nonce( na), nonce( nb))), 4).
h(send( a, i, content( key( ki), nonce( nb), empty( 0))), 5).
h(send( i, b, content( key( kb), nonce( nb), empty( 0))), 6).

```

**Fig. 8.** Lowe’s attack, recognized as a compliant history.

### 3.2 Static verification of protocol properties

In order to verify protocol properties, we have developed an extension of the SCIFF proof procedure, called *g-SCIFF*. Besides verifying whether a history is compliant to a protocol, *g-SCIFF* is able to generate a compliant history, given a protocol. *g-SCIFF* has been proved sound, which means that the histories that it generates (in case of success) are guaranteed to be compliant to the interaction protocols while entailing the goal. Note that the histories generated by *g-SCIFF* are in general not only a collection of ground events, like the sets **HAP** given as an input to SCIFF. They can, in fact, contain variables, which means that they represent *classes* of event histories.

The use of *g-SCIFF* for verification of properties can be summarised as follows.

We express properties to be verified as formulae, defined as conjunctions of literals. If we want to verify if a formula  $f$  is a property of a protocol  $\mathcal{P}$ , we express the protocol in our language and  $\neg f$  as a *g-SCIFF* goal. Two results are possible:

- *g-SCIFF* returns success, generating a history **HAP**. Thanks to the soundness of *g-SCIFF*, **HAP** entails  $\neg f$  while being compliant to  $\mathcal{P}$ :  $f$  is not a property of  $\mathcal{P}$ , **HAP** being a counterexample;
- *g-SCIFF* returns failure: this suggests that  $f$  is a property of  $\mathcal{P}$ <sup>5</sup>.

In the following, we show the automatic generation of Lowe’s attack by *g-SCIFF*, obtained as a counterexample of a property of the Needham-Schroeder protocol. The property that we want to disprove is  $\mathcal{P}_{trust}$  defined as  $trust_B(X, A) \rightarrow X = A$ , i.e., if  $B$  trusts that he is communicating with  $A$ , then he is indeed communicating with  $A$ . The notion of  $trust_B(X, A)$  is defined as follows:

**Definition 1** ( $trust_B(X, A)$ ).

<sup>5</sup> If we had a completeness result for *g-SCIFF*, this would indeed be a proof and not only a suggestion.

*B trusts that the agent X he is communicating with is A, once two messages have been exchanged at times  $T_1$  and  $T_2$ ,  $T_1 < T_2$ , having the following sender, recipient, and content:*

$$\begin{aligned} (T_1) \quad & B \rightarrow X : \{N_B, \dots\}_{pub\_key(A)} \\ (T_2) \quad & X \rightarrow B : \{N_B, \dots\}_{pub\_key(B)} \end{aligned}$$

where  $N_B$  is a nonce generated by  $B$ .

In order to check whether  $\mathcal{P}_{trust}$  is a property of the protocol, we ground  $\mathcal{P}_{trust}$  and define its negation  $\neg\mathcal{P}_{trust}$  as a goal,  $g$ , where we choose to assign to  $A$ ,  $B$ , and  $X$  the values  $a$ ,  $b$  and  $i$ :

$$\begin{aligned} g \leftarrow & isNonce(NA), NA \neq nb, \\ & \mathbf{E}(send(b, i, content(key(ka), nonce(NA), nonce(nb))), 3), \\ & \mathbf{E}(send(i, b, content(key(kb), nonce(nb), empty(0))), 6). \end{aligned}$$

This goal negates  $\mathcal{P}_{trust}$ , in that  $b$  has sent to an agent one of its nonces, encrypted with  $a$ 's public key, and has received the nonce back unencrypted, so being entitled to believe the other agent to be  $a$ ; whereas the other agent is, in fact,  $i$ .

Besides defining  $g$  for three specific agents, we also assign definite time points (3 and 6) in order to improve the efficiency of the proof by exploiting constraint propagation.

Running the g-SCIFF on  $g$  results in a compliant history:

$$\begin{aligned} \mathbf{HAP}_g = \{ & h(send(a, i, content(key(ki), agent(a), nonce(na))), 1), \\ & h(send(i, b, content(key(kb), agent(a), nonce(na))), 2), \\ & h(send(b, i, content(key(ka), nonce(na), nonce(nb))), 3), \\ & h(send(i, a, content(key(ka), nonce(na), nonce(nb))), 4), \\ & h(send(a, i, content(key(ki), nonce(nb), empty(0))), 5), \\ & h(send(i, b, content(key(kb), nonce(nb), empty(0))), 6)\}, \end{aligned}$$

that is, we generate Lowe's attack on the protocol.  $\mathbf{HAP}_g$  represents a counterexample of the property  $\mathcal{P}_{trust}$ .

## 4 Related Work

The focus of our work is not on security protocols themselves, for which there exist many efficient specialised methods, but on a language for describing protocols, for verifying the compliance of interactions, and for proving general properties of the protocols. To the best of our knowledge, this is the only comprehensive approach able to address both the two types of verification, using the same protocol definition language. Security protocols and their proof of flawedness are, in our viewpoint, instances of the general concepts of agent protocols and their properties.

In recent years the provability of properties in particular has drawn considerable attention in several communities. Various techniques have been adopted to address automatic verification of security properties. In the following, we summarise and compare some of the logic-based approaches with ours.

Russo *et al.* [RMNK02] discuss the application of abductive reasoning for analysing safety properties of declarative specifications expressed in the Event Calculus. In their abductive approach, the problem of proving that, for some invariant  $I$ , a domain description  $D$  entails  $I$  ( $D \models I$ ), is translated into an equivalent problem of showing that it is not possible to consistently extend  $D$  with assertions that particular events have actually occurred (i.e., with a set of abductive hypotheses  $\Delta$ ), in such a way that the extended description entails  $\neg I$ . In other words, there is no set  $\Delta$  such that  $D \cup \Delta \models \neg I$ . They solve this latter problem by a complete abductive decision procedure, thus exploiting abduction in a refutation mode. Whenever the procedure finds a  $\Delta$ , the assertions in  $\Delta$  act as a counterexample for the invariant. Our work is closely related: in fact, in both cases, goals represent negation of properties, and the proof procedure attempts to generate counterexamples by means of abduction. However, we rely on a different language (in particular, ours can also be used for checking compliance on the fly without changing the specification of the protocol, which is obviously a complex and time consuming task) and we deal with time by means of CLP constraints, whereas Russo *et al.* employ a temporal formalism based on Event Calculus.

In [BMV03] the authors present a new approach, On-the-Fly Model Checker, to model check security protocols, using two concepts quite related to our approach. Firstly, they introduce the concept of lazy data types for representing a (possibly) infinite transition system; secondly, they use variables in the messages that an intruder can generate. In particular, the use of unbound variables reduces the state space generated by every possible message that an intruder can utter. Protocols are represented in the form of transition rules, triggered by the arrival of a message: proving properties consists of exploring the tree generated by the transition rules, and verifying that the property holds for each reachable state. They prove results of soundness and completeness, provided that the number of messages is bounded. Our approach is very similar, at least from the operational viewpoint. The main difference is that the purpose of our language is not limited to the analysis of security protocols. Moreover, we have introduced variables in all the messages, and not only in the messages uttered by the intruder; we can pose CLP constraints on these variables, where OFMC instead can generate only equality/inequality constraints. OFMC provides state-of-the-art performance for security protocol analysis; our approach instead suffers for its generality, and its performance is definitely worse than the OFMC.

A relevant work in computer science on verification of security protocols was done by Abadi and Blanchet [Bla03,AB05]. They adopt a verification technique based on logic programming in order to verify security properties of protocols, such as secrecy and authenticity in a fully automatic way, without bounding the number of sessions. In their approach, a protocol is represented in extensions of pi calculus with cryptographic primitives. The protocol represented in this extended calculus is then automatically translated into a set of Horn clauses [AB05]. To prove secrecy, in [Bla03,AB05] attacks are modelled by relations and secrecy can be inferred by non-derivability: if  $attacker(M)$  is not derivable, then secrecy

of  $M$  is guaranteed. More importantly, the derivability of  $attacker(M)$  can be used, instead, to reconstruct an attack. This approach was later extended in [Bla02] in order to prove authenticity. By first order logic, having variables in the representation, they overcome the limitation of bounding the number of sessions. We achieve the same generality of [Bla03,AB05], since in their approach Horn clause verification technique is not specific to any formalism for representing the protocol, but a proper translator from the protocol language to Horn clause has to be defined. In our approach, we preferred to directly define a rewriting proof procedure ( $\mathcal{SCIFF}$ ) for the protocol representation language. Furthermore, by exploiting abduction and CLP constraints, also in the implementation of g- $\mathcal{SCIFF}$  transitions themselves, in our approach we are able to generate proper traces where terms are constrained when needed along the derivation avoiding to impose further parameters to names as done in [AB05]. CLP constraints can do this more easily.

Armando *et al.* [ACL04] compile a security program into a logic program with choice lp-rules with answer set semantics. They search for attacks of length  $k$ , for increasing values of  $k$ , and they are able to derive the flaws of various flawed security protocols. They model explicitly the capabilities of the intruder, while we take the opposite viewpoint: we explicitly state what the intruder cannot do (like decrypting a message without having the key, or guessing the key or the nonces of an agent), without implicitly limiting the abilities of the intruder.

Our social specifications can be seen as intensional formulations of the possible (i.e., compliant) traces of communication interactions. In this respect, our way of modeling protocols is very similar to the one of Paulson's inductive approach [Pau98]. In particular, our representation of the events is almost the same, but we explicitly mention time in order to express temporal constraints. In the inductive approach, the protocol steps are modeled as possible extensions of a trace with new events and represented by (forward) rules, similar to our SICs. However, in our system we have expectations, which allow us to cope with both compliance on the fly and verification of properties without changing the protocol specification. Moreover, SICs can be considered more expressive than inductive rules, since they deal with constraints (and constraint satisfaction in the proof), and disjunctions in the head. As far as verification, the inductive approach requires more human interaction and expertise, since it exploits a general purpose theorem prover, and has the disadvantage that it cannot generate counterexamples directly (as most theorem prover-based approaches). Instead, we use a specialized proof procedure based on abduction that can perform the proof without any human intervention, and can generate counterexamples.

Millen and Shmatikov [MS01] define a proof-procedure based on constraint solving for cryptographic protocol analysis. Their proof-procedure is sound and complete. Corin and Etalle [CE02] improve the system by Millen and Shmatikov by adding explicit checks and considering partial runs of the protocols; the resulting system is considerably faster. g- $\mathcal{SCIFF}$  is based on constraint solving as well, but with a different flavour of constraint: while the approaches by Millen and Shmatikov and by Corin and Etalle are based on abstract algebra, our con-

straint solver comprises a CLP(FD) solver, and embeds constraint propagation techniques to speed-up the solving process.

In [Son99], Song presents Athena, an approach to automatic security protocol analysis. Athena is a very efficient technique for proving protocol properties: unlike other techniques, Athena copes well with state space explosion and is applicable with an unbounded number of peers participating in a protocol, thanks to the use of theorem proving and to a compact way to represent states. Athena is correct and complete (but termination is not guaranteed). Like Athena, the representation of states and protocols in g-SCIFF is non ground, and therefore general and compact. Unlike Athena's, the implementation of g-SCIFF is not optimised, and suffers from the presence of symmetrical states. On the other hand, a clear advantage of the SOCS approach is that protocols are written and analyzed in a formalism which is the same used for run-time verification of compliance.

## 5 Conclusion and future work

In this paper, we have shown how the *SOCS-SI* abductive framework can be applied to the specification and verification of security protocols, using, as a running example, the Needham-Schroeder Public Key authentication protocol.

The declarative framework is expressive enough to specify both which sequences of messages represent a legal protocol run, and constraints about the messages that a participant is able to synthesize.

We have implemented and experimented with the operational counterpart of the *SOCS-SI* framework. Two kinds of verification are possible on a protocol specified in the *SOCS-SI* formalism: on-the-fly verification of compliance at runtime (by means of the sound and complete *SCIFF* proof procedure), and static verification of protocol properties at design time (by means of the sound g-*SCIFF* proof procedure). In this way, our approach tackles both the case of agents misbehaving (which, in an open society, cannot be excluded) and the case of a flawed protocol (which can make the interaction exhibit an undesirable feature even if the participants follow the protocol correctly). The refutation of a protocol property is given by means of a counterexample. Both types of verification are automatic, and require no human intervention.

We believe that one of the main contributions of this work consists of providing a unique framework to both the two types of verification. The language used for protocol definition is the same in both the cases, thus lowering the chances of errors introduced in describing the protocols with different notations. The protocol designer can benefit of our approach during the design phase, by proving properties, and during the execution phase, where the interaction can be proved to be compliant with the protocol, and thus to exhibit the protocol properties.

Future work will be aimed to prove a result of completeness for g-*SCIFF*, and to extend the experimentation on proving protocol properties to a number of security and e-commerce protocols, such as NetBill [CTS95] and SPLICE/AS [YOM91].

## Acknowledgments

This work has been supported by the European Commission within the SOCS project (IST-2001-32530), funded within the Global Computing Programme and by the MIUR COFIN 2003 projects *La Gestione e la negoziazione automatica dei diritti sulle opere dell'ingegno digitali: aspetti giuridici e informatici* and *Sviluppo e verifica di sistemi multiagente basati sulla logica*.

## References

- [AB05] Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. *J. ACM*, 52(1):102–146, 2005.
- [ACG<sup>+</sup>03] M. Alberti, A. Ciampolini, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. A social ACL semantics by deontic constraints. In V. Mařík, J. Müller, and M. Pěchouček, editors, *Multi-Agent Systems and Applications III. Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003*, volume 2691 of *Lecture Notes in Artificial Intelligence*, pages 204–213, Prague, Czech Republic, June 16–18 2003. Springer-Verlag.
- [ACL04] Alessandro Armando, Luca Compagna, and Yuliya Lierler. Automatic compilation of protocol insecurity problems into logic programming. In José Júlio Alferes and João Alexandre Leite, editors, *Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lisbon, Portugal, September 27-30, 2004, Proceedings*, volume 3229 of *Lecture Notes in Artificial Intelligence*, pages 617–627. Springer-Verlag, 2004.
- [AGL<sup>+</sup>03a] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. An Abductive Interpretation for Open Societies. In A. Cappelli and F. Turini, editors, *AI\*IA 2003: Advances in Artificial Intelligence, Proceedings of the 8th Congress of the Italian Association for Artificial Intelligence, Pisa*, volume 2829 of *Lecture Notes in Artificial Intelligence*, pages 287–299. Springer-Verlag, September 23–26 2003.
- [AGL<sup>+</sup>03b] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Specification and verification of agent interactions using social integrity constraints. *Electronic Notes in Theoretical Computer Science*, 85(2), 2003.
- [AGL<sup>+</sup>04] Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Abduction with hypothesis confirmation. Number 390 in *Quaderno del Dipartimento di Matematica, Research Report*. Università di Parma, November 2004.
- [Bla02] Bruno Blanchet. From secrecy to authenticity in security protocols. In *SAS '02: Proceedings of the 9th International Symposium on Static Analysis*, pages 342–359, London, UK, 2002. Springer-Verlag.
- [Bla03] Bruno Blanchet. Automatic verification of cryptographic protocols: a logic programming approach. In *PPDP '03: Proceedings of the 5th ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 1–3, New York, NY, USA, 2003. ACM Press.
- [BMV03] David A. Basin, Sebastian Mödersheim, and Luca Viganò. An on-the-fly model-checker for security protocol analysis. In Einar Snekkenes and Dieter Gollmann, editors, *ESORICS*, volume 2808 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2003.

- [CE02] Ricardo Corin and Sandro Etalle. An improved constraint-based system for the verification of security protocols. In Manuel V. Hermenegildo and German Puebla, editors, *Static Analysis, 9th International Symposium, SAS 2002, Madrid, Spain, September 17-20, 2002, Proceedings*, volume 2477 of *Lecture Notes in Computer Science*, pages 326–341, Berlin, Germany, 2002. Springer.
- [CTS95] B. Cox, J.C. Tygar, and M. Sirbu. Netbill security and transaction protocol. In *Proceedings of the First USENIX Workshop on Electronic Commerce*, New York, July 1995.
- [DY83] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.
- [FK97] T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 33(2):151–165, November 1997.
- [GP02] F. Guerin and J. Pitt. Proving properties of open agent systems. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II*, pages 557–558, Bologna, Italy, July 15–19 2002. ACM Press.
- [JM94] J. Jaffar and M.J. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19-20:503–582, 1994.
- [KKT93] A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1993.
- [Low96] G. Lowe. Breaking and fixing the Needham-Shroeder public-key protocol using CSP and FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems: Second International Workshop, TACAS'96*, volume 1055 of *Lecture Notes in Artificial Intelligence*, pages 147–166. Springer-Verlag, 1996.
- [MS01] Jonathan K. Millen and Vitaly Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 166–175. ACM press, 2001.
- [NS78] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.
- [Pau98] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.
- [RMNK02] A. Russo, R. Miller, B. Nuseibeh, and J. Kramer. An abductive approach for analysing event-based requirements specifications. In P.J. Stuckey, editor, *Logic Programming, 18th International Conference, ICLP 2002*, volume 2401 of *Lecture Notes in Computer Science*, pages 22–37, Berlin Heidelberg, 2002. Springer-Verlag.
- [SOCa] Societies Of Computees (SOCS): a computational logic model for the description, analysis and verification of global and open societies of heterogeneous computees. IST-2001-32530. Home Page: <http://lia.deis.unibo.it/Research/SOCS/>.
- [SOCb] The SOCS Protocol Repository. Available at <http://lia.deis.unibo.it/research/socs/partners/societies/protocols.html>.
- [Son99] Dawn Xiaodong Song. Athena: a new efficient automatic checker for security protocol analysis. In *CSFW '99: Proceedings of the 1999 IEEE Com-*

*puter Security Foundations Workshop*, page 192, Washington, DC, USA, 1999. IEEE Computer Society.

- [YOM91] Suguru Yamaguchi, Kiyohiko Okayama, and Hideo Miyahara. The design and implementation of an authentication system for the wide area distributed environment. *IEICE Transactions on Information and Systems*, E74(11):3902–3909, November 1991.