Security Protocols Verification in Abductive Logic Programming: A Case Study*

Marco Alberti[1], Federico Chesani[2], Marco Gavanelli[1], Evelina Lamma[1], Paola Mello[2], and Paolo Torroni[2]

[1] ENDIF, Università di Ferrara - Via Saragat, 1 - 44100 Ferrara, Italy
{marco.alberti, marco.gavanelli, evelina.lamma}@unife.it
[2] DEIS, Università di Bologna - Viale Risorgimento, 2 - 40126 Bologna, Italy
{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).

In order to explain the approach, 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.

We also demonstrate the use of the *SOCS-SI* framework for the static verification of the NetBill e-commerce protocol.

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

* This paper is a revised version of work discussed at the Twentieth Italian Symposium on Computational Logic, CILC 2005, whose informal proceedings are available from the URL: http://www.disp.uniroma2.it/CILC2005/

O. Dikenelli, M.-P. Gleizes, and A. Ricci (Eds.): ESAW 2005, LNAI 3963, pp. 106–124, 2006.

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. By "well defined", we mean that they guarantee some desirable properties (assuming that agents act according to them). 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*.

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" [1]. Open agent societies are a useful setting for heterogenous 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. In order to know whether the desired "static" properties hold at run-time, we need to be able to verify that agents do follow the protocols. In other words, we can do what Guerin and Pitt call *on-the-fly verification of compliance* [2]. 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 [3] 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 over-constrain the agents' behaviour. We have shown that *SOCS-SI* is suitable for semantic specification of agent communication languages [4], and that it lends itself to the definition of a range of agent interaction protocols [5].[1]

In this paper, we demonstrate by a case study on the well known Needham-Schroeder security protocol [7] 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 ($\mathcal{S}$CIFF and g-$\mathcal{S}$CIFF). 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 error-prone translation step.

SOCS-SI can thus be viewed as a tool for protocol designers, which can be used to 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

[1] A repository of protocols is available on the web [6].

of compliance and static verification of properties of the protocol. In Sect. 4, as a further example, we propose the static verification of the NetBill e-commerce protocol. Related work and conclusions follow.

2 Specifying the Needham-Schroeder Public Key Encryption Protocol

In this section, we show how the *SOCS-SI* framework can be used to represent the well-known Needham-Schroeder security protocol [7]. The purpose of the protocol is to ensure mutual authentication while maintaining secrecy. In other words, once agents A and B have successfully completed a run of the protocol, A should believe his partner to be B if and only if B believes his partner to be A.

(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.

$A \rightarrow B : \langle M \rangle_{PK}$ means that A has sent to B a message M, encrypted with the key PK. A message of form N_X represents a *nonce*: a message whose content is assumed impossible to guess (such as a long binary string), and thus known only to the agent that synthesized it and to those who received it.

In step (1), A sends to B a new nonce N_A, together with A's identifier, encrypted with B's public key. In step (2), B sends N_A back to A, together with a new nonce N_B, encrypted with A's public key. A is now sure about B's identity, since only B can have decrypted the first message and know N_A. Similarly, B is sure about A's identity after step (3), because only A can have decrypted the second message and have read N_B to send it back to B.

At the end of the protocol, seemingly, A and B are mutually authenticated.

Lowe's attack on the protocol. Eighteen years after the publication of the Needham-Schroeder protocol, Lowe [8] 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.

It is important to notice that Lowe's attack is effective even if the nonces and keys are not compromised, differently from other kinds of attack (see, for instance, those exemplified by Denning and Sacco [9]).

(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

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 [3][2] to specify interaction protocols for open societies of agents in a declarative way.

Since in open societies the agents' internal state is not observable, the *SOCS-SI* framework is aimed at specifying and verifying the agents' observable behaviour. The verification is performed by an external entity, the *social infrastructure*, which can observe the agent behaviour.

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

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

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,

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

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

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

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 [11] constraints can be imposed on the variables. This is because the desired agent behaviour may be underspecified (hence variables), yet subject to restriction (hence CLP constraints).

For instance,

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

[2] The reader can refer to [10] for a more detailed description.

could represent the expectation for agent a to send to agent b a message consisting of a nonce (n_b) and an empty part ($empty(0)$), 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 $\neg\mathbf{EN}$).

In the *SOCS-SI* framework, the agent interaction is specified by means of interaction protocols.

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 logic frameworks [12], *abducibles* represent hypotheses, a logic program specifies which set of hypotheses entail a goal, and integrity constraints rule out inconsistent set of hypotheses. The abductive reasoning is successful if it finds a set of hypotheses which entail the goal while not violating the integrity constraints.

In our (abductive) framework, we map expectations to abducibles, and the abductive semantics is used to select a desired behaviour which entails a social goal, while not violating the SICs. In addition, we require the desired behaviour to be matched by the actual agent behaviour.

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

- $\mathcal{IC}_S$*-consistent*: it must entail $\mathcal{IC}_S$, for the given $\mathcal{S}$ and **HAP**;
- $\neg$*-consistent*: for any ground p, **EXP** cannot include $\{\mathbf{E}(P), \neg\mathbf{E}(p)\}$ or $\{\mathbf{EN}(p), \neg\mathbf{EN}(p)\}$ (this requirement implements explicit negation for expectations);
- **E***-consistent*: for any ground p, **EXP** cannot include $\{\mathbf{E}(p), \mathbf{EN}(p)\}$ (an event cannot be both expected to happen and expected not to happen);
- *fulfilled*: for any ground p, **EXP** cannot contain $\mathbf{EN}(p)$ if **HAP** contains $\mathbf{H}(p)$, and **EXP** cannot contain $\mathbf{E}(p)$ if **HAP** does not contain $\mathbf{H}(p)$ (happened events are required to match the expectations).

In order to support goal-oriented societies, **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

In the following, we show a specification of the Needham-Schroeder protocol in the *SOCS-SI* language.

With the atom:

$$\mathbf{H}(send(X, Y, content(key(K), Term1, Term_2)), T_1)$$

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 $Term_1$ and $Term_2$ and has been encrypted with the key K. T_1 is the time at which Y receives the message.

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

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

A first group of SICs, depicted in Figure 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 Figure 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), and 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 Figure 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 B's public key.

The second group of SICs consists of the one in Figure 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

```
    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 [13])

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 Needham-Schroeder security protocol, specified in 2.2 as a case study.

By "static verification of protocol properties" we mean a verification (by means of a formal proof, performed at design time) that a protocol enjoyes desirable properties. If the agents follow the protocol, then the agent interaction will itself exhibit the properties. However, since in open agent societies it cannot be assumed that the agents will follow the protocols, it becomes necessary to verify the agents' compliance to the protocol by means of an external trusted entity, able to observe the agent behaviour at runtime. Following Guerin and Pitt [2], we call this process "on-the-fly verification of compliance".

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, $\mathcal{S}$CIFF and g-$\mathcal{S}$CIFF, which are closely related. In fact, the proof-procedure used for the static verification of protocol properties (g-$\mathcal{S}$CIFF) is defined as an extension of the one used for on-the-fly verification of compliance ($\mathcal{S}$CIFF): for this reason, we first present on-the-fly verification, although, in the intended use of *SOCS-SI*, static verification would come first.

3.1 On-the-Fly Verification of Compliance

In this section, we show examples where the $\mathcal{S}$CIFF proof-procedure is used as a tool for verifying that the agent interaction is *compliant* to a protocol.

```
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)

$\mathcal{S}$CIFF verifies compliance by trying to generate a set **EXP** which fulfils the four conditions defined in Section 2.1.

The $\mathcal{S}$CIFF proof-procedure [14] is an extension of the IFF proof-procedure[3] [15]. Operationally, if the agent interaction has been compliant to the protocol, $\mathcal{S}$CIFF reports success and the required set **EXP** of expectations; otherwise, it reports failure. The proof-procedure has been proven sound and complete with respect to the declarative semantics. A result of termination also holds, under acyclicity assumptions.

The following examples can be verified by means of $\mathcal{S}$CIFF. Figure 5 shows an example of a history compliant to the SICs of Figure 3 and Figure 4.

Figure 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 Figure 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 Figure 7, instead, while containing a complete protocol run, violates the integrity constraint of Figure 4 because agent `a` has used a nonce (`nc`) that it cannot know, being not one of its own nonces (as defined in the KB_S), nor one of those `a` received in any previous message (or better, we have no evidence of it). In terms of integrity constraints, the history satisfies those in Figure 3, but it violates the one in Figure 4.

Based on $\mathcal{S}$CIFF, *SOCS-SI* is able to capture at run-time violation cases such as these.

Figure 8 depicts Lowe's attack, which is compliant both to the protocol and to the SICs in Figure 4.

A number of experiments made on a number of protocols can be downloaded from the SOCS Protocol Repository [6].

[3] 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 $\mathcal{S}$CIFF proof-procedure, called g-$\mathcal{S}$CIFF. Besides verifying whether a history is compliant to a protocol, g-$\mathcal{S}$CIFF is able to generate a compliant history, given a protocol. g-$\mathcal{S}$CIFF has been proved sound [16], 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-$\mathcal{S}$CIFF are in general not only a collection of ground events, like the **HAP** sets given as an input to $\mathcal{S}$CIFF. They can, in fact, contain variables, which means that they represent *classes* of event histories.

In order to use g-$\mathcal{S}$CIFF for verification, we express the property to be verified as a conjunction 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-$\mathcal{S}$CIFF goal. Then either:

- g-$\mathcal{S}$CIFF returns success, generating a history **HAP**. Thanks to the soundness of g-$\mathcal{S}$CIFF, **HAP** entails $\neg f$ while being compliant to $\mathcal{P}$: f is not a property of $\mathcal{P}$, **HAP** being a counterexample; or
- g-$\mathcal{S}$CIFF returns failure, suggesting that f is a property of $\mathcal{P}$[4].

In the following, we exemplify such a use of g-$\mathcal{S}$CIFF by showing the automatic generation of Lowe's attack by g-$\mathcal{S}$CIFF, 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.

Thanks to the properties of public keys (a message encrypted with a public key can only be decrypted by the owner of the corresponding private key) and nonces (a nonce cannot be guessed), the notion of $trust_B(X, A)$ can be characterized as follows:

[4] If we had a completeness result for g-$\mathcal{S}$CIFF, this would indeed be a proof and not only a suggestion.

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

$$(T_1)\ B \rightarrow X : \{N_B, \dots\}_{pub\_key(A)}$$
$$(T_2)\ X \rightarrow B : \{N_B, \dots\}_{pub\_key(B)}$$

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:

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

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-$\mathcal{S}$CIFF on g results in a compliant history:

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

that is, we generate Lowe's attack on the protocol.

$\mathbf{HAP}_g$ represents a counterexample which shows that the Needham-Schroeder protocol does not have the property $\mathcal{P}_{trust}$, being a history that is compliant to the protocol while violating the property.

4 Verifying the NetBill Protocol

In this section, we further demonstrate the specification and verification of agent interaction protocols in the *SOCS-SI* framework, on the NetBill (see [17]) protocol.

NetBill is a security and transaction protocol optimized for the selling and delivery of low-priced information goods, like software or journal articles. The protocol rules transactions between two agents: *merchant* and *customer*. A NetBill server is used to deal with financial issues such as those related to credit card accounts of customer and merchant.

In the following, we focus on the type of the NetBill protocol designed for non zero-priced goods, and do not consider the variants that deal with zero-priced goods.

A typical protocol run is composed of three phases:

1. *price negotiation.* The customer requests a quote for a good identified by *PrId* (`priceRequest(PrId)`), and the merchant replies with (`priceQuote(PrId,Quote)`).
2. *good delivery.* The customer requests the good (`goodRequest(PrId,Quote)`) and the merchant delivers it in an encrypted format (`deliver(crypt(PrId,Key),Quote)`).
3. *payment.* The customer issues an Electronic Payment Order (EPO) to the merchant, for the amount agreed for the good (`payment(epo(C,crypt(PrId,K),Quote))`); the merchant appends the decryption key for the good to the EPO, signs the pair and forwards it to the NetBill server (`endorsedEPO(epo(C,crypt(PrId,K),Quote),M)`); the NetBill server deals with the actual money transfer and returns the result to the merchant (`signedResult(C,PrID,Price,K)`), who will, in her turn, send a receipt for the good and the decryption key to the customer (`receipt(PrId,Price,K)`).

The customer can withdraw from the transaction until she has issued the *EPO* message; the merchant until she has issued the *endorsedEPO* message.

4.1 NetBill Protocol Specification in *SOCS-SI*.

The NetBill protocol is implemented in the *SOCS-SI* framework by means of SICs of two types:

- *backward integrity constraints* (Fig. 9), i.e., integrity constraints that state that if some set of event happens, then some other set of event is expected to have happened before.

  For instance, the first backward integrity constraints imposes that, if `M` has sent a `priceQuote` message to `C`, stating that `M`'s quote for the good identified by `PrId` is `Quote`, in the interaction identified by `Id`, then `C` is expected to have sent to `M` a `priceRequest` message for the same good, in the same interaction, at an earlier time.
- *forward integrity constraints* (Fig. 10), i.e., constraints that state that if some conjunction of event has happened, then some other set of event is expected to happen in the future.

  For instance, the first forward integrity constraint in Fig. 10 imposes that an `endorsedEPO` message from `M` to the *netbill* server be followed by a `signedResult` message, with the corresponding parameters.

We only impose forward constraints from the `endorsedEPO` message onwards, because both parties (merchant and customer) can withdraw from the transaction at the previous steps.

```
H(tell(M,C,priceQuote(PrId,Quote),Id),T)
--->
E(tell(C,M,priceRequest(PrId),Id),T2) /\ T2 < T.

H(tell(C,M,goodRequest(PrId,Quote),Id),T)
--->
E(tell(M,C,priceQuote(PrId,Quote),Id),Tpri) /\ Tpri < T.

H(tell(M,C,goodDelivery(crypt(PrId,K),Quote),Id),T)
--->
E(tell(C,M,goodRequest(PrId,Quote),Id),Treq) /\ Treq < T.

H(tell(C,M,payment(C,crypt(PrId,K),Quote),Id),T)
--->
E(tell(M,C,goodDelivery(crypt(PrId,K),Quote),Id),Tdel) /\ Tdel <
T.

H(tell(netbill,M,signedResult(C,PrId,Quote,K),Id),Tsign)

/\ M != netbill
--->
E(tell(M,netbill,endorsedEPO(epo(C,PrId,Quote),K,M),Id),T) /\ T
< Tsign.

H(tell(M,C,receipt(PrId,Quote,K),Id),Ts)
--->
E(tell(netbill,M,signedResult(C,PrId,Quote,K),Id),Tsign) /\
Tsign < Ts.
```

Fig. 9. NetBill protocol: backward integrity constraints

```
H(tell(M,netbill,endorsedEPO(epo(C,PrId,Quote),K,M),Id),T)
--->
E(tell(netbill,M,signedResult(C,PrId,Quote,K),Id),Tsign) /\ T <
Tsign.

H(tell(netbill,M,signedResult(C,PrId,Quote,K),Id),Tsign)
--->
E(tell(M,C,receipt(PrId,Quote,K),Id),Ts) /\ Tsign < Ts.
```

Fig. 10. NetBill protocol: forward integrity constraints

4.2 Verification of NetBill Properties

In this section, we show how a simple property of the NetBill protocol can be expressed, and verified, in the *SOCS-SI* framework.

We want to verify the following property: *the merchant receives the payment for a good G if and only if the customer receives the good G*, as long as the protocol is respected.

Since the $\mathcal{S}$CIFF deals with (communicative) events and not with the states of the agents, we need to express the properties in terms of happened events. To this purpose, we can assume that merchant has received the payment once the NetBill server has issued the *signedResult* message, and that the the customer has received the good if she has received the encrypted good (with a *deliver* message) and the encryption key (with a *receipt* message).

Thus, the property that we want to verify can be espressed as

$$\begin{aligned} & \mathbf{H}(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign) \\ \Longleftrightarrow\ & \mathbf{H}(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T) \\ & \wedge \mathbf{H}(tell(M, C, receipt(PrId, Quote, K), Id), Ts) \end{aligned} \tag{1}$$

whose negation is

$$\begin{aligned} & (\neg \mathbf{H}(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign) \\ & \wedge \mathbf{H}(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T) \\ & \wedge \mathbf{H}(tell(M, C, receipt(PrId, Quote, K), Id), Ts)) \\ & \vee \\ & (\mathbf{H}(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign) \\ & \wedge \neg \mathbf{H}(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T) \\ & \vee \\ & (\mathbf{H}(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign) \\ & \wedge \neg \mathbf{H}(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T)) \end{aligned} \tag{2}$$

In other words, a history that entails Eq. (2) is a counterexample of the property that we want to prove. In order to search for such a history, we define a g-$\mathcal{S}$CIFF goal as follows:

$$\begin{aligned} g \leftarrow & \mathbf{EN}(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign), \\ & \mathbf{E}(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T), \\ & \mathbf{E}(tell(M, C, receipt(PrId, Quote, K), Id), Ts)). \\ g \leftarrow & \mathbf{E}(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign), \\ & \mathbf{EN}(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T). \\ g \leftarrow & \mathbf{E}(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign), \\ & \mathbf{EN}(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T)) \end{aligned} \tag{3}$$

and run g-$\mathcal{S}$CIFF.

The result of the call is a failure. This suggests that there is no history that entails the negation of the property while respecting the protocol, i.e., the property is likely to hold if the protocol is respected. However, yet no guarantee can be given, because g-$\mathcal{S}$CIFF has not been proven complete.

If we remove the second forward integrity constraints (which imposes that a `signedResult` message be followed by a `receipt` message), then g-$\mathcal{S}$CIFF reports success, and the following history is generated:

```
h(tell(_E,_F,priceRequest(_D),_C),_M),
h(tell(_F,_E,priceQuote(_D,_B),_C),_L),
h(tell(_E,_F,goodRequest(_D,_B),_C),_K),
h(tell(_F,_E,goodDelivery(crypt(_D,_A),_B),_C),_J),
h(tell(_E,_F,payment(_E,crypt(_D,_A),_B),_C),_I),
h(tell(_F,netbill,endorsedEPO(epo(_E,_D,_B),_A,_F),_C),_H),
h(tell(netbill,_F,signedResult(_E,_D,_B,_A),_C),_G),
_I<_H, _H<_G,
_L>_M, _K>_L, _I>_J, _J>_K,
```

The *receipt* event is missing, which would violate the integrity constraint that has been removed. In other words, without that integrity constraint, the protocol no longer has the desired property.

In this way, a protocol designer can make sure that an integrity constraint is not redundant with respect to a desired property of the protocol.

5 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 first comprehensive and fully operational approach addressing both types of verification, and using the same protocol definition language in both cases. Security protocols and their proof of flawedness are, in our viewpoint, instances of the general concepts of agent protocols and their properties.

However, in this section we will discuss some related logic-based approaches to automatic verification of security properties.

Russo *et al.* [18] 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 Δ), in such a way that the extended description entails $\neg I$. In other words, there is no set Δ 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 such a set Δ, the assertions in Δ 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 a demanding task) and we deal with time by means of CLP constraints, whereas Russo *et al.* employ a temporal formalism based on Event Calculus.

In [19] the authors present a new approach, On-the-Fly Model Checker, to model check security protocols, using two concepts quite related to our approach:

the concept of lazy data types for representing a (possibly) infinite transition system, and the use of 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, 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, whereas OFMC can only generate equality/inequality constraints. On the downside, 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 [20, 21]. 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 [21]. To prove secrecy, in [20, 21] 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 [22] 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 [20, 21], 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 ($\mathcal{S}$CIFF) for the protocol representation language. Furthermore, by exploiting abduction and CLP constraints, also in the implementation of g-$\mathcal{S}$CIFF 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 [21]. CLP constraints can do this more easily.

Armando *et al.* [23] 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 [24]. 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 [25] define a sound and complete proof-procedure, later improved by Corin and Etalle [26], based on constraint solving for cryptographic protocol analysis. g-$\mathcal{S}$CIFF 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 constraint solver comprises a CLP(FD) solver, and embeds constraint propagation techniques to speed-up the solving process.

In [27], 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 g-SCIFF's implementation 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.

Özkohen and Yolum [28] propose an approach for the prediction of exceptions in supply chains which builds upon the well-known commitment-based approach for protocol specification (see, for instance, Yolum and Singh [29]); their approach is related in many aspects to our on-the-fly verification. They represent the expected agent behaviour by means of commitments between agents; commitments have timeouts, i.e., they must be fulfilled by a deadline, and can be composed by means of conjunction and disjunction. In this perspective,

commitments are similar to our expectations, which can have deadlines represented by CLP constraints, and which are composed in disjunctions of conjunctions in the head of the social integrity constraints. However, our expectations can regard any kind of events expected to happen, not only those that can be represented as a commitment of a debtor towards a creditor; and we can also represent negative expectations. Operationally, in [28] the reasoning about commitments is centralized in a monitoring agents; in our framework, a similar task in performed by the social infrastructure.

6 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.

Based on the *SOCS-SI* framework, we have implemented and experimented with two kinds of automatic verification: on-the-fly verification of compliance (by means of the sound and complete $\mathcal{S}$CIFF proof-procedure), and static verification of protocol properties (by means of the sound g-$\mathcal{S}$CIFF 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).

We believe that the main contribution 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 the protocol translation from one notation to a different one. 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 investigate a result of completeness for g-$\mathcal{S}$CIFF, and to extend the experimentation on proving protocol properties to a number of security and e-commerce protocols, such as SPLICE/AS [30].

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

1. Davidsson, P.: Categories of artificial societies. In Omicini, A., Petta, P., Tolksdorf, R., eds.: Engineering Societies in the Agents World II. Volume 2203 of Lecture Notes in Artificial Intelligence, Springer-Verlag (2001) 1–9 2nd International Workshop (ESAW'01), Prague, Czech Republic, July 7, 2001, Revised Papers
2. Guerin, F., Pitt, J.: Proving properties of open agent systems. In Castelfranchi, C., Lewis Johnson, W., eds.: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II, Bologna, Italy, ACM Press (2002) 557–558
3. 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 (2001) Home Page: http://lia.deis.unibo.it/Research/SOCS/
4. Alberti, M., Ciampolini, A., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: A social ACL semantics by deontic constraints. In Mařík, V., Müller, J., Pěchouček, M., eds.: 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, Prague, Czech Republic, Springer-Verlag (2003) 204–213
5. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Specification and verification of agent interactions using social integrity constraints. Electronic Notes in Theoretical Computer Science **85** (2003)
6. The socs protocol repository (2005) Available at http://lia.deis.unibo.it/research/socs/partners/societies/protocols.html
7. Needham, R., Schroeder, M.: Using encryption for authentication in large networks of computers. Communications of the ACM **21** (1978) 993–999
8. Lowe, G.: Breaking and fixing the Needham-Shroeder public-key protocol using CSP and FDR. In Margaria, T., Steffen, B., eds.: Tools and Algorithms for the Construction and Analysis of Systems: Second International Workshop, TACAS'96. Volume 1055 of Lecture Notes in Artificial Intelligence, Springer-Verlag (1996) 147–166
9. Denning, D.E., Sacco, G.M.: Timestamps in key distribution protocols. Communications of the ACM **24** (1981) 533–536
10. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: An Abductive Interpretation for Open Societies. In Cappelli, A., Turini, F., eds.: 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, Springer-Verlag (2003) 287–299
11. Jaffar, J., Maher, M.: Constraint logic programming: a survey. Journal of Logic Programming **19–20** (1994) 503–582
12. Kakas, A.C., Kowalski, R.A., Toni, F.: Abductive Logic Programming. Journal of Logic and Computation **2** (1993) 719–770
13. Dolev, D., Yao, A.C.C.: On the security of public key protocols. IEEE Transactions on Information Theory **29** (1983) 198–207
14. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: The sciff abductive proof-procedure. In: Proceedings of the 9th National Congress on Artificial Intelligence, AI*IA 2005. Volume 3673 of Lecture Notes in Artificial Intelligence, Springer-Verlag (2005) 135–147

15. Fung, T.H., Kowalski, R.A.: The IFF proof procedure for abductive logic programming. Journal of Logic Programming **33** (1997) 151–165
16. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: On the automatic verification of interaction protocols using g-$\mathcal{S}$CIFF. Technical Report DEIS-LIA-04-004, University of Bologna (Italy) (2005) LIA Series no. 72.
17. Cox, B., Tygar, J., Sirbu, M.: Netbill security and transaction protocol. In: Proceedings of the First USENIX Workshop on Electronic Commerce, New York (1995)
18. Russo, A., Miller, R., Nuseibeh, B., Kramer, J.: An abductive approach for analysing event-based requirements specifications. In Stuckey, P., ed.: Logic Programming, 18th International Conference, ICLP 2002. Volume 2401 of Lecture Notes in Computer Science, Berlin Heidelberg, Springer-Verlag (2002) 22–37
19. Basin, D.A., Mödersheim, S., Viganò, L.: An on-the-fly model-checker for security protocol analysis. In Snekkenes, E., Gollmann, D., eds.: ESORICS. Volume 2808 of Lecture Notes in Computer Science, Springer (2003) 253–270
20. Blanchet, B.: 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 declaritive programming, New York, NY, USA, ACM Press (2003) 1–3
21. Abadi, M., Blanchet, B.: Analyzing security protocols with secrecy types and logic programs. J. ACM **52** (2005) 102–146
22. Blanchet, B.: From secrecy to authenticity in security protocols. In: SAS '02: Proceedings of the 9th International Symposium on Static Analysis, London, UK, Springer-Verlag (2002) 342–359
23. Armando, A., Compagna, L., Lierler, Y.: Automatic compilation of protocol insecurity problems into logic programming. In Alferes, J.J., Leite, J.A., eds.: Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lisbon, Portugal, September 27–30, 2004, Proceedings. Volume 3229 of Lecture Notes in Artificial Intelligence, Springer-Verlag (2004) 617–627
24. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security **6** (1998) 85–128
25. Millen, J.K., Shmatikov, V.: Constraint solving for bounded-process cryptographic protocol analysis. In: CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, ACM press (2001) 166–175
26. Corin, R., Etalle, S.: An improved constraint-based system for the verification of security protocols. In Hermenegildo, M.V., Puebla, G., eds.: Static Analysis, 9th International Symposium, SAS 2002, Madrid, Spain, September 17–20, 2002, Proceedings. Volume 2477 of Lecture Notes in Computer Science, Berlin, Germany, Springer (2002) 326–341
27. Song, D.X.: Athena: a new efficient automatic checker for security protocol analysis. In: CSFW '99: Proceedings of the 1999 IEEE Computer Security Foundations Workshop, Washington, DC, USA, IEEE Computer Society (1999) 192
28. Özkohen, A., Yolum, P.: Predicting exceptions in agent-based supply chains. In this volume. (2006)
29. Yolum, P., Singh, M.: Flexible protocol specification and execution: applying event calculus planning using commitments. In Castelfranchi, C., Lewis Johnson, W., eds.: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II, Bologna, Italy, ACM Press (2002) 527–534
30. Yamaguchi, S., Okayama, K., Miyahara, H.: The design and implementation of an authentication system for the wide area distributed environment. IEICE Transactions on Information and Systems **E74** (1991) 3902–3909