Electronic Notes in Theoretical Computer Science 47 (2001)
URL: `http://www.elsevier.nl/locate/entcs/volume47.html` pp. 1 – 18

Attacking Fair-Exchange Protocols: Parallel Models vs Trace Models [1]

Luigia Carlucci Aiello [2]

Dip. Infomatica e Sistemistica - Univ. di Roma "La Sapienza", via Salaria 113, 00198 Roma - ITALY

Fabio Massacci [3]

Dip. Ingegneria dell'Informazione - Univ. di Siena via Roma 56, 53100 Siena – ITALY

Abstract

Most approaches to formal protocol verification rely on an operational model based on traces of atomic actions. Modulo CSP, CCS, state-exploration, Higher Order Logic or strand spaces frills, authentication or secrecy are analyzed by looking at the existence or the absence of traces with a suitable property.

We introduced an alternative operational approach based on parallel actions and an explicit representation of time. Our approach consists in specifying protocols within a logic language ($\mathcal{AL}_{SP}$), and associating the existence of an attack to the protocol with the existence of a model for the specifications of both the protocol and the attack.

In this paper we show that, for a large class of protocols such as authentication and key exchange protocols, modeling in $\mathcal{AL}_{SP}$ is equivalent – as far as authentication and secrecy attacks are considered – to modeling in trace based models.

We then consider fair exchange protocols introduced by N. Asokan et al. showing that parallel attacks may lead the trusted third party of the protocol into an inconsistent state. We show that the trace based model does not allow for the representation of this kind of attacks, whereas our approach can represent them.

[1] We thank the anonymous reviewers for many useful suggestions and C. Marchetti for discussing the tricky issue of (fault-tolerant) local and distributed implementations of the Fair-Exchange protocol. This work is partly supported by ASI, CNR, and MURST grants. F. Massacci acknowledges the support of the CNR Fellowship 203-7-27.

[2] Email:`aiello@dis.uniroma1.it`

[3] Email:`massacci@dii.unisi.it`

1 Introduction

The formal verification of security protocols has been the subject of intense research in the last decade. Beliefs logics [8,7,27], model checking and verification in process algebras using CSP [13,22] or CCS [12], theorem proving with induction [19,5] or refinements [6], state exploration methods [18,16] or strand spaces [28,26], have been successfully applied to security protocols.

Notwithstanding the differences, there is a red thread running through CSP, CCS, state-exploration, higher order logic or strand spaces frills: a common operational model based on traces of atomic actions, so that authentication or secrecy can be analyzed by looking at the existence or the absence of traces with a suitable property.

For instance, Schneider [22] defines authentication as a pair of predicates (R and T) so that whenever a trace of the protocol satisfies R (i.e. some receive event happened) then it must also satisfy T (i.e. some transmission event must have happened). Lowe proposed refined concepts of authentication where not only the existence but also the number of traces with a given property is important [14]. In Paulson's approach to authentication [19], properties are exactly theorems over all traces. Using Bolignano's refinement approach [6], we prove that an abstract model of the actual traces is indeed correct. Strand spaces by Fabrega et al. [28] or Meadows' model behind the NPA [16,17] can also be brought under the same roof.

Expressing authentication as a property over traces is not necessarily a compulsory choice for some formalisms. However, reasoning about traces is most comfortable and it is widely believed that nothing is lost.

In [9,10] we introduced an alternative operational approach based on parallel actions and an explicit representation of time. Our approach consists in specifying protocols within a logic programming language ($\mathcal{AL}_{SP}$), and associating each run of the protocol, possibly described by many parallel actions, with a stable model of the protocol's specification. The existence of an attack can then be mapped into the existence of a model for the specifications of both the protocol and the attack.

Obviously, if we impose that only one action can be executed at a time, our model boils down again to the standard trace-based model, where time is just an index over a trace. This makes it possible to state soundness and completeness for bounded model checking over the classical model [9].

However, things are not so simple when we allow for truly parallel actions.

1.1 Contribution of this paper

We show that, for many protocols such as authentication and key exchange protocols, modeling in $\mathcal{AL}_{SP}$ and in trace based models is equivalent, as far as the representation of authentication and secrecy attacks are concerned.

We follow upon the analysis of fair exchange protocols by Asokan et al.

[2,3] started by Shmatikov and Mitchell [24]. We show that there are parallel attacks that may lead the trusted third party in the protocol into an inconsistent state. The trace based model approach does not allow for the representation of this kind of attacks (so that the protocol is "secure by default"), whereas our approach can represent them.

This result shows that verification is sensitive to the model we use to represent protocols, and that we can classify protocols in two major classes: *monotone and nonmonotone protocols*. To execute an action in a monotone protocol an agent must only look at what happened, whereas an agent running a nonmonotone protocol must take care of the past and of what is currently happening in parallel with his intended action. An alternative way to look at the same problem is to say that in a monotone protocol we can always merge two "distinct" traces of the protocol and obtain another trace of the same protocol. With a nonmonotone protocol, we can no longer do so.

This makes also a difference for implementors: monotone protocols can have parallel implementations — just fork a new thread for each TCP/IP request and forget what's happening at your side —, for nonmonotone ones even variable synchronization might not be enough to avoid a disaster.

2 A Trace-based Model

Here we define an operational trace-based model of protocols. Mostly we borrow from Paulson [19,5] and Schneider [22]. However, the features are shared with all cited verification papers.

We assume a sorted *Herbrand domain of messages* H_m, where equality boils down to syntactic equality. The basic building blocks are *agents* A, B, etc. *nonces* N_A, N_B, etc. and *cryptographic keys*. We have at least two types of encryption keys: *symmetric keys* and *asymmetric (public/private) keys*. A key K is decorated by indices that denote the agents that know that key, e.g. K_A is a key known (only) to agent A; K_{AB} is a key shared by agents A and B etc. Public and private keys are sometimes prefixed by the letters "p" or "s". For instance pK_A is A's public key and sK_A is A's secret key.

We denote the concatenation of message m_1 and m_2 as $m_1 \| m_2$. The hashing of a message m is $h(m)$. The encryption of m with key k is denoted by $\{m\}_k$ and the type of encryption is determined by the type of the key.

We assume a distinguished domain of *traces of actions* $H_{\mathcal{T}}$ built by the concatenation of the primitive actions, i.e. elements of the Herbrand domain H_{act} representing the set of all possible $\underline{says}(A, B, M)$, $\underline{gets}(B, M)$, and $\underline{notes}(A, M)$, where A and B are agents and M is a message. Many protocols require reasoning about time; we thus assume a function $\underline{now}(\cdot)$ defined over a trace that identifies what time it is at any point of the trace. We also allow for natural numbers (time) and the primitive relation "less-than", which allows us to compare numbers with the current time. We assume that time is discrete: if $\mathcal{T}$ is a trace and $\#$ the concatenation operator on traces, we may have that

$$cr_1 \quad A \longrightarrow B : \{A \| N\}_{pK_B}$$

$$cr_2 \quad B \longrightarrow A : \{B \| N\}_{pK_A}$$

$$cr_3 \quad A \longrightarrow B : N$$

cr_1: if $\mathcal{T} \in \mathcal{P}_{cr}$ and $N \notin \mathcal{T}$ and $\mathsf{ag}(A) \wedge \mathsf{ag}(B) \wedge \mathsf{nonce}(N)$
then $\underline{says}(A, B, \{A \| N\}_{pK_B}) \# \mathcal{T} \in \mathcal{P}_{cr}$

cr_3: if $\mathcal{T} \in \mathcal{P}_{cr}$ and $\underline{says}(A, B, \{A \| N\}_{pK_B}) \in \mathcal{T}$ and $\underline{gets}(A, \{B \| N\}_{pK_A}) \in \mathcal{T}$
and $\mathsf{ag}(A) \wedge \mathsf{ag}(B) \wedge \mathsf{nonce}(N)$
then $\underline{says}(A, B, \{B \| N\}_{pK_B}) \# \mathcal{T} \in \mathcal{P}_{cr}$

Fig. 1. A simple protocol and some inductive rules

$\underline{now}(\underline{says}(a, b, m) \# \mathcal{T}) = \underline{now}(\mathcal{T}) + 1$, and similarly for the other actions.

Now we are ready to define the protocol model:

Definition 2.1 A *protocol* $\mathcal{P}$ is an inductively defined set of traces, starting from the empty trace, such that all inductive rules are instances of the following schema:
Let act_{next} be an action in H_{act}; let $\mathcal{A} \subseteq H_{act}$ be a subset of primitive actions; let $\mathcal{M}$ be the set of (fresh) messages, i.e. $\mathcal{M} \subseteq H_m$.

if • $\mathcal{T} \in \mathcal{P}$ and
- for all $act \in \mathcal{A}$, act occurs in $\mathcal{T}$ and
- for all $m \in \mathcal{M}$, m does not occur in $\mathcal{T}$ (i.e. it is fresh) and
- a conjunction of monadic predicates (or time predicates) over $\mathcal{T}$, the objects occuring in the actions in $\mathcal{A}$ and in the action act_{next}.

then $act_{next} \# \mathcal{T} \in \mathcal{P}$.

Equality or monadic predicates may specify roles, such as servers or clients, and temporal relations less-than, greater-than may be used for timestamps.

Each inductive rule of a protocol can also be seen as a prefix of a strand, in the sense of Guttman et al. [28,26].

In Figure 1 we show a simple protocol and some inductive rules characterizing it. We use capital letters to indicate (implicitly) universally quantified variables. We also abuse notation and we write $act \in \mathcal{T}$ for "the action act occurs in the trace $\mathcal{T}$" and $m \in \mathcal{T}$ for "the message m occurs in the trace $\mathcal{T}$".

The condition on the set $\mathcal{M}$ does not make possible (unless the protocol is broken) to mismatch the ordering of nonces so that an old reply is taken as an answer for a new challenge. Suppose that we have a trace like the following one (traces grow to the left):

$$\underline{gets}(a, \{b \| n\}_{pK_a}) \# \cdots \# \underline{says}(a, b, \{a \| n'\}_{pK_b}) \# \cdots$$
$$\cdots \# \underline{gets}(a, \{b \| n\}_{pK_a}) \# \cdots \# \underline{says}(a, b, \{a \| n\}_{pK_b}) \# \cdots$$

Clearly we do not want that the reply $\underline{gets}(a, \{b\|n\}_{pK_a})$, which uses n, is accepted as an acknowledgement of the second challenge, which uses n'.

The rule cr_3 in Figure 1 will fire for the first two actions and add the acknowledgement $\underline{says}(a, b, n)$ to the trace. The action $\underline{gets}(a, \{b\|n\}_{pK_a})$ will never be considered a response for the challenge $\underline{says}(a, b, \{a\|n'\}_{pK_b})$ because the nonces n and n' do not match and thus rule cr_3 does not fire.

So far we have no intruder, but we can still define the notion of authentication properties. The definition is borrowed from Schneider [22]. Other, more refined properties can be defined following Lowe [14].

Definition 2.2 An *authentication property* over a set of primitive actions is a pair of finite sets of actions $\mathcal{C}$ (for Causes) and $\mathcal{E}$ (for Effects). A *protocol* $\mathcal{P}$ *satisfies the authentication property* $\langle\mathcal{C}, \mathcal{E}\rangle$ iff whenever all actions in $\mathcal{E}$ occur in a trace $\mathcal{T} \in \mathcal{P}$ at least one action in $\mathcal{C}$ occurs in $\mathcal{T}$.

In other words, $\mathcal{C}$ is the set of all (possibly alternative) causes for all the effects in $\mathcal{E}$. In most cases the sets $\mathcal{C}$ and $\mathcal{E}$ are singletons. It is also possible to change the wording "at least one action" into "all actions" and viceversa, to consider disjunction of effects and conjunction of causes etc. With infinite domains this can be translated into family of pairs or, as done by Schneider, into arbitrary predicates over traces.

Verifying security properties without an intruder has little sense: in the Dolev-Yao model, the intruder is able to read traffic not directed to him, and to send messages without respecting the protocol. Thus we need to define *operators on traces* which take a trace and return a set of messages (the messages grabbed by the intruder) and *operators on messages* which take a set of messages and return the set of messages (the messages inferred by the intruder out of the knowledge of the input set of messages).

For instance, Bella uses $init(A)$ as the set of messages initially known to agent A and defines the inductive operator $\underline{knows}$ as in Figure 2: A similar definition, modulo notational changes, is introduced in the CSP modelling by Schneider and Lowe, where the intruder process is labelled by a set S, the set of known messages, which grows as soon as the intruder receives (i.e. synchronizes, in process algebra's terminology) a message.

For operators over messages, Schneider [22] and Lowe [14] use the inductively defined operator $S \vdash m$, where S is a set of messages and m is a message. Paulson [19] chose to split $\vdash$ into two different operators, one for analyzing and one for composing message (resp. called $\underline{analz}$, and $\underline{synth}$). The "knowledge of the intruder" is then defined by the combination of the predicates $\underline{synth}(\underline{analz}(\underline{knows}(spy, \mathcal{T})))$. This defines the set of messages that the spy can compose after decrypting all messages she has intercepted in the traffic.

Now, we can upgrade the definition of protocols given in Definition 2.1 by allowing two additional inductive rules :

- if $\mathcal{T} \in \mathcal{P}$ and $M \in \underline{pSet}(\underline{pTrace}(spy, \mathcal{T}))$ then $\underline{says}(spy, A, M)\#\mathcal{T} \in \mathcal{P}$.

$$
\begin{aligned}
\underline{knows}(C, []) &= init(C) \\
\underline{knows}(C, \underline{says}(A,B,M)\#\mathcal{T}) &= \begin{cases} \{M\} \cup \underline{knows}(C,\mathcal{T}) & \text{if } C = A \\ \{M\} \cup \underline{knows}(C,\mathcal{T}) & \text{if } C = spy \\ \underline{knows}(C,\mathcal{T}) & \text{otherwise} \end{cases} \\
\underline{knows}(C, \underline{gets}(A,M)\#\mathcal{T}) &= \begin{cases} \{M\} \cup \underline{knows}(C,\mathcal{T}) & \text{if } C = A \\ \underline{knows}(C,\mathcal{T}) & \text{otherwise} \end{cases} \\
\underline{knows}(C, \underline{notes}(A,M)\#\mathcal{T}) &= \begin{cases} \{M\} \cup \underline{knows}(C,\mathcal{T}) & \text{if } C = A \\ \{M\} \cup \underline{knows}(C,\mathcal{T}) & \text{if } bad(A) \\ & \text{and } C = spy \\ \underline{knows}(C,\mathcal{T}) & \text{otherwise} \end{cases}
\end{aligned}
$$

Fig. 2. A sample inductive predicate over traces

- if $\mathcal{T} \in \mathcal{P}$ and $\underline{says}(A, B, M)$ occurs in $\mathcal{T}$ then $\underline{gets}(spy, M)\#\mathcal{T} \in \mathcal{P}$.

Obviously $\underline{pSet}(\cdot)$ and $\underline{pTrace}(\cdot)$ depend on the particular model and abilities of the intruder. Almost all models incorporate syntactic variants of the combination of the predicates $M \in \underline{synth}(\underline{analz}(\underline{knows}(spy, \mathcal{T})))$ by Paulson or $\underline{knows}(spy, \mathcal{T}) \vdash M$ by Lowe.

3 A Time-based Model

We consider the same *Herbrand domains* of agents, nonces, keys, messages etc. as for traces, but model the "state-of-the-world" differently. We just sketch the key ideas here, and refer to Aiello and Massacci [9,10] for further details.

First, we assume a number of predicates over the same domains, called *fluents*[4], which denote properties of the world and whose value changes over time, i.e. the predicates have an extra parameter (a natural number) denoting the time t when they hold. As an example: $\mathsf{got}(alice, m, t_1)$, and $\neg\mathsf{got}(alice, n, t_2)$ indicate that the agent *alice* got a message m at time t_1, whereas she didn't get it at time t_2.

Action predicates are denoted by predicate symbols with an extra time parameter, in one-to-one correspondence with the actions of the trace-based model. To distinguish the two sorts, we use the underlined $\underline{says}(A, B, M)$ to denote the action occurring within a trace, and $says(A, B, M, T)$ to denote the corresponding action predicate. The time argument of the action predicates indicate the time at which the action is performed: $says(bob, alice, m, t_1)$ indicates that the agent *bob* sends message m to agent *alice* at time t_1. It is assumed that actions are instantaneous, i.e. their execution initiates at time

[4] Our nomenclature is borrowed from Reiter [21].

t and terminates at time $t+1$.

Then, we consider a *situation* as a set of predicates having the same time instant as parameter, and denoting the true properties of the world at the corresponding time. By close world assumption, we read as false all predicates not occurring in the set. The action predicates true in a situation mean that the corresponding actions have been all started in parallel at time t. In this way we can model communication concurrency.

A *run* is a finite sequence of situations indexed by the time instants 0, 1, 2, ... A *protocol run* is a run that satisfies some additional constraints concerning the particular protocol specification, and some general properties concerning causes and effects.

"*Causal laws*" specify how the truth value of action predicates affect the values of the fluents, in the form of the so called *successor state axioms*. For example, the effect of an agent getting a message is the logical constraint

$$\mathsf{got}(A, M, T+1) \leftarrow gets(A, M, T).$$

The *preconditions* for sending a response after having received the appropriate challenge can be described with the following constraint:

$$\begin{aligned}\{says(A, B, N, T)\} \leftarrow\ & \mathsf{said}(A, B, \{A \| N\}_{pK_B}, T),\\ & \mathsf{got}(A, \{B \| N\}_{pK_A}, T), \ldots\end{aligned}$$

The "*laws of inertia*" specify which fluents are unaffected by particular actions, so their truth values persist through the action execution. As an example, intercepting a message does not change its content, the status of other messages already sent by the agents participating in the protocol, the encryption keys used for them, and so on. For instance, sent messages remain sent:

$$\mathsf{said}(A, B, M, T+1) \leftarrow \mathsf{said}(A, B, M, T).$$

Freshness of nonces is enforced by additional rules: we introduce the predicate *used* to specify that an object has occurred in a protocol action, hence freshness amounts to not used. Additional care must be taken when we allow for parallel actions. Intuitively, $\mathsf{fresh}(N, T)$ is true in a stable model if N does not occur in any $says(A, B, M, T')$ in the stable model for $T' < T$ and is not used as fresh by two distinct agents (or by the same agent in two distinct messages) in parallel at time T. See [9,10] for details. This treatment is close to the "uniquely originates here" of strand models [28,26].

All laws can be recast as constraints on the possible sets of predicates that are admissible for consecutive times t and $t+1$. These constraints are specified using the logic language $\mathcal{AL}_{SP}$. So all stable models of the specification of the protocol, i.e. of the logical formulae in $\mathcal{AL}_{SP}$ describing the protocols plus the effect and inertial axioms, describe a possible protocol run.

In a nutshell, in a trace based model the "state-of-the-world" is a trace, i.e. a sequence of actions. The inductive definition of a protocol just specifies which worlds are possible. Here, the "world" is a time indexed sequence of situations, where each situation is a set of actions which happen in parallel at the specified time instant. The $\mathcal{AL}_{SP}$ specifications indicate which worlds are possible, i.e. which protocol runs, with possibly parallel actions, correspond to stable models of the specification.

So far we have not imposed a limitation on the finiteness of the Herbrand domain. To keep decidability we impose finitenes: that is we only deal with a finite number of agents and other atomic types (e.g. keys); see [10] for details.

4 Equivalence for Monotone Protocols

So far we have only spoken about protocols. To be, precise we should call protocols defined according to Definition 2.1, *monotone protocols*. In this section we show how we can recast this trace-based description of (monotone) protocols into equivalent logical constraints in $\mathcal{AL}_{SP}$.

To this end, we assume some familiarity with logic programming notation [5], as for instance introduced in [1], and refer to [9] for the definition of the semantics of $\mathcal{AL}_{SP}$ rules.

Definition 4.1 Let $\mathcal{P}$ be a protocol. The $\mathcal{AL}_{SP}$ *specification P is adequate for $\mathcal{P}$* if for every rule of the protocol's inductive definition as defined in Definition 2.1 where $act_{next} = \underline{says}(a,a,b)m$ there is an $\mathcal{AL}_{SP}$ rule of the form

$$\begin{aligned}\{says(a,b,m,T)\} \longleftarrow\ & \textstyle\bigwedge_{\underline{says}(a,b_i,m_i)\in\mathcal{A}} \mathsf{said}(a,b_i,m_i,T),\\ & \textstyle\bigwedge_{\underline{gets}(a,m_j)\in\mathcal{A}} \mathsf{got}(a,m_j,T),\\ & \textstyle\bigwedge_{\underline{notes}(a,m_k)\in\mathcal{A}} \mathsf{noted}(a,m_k,T),\\ & \textstyle\bigwedge_{m_l\in\mathcal{M}} \mathsf{fresh}(m_l,T)\\ & \text{other predicates in the inductive rule}\end{aligned}$$

and similarly for $\underline{gets}(a,m)$ and $\underline{notes}(a,m)$. The laws of inertia and effect axioms for $\mathsf{said}(A,B,M,T)$, $\mathsf{got}(A,M,T)$, $\mathsf{noted}(A,M,T)$ are the only other rules for these fluents in P, and P includes the rules for defining $\mathsf{fresh}(M,T)$.

The definition of $\mathsf{fresh}(M,T)$ can be found in [9,10].

Then we can prove the following lemma:

Lemma 4.2 *Let $\mathcal{P}$ be a protocol over a ground domain $\hat{D}$ and let P be an adequate $\mathcal{AL}_{SP}$ specification of $\mathcal{P}$. Then for every trace $\mathcal{T} \in \mathcal{P}$ there is a stable model of $P \cup \hat{D}$ such that for every action $\underline{says}(a,b,m)$ in $\mathcal{T}$, respectively $\underline{gets}(a,m)$ or $\underline{notes}(a,m)$,*

(i) *there is a time t such that $says(a,b,m,t)$, respectively $gets(a,m,t)$ or $notes(a,m,t)$, is in the stable model of P;*

[5] We recall that $a \longleftarrow \bigwedge_i b_i \wedge \bigwedge_j not\ c_j$ means that a must be in all stable models of the specifications which contain all b_i and none of the c_j [1]. The choice rule $\{a\} \longleftarrow \bigwedge_i b_i \wedge \bigwedge_j not\ c_j$ means that a may be in a stable model satisfying all b_i and no c_j.

(ii) *if* $\underline{says}(a', b', m')\#\mathcal{T} \in \mathcal{P}$*, resp.* $\underline{gets}(a', m')\#\mathcal{T} \in \mathcal{P}$ *or* $\underline{notes}(a', m')\#\mathcal{T} \in \mathcal{P}$*, then there is a time* $t' \geq t$ *such that* $says(a', b', m', t')$*, respectively* $gets(a', m', t')$ *or* $notes(a', m', t')$*, is in the stable model of* P*.*

The proof is by induction on the length of the trace (see [10]).

To prove the opposite direction we must weaken the hypothesis that the granularity of time is 1. Thus, we assume that time is discrete but its granularity can be as small as needed, i.e. $\underline{now}(\underline{says}(a, b, m)\#\mathcal{T}) = \underline{now}(\mathcal{T}) + \epsilon$ where ϵ is a small positive quantity. Otherwise, there will be obvious cases where the parallel execution of the protocol cannot be serialized in a trace. For instance, suppose that we get a "ticket for services" (such as in Kerberos [23]) with a lifetime of 10 seconds. If we are able to make 20 parallel requests of services per each second, we can have 200 requests before the ticket expires. Once we serialize everything – one action per second – we can only ask for 10 requests before the expiration time.

Lemma 4.3 *Let* $\mathcal{P}$ *be a protocol over a finite ground domain* $\hat{D}$ *and let* P *be an adequate* $\mathcal{AL}_{SP}$ *specification of* P*. For every stable model of* $P \cup \hat{D}$ *there is a trace* $\mathcal{T} \in \mathcal{P}$ *such that for every action fluent* $says(a, b, m, t)$*, respectively* $gets(a, m, t)$ *or* $notes(a, m, t)$*, true in the stable model of* P *one has*

(i) $\underline{says}(a, b, m)$*, respectively* $\underline{gets}(a, m)$ *or* $\underline{notes}(a, m)$*, is in* $\mathcal{T}$*;*

(ii) *if* $says(a', b', m', t')$*, respectively* $gets(a', m', t')$ *or* $notes(a', m', t')$*, is in the stable model of* P *for some* $t' > t$ *then the trace* $\mathcal{T}$ *can be decomposed as the concatenation* $\mathcal{T}'\#\underline{says}(a', b', m')\#\mathcal{T}''\#\underline{says}(a, b, m)\#\mathcal{T}'''$*, respectively with* $\underline{gets}(a', m')$ *or* $\underline{notes}(a', m)$ *etc.*

provided the $\underline{now}()$ *function is discrete but not necessarily with a unit step.*

In the proof we need the hypothesis that protocols are defined according to Definition 2.1. Consider the simple case of two actions done in parallel at time t. From the view point of either action, the other action was not done yet. After we serialize them, when the "second" action is added to the trace, the "first" action has been already done. With monotonic preconditions this is not a problem, as we have no precondition on what is *not* in a trace.

Then we can define the notion of adequate specifications in $\mathcal{AL}_{SP}$ of the inductive predicates $\underline{pSet}(\vec{t})$ and $\underline{pTrace}(\vec{t}, \cdot)$ and specify an adequate representation for an authentication property:

Definition 4.4 Let $\langle \mathcal{C}, \mathcal{E} \rangle$ be an authentication property over a domain D and a protocol $\mathcal{P}$, and a $\mathcal{AL}_{SP}$ specification P. The *definition of attack is adequate for* $\mathcal{AL}_{SP}$ if for all time instants t' there is a time $t \geq t'$ in P and a rule of the form

$$\begin{aligned} attack \longleftarrow\ & \textstyle\bigwedge_{\underline{says}(a,b,m)\in\mathcal{E}} \mathsf{said}(a,b,m,t), \bigwedge_{\underline{gets}(a,m)\in\mathcal{E}} \mathsf{got}(a,m,t), \\ & \textstyle\bigwedge_{\underline{notes}(a,m)\in\mathcal{E}} \mathsf{noted}(a,m,t) \\ & \textstyle\bigwedge_{\underline{gets}(a,m)\in\mathcal{C}} not\ \mathsf{got}(a,m,t), \bigwedge_{\underline{says}(a,b,m)\in\mathcal{C}} not\ \mathsf{said}(a,b,m,t), \\ & \textstyle\bigwedge_{\underline{notes}(a,m)\in\mathcal{C}} not\ \mathsf{noted}(a,m,t) \end{aligned}$$

No other rule for *attack* is present.

The condition on time is essential for the final equivalence result to hold: attacks in the protocol models correspond to stable models where *attack* is true. Otherwise, there might be stable models where an attack is indeed present but, because it took longer than expected, the *attack* atom is not present in the stable model. If time is finite, we only need one ground rule (for the $t_{max}+1$ instant), otherwise we need infinitely many ground rules.

Then, one can prove the equivalent of Lemma 4.2 and Lemma 4.3 for the upgraded protocol model with the intruder. These results can be combined yielding the final theorem:

Theorem 4.5 *Let $\mathcal{PI}$ be a protocol augmented with an intruder and $\langle\mathcal{C},\mathcal{E}\rangle$ an authentication property over a finite ground domain $\hat{D}$. Let P be an admissible $\mathcal{AL}_{SP}$ specification such that all its ground instances for the time instants $t = 1, 2, 3\ldots$ are adequate for $\mathcal{P}$. There is a trace of the protocol $\mathcal{P}$ of length at most t_{max} which violates the authentication properties iff there is a stable model of the ground instance P for $t = t_{max}+1$ which contains the atom attack.*

Clearly the actions which lead to the attack are identified by the action predicates ($says(A,B,M,T)$, $gets(B,M,T)$, $notes(A,M,T)$) that are true in the stable model. This is also true when the stable model is infinite, e.g. if we allow for an infinite number of agents and nonces or do not set a bound on the maximum length of possible runs.

Similar definitions and theorems can be given for secrecy properties.

5 (Optimistic) Fair-exchange protocols

Fair exchange protocols have been introduced by Asokan et al. [3], and have also been formally verified for fairness by Shmatikov and Mitchell [24]. We take the description of the protocol from Asokan's PhD thesis [2, Sec.2.2].

The protocol is run by two agents, O and R, who want to sign a contract, occasionally calling a third agent, a trusted third party TTP, in case of disputes. Given the inherent asymmetry of electronic communication, O and R do not want to commit to something and then be blocked forever waiting for the other party to commit, just because the other party is lousy or the communication is unreliable. The word "optimistic" in the definition stems from the assumption that both O and R are usually committed to sign the contract or exchange the goods and thus the TTP will be called into play only when things drag for too long. It is assumed that the communication channel between TTP and any other party is resilient (i.e. all sent messages will be eventually received), that the TTP behaves correctly, or at least always send a syntactically valid reply (not necessarily the right one) to every request.

Once we abstract away from the cryptographic details, the protocol is constituted by three subprotocols [2, Sec.2.2]: one for the normal exchange, one for aborting, and one for resolving the protocol. In the sequel we denote by me_i the i-th message of the exchange protocol, by mr_j the j-th message of the resolve a protocol, and so on.

Normal Exchange. O creates a random number N_O and then sends R a signed message including the hash of N_O, the text of the contract, the name of TTP and both O and R public verification keys that is

$$me_1 = \{pK_O \| pK_R \| TTP \| text \| h(N_O)\}_{sK_O}.$$

If R wants to continue, he generates another random number N_R, concatenates the received message with the hash of N_R, signs the whole lot, and sends it to O: $me_2 = \{me_1 \| h(N_R)\}_{sK_R}$. O responds by sending N_O, and R concludes the protocol by returning N_R.

$$\begin{array}{ll}
me_1 & O \longrightarrow R : \{pK_O \| pK_R \| TTP \| text \| h(N_O)\}_{sK_O} \\
me_2 & R \longrightarrow O : \{me_1 \| h(N_R)\}_{sK_R} \\
me_3 & O \longrightarrow R : N_O \\
me_4 & R \longrightarrow O : N_R
\end{array}$$

O **Aborts.** If the first reply from R is late for whatever reason, O may decide to abort. Then she sends TTP a signed message including her first message to R and the `abort` token: $ma_1 = \{\texttt{abort} \| me_1\}_{sK_O}$. The TTP checks whether the protocol has been already resolved — e.g. it has already received the missing reply me_2 from R — and then issues a replacement contract, i.e. it signs the pair comprising O first signed message and R first signed message: $ma_2 = \{me_1 \| me_2\}_{sK_{TTP}}$. If the contract has not been resolved, it stores the information that me_1 aborted and returns the message $ma_2 = \{\texttt{abort} \| ma_1\}_{sK_{TTP}}$.

$$\begin{array}{llll}
ma_1 & O \longrightarrow TTP & : & \{\texttt{abort} \| me_1\}_{sK_O} \\
ma_2 & TTP \longrightarrow O & : & \text{if } \quad \texttt{resolve} == true \\
 & & & \text{then } \{me_1 \| me_2\}_{sK_{TTP}} \\
 & & & \text{else } \texttt{abort} = true \\
 & & & \qquad \{\texttt{abort} \| ma_1\}_{sK_{TTP}}
\end{array}$$

O **or** R **Resolves.** If either N_O or N_R are late for whatever reason, either R or O may decide to contact TTP to resolve the contract. To this end, one sends TTP both messages me_1 and me_2, i.e. $mr_1 = me_1 \| me_2$. The TTP checks whether the protocol already aborted — e.g. it has already received the `abort` token from O — and then issues an abort message i.e. $mr_2 = \{\texttt{abort} \| ma_1\}_{sK_{TTP}}$. Otherwise, it checks that both parties have duly signed the text and then issues a replacement contract i.e. $mr_2 = \{me_1 \| me_2\}_{sK_{TTP}}$.

$$\begin{array}{lll}
ma_1 & O \longrightarrow TTP & : me_1 \| me_2 \\
ma_2 & TTP \longrightarrow O & : \text{if} \quad \texttt{abort} == true \\
 & & \quad \text{then } \{\texttt{abort} \| me_1\}_{sK_{TTP}} \\
 & & \quad \text{else } \texttt{resolve} = true \\
 & & \qquad \{me_1 \| me_2\}_{sK_{TTP}}
\end{array}$$

Valid Contracts. A valid contract has either the form $me_1 \| N_A \| me_2 \| N_B$, the normal exchange went through, or the form $\{me_1 \| me_2\}_{sK_{TTP}}$, the resolve protocol was called upon. Notice, as pointed out in [24], that abort does not mean that the exchange has been canceled, but just a promise from TTP to respond the same way in the future.

he tricky bit is modelling the condition “if the protocol has not yet been resolved” in our trace based model. A natural way is to use the action *notes* to model the boolean variable: rather than checking whether aborted is true we check whether $\underline{notes}(ttp, aborted)$ is in the trace, and similarly for resolved.

This protocol has been already verified by Shmatikov and Mitchell [24] and, though a number of attacks have been found (e.g. O can obtain a contract which is inconsistent with a contract obtained by R), no substantial weakness of the TTP-side of the protocol has been noted.

We now consider the following basic security property, which is implicitly assumed in the proof of verifiability of TTP by [2,3]:

Claim 5.1 *A honest trusted third party cannot be misled to issue two conflicting statements concerning the final outcome of the protocol.*

Indeed, in [2,3] a stronger claim is made: the protocol may be unfair to O, that is O gets an abort token from TTP whereas R gets a replacement contract, only if TTP misbehaves.

The natural trace based description of the abort and resolve steps of the TTP actions is shown in Figure 3 where we denote $\mathcal{P}_{fe}$ the fair exchange protocol. The preconditions of the inductive definition are shown above the rule and the consequence is added below the rule.

It is immediate to see that with this model no attack is possible. Since there is a total order on actions, and only one action at a time can be concatenated to a trace, as soon as $\underline{notes}(TTP, \texttt{abort})$ is inserted into the trace, it is no longer possible to insert $\underline{notes}(TTP, \texttt{resolve})$ into an extension of that trace. So, in

AbortResolved.
if $\mathcal{T} \in \mathcal{P}_{fe}$ and $\underline{gets}(TTP, \{\mathtt{abort}\|me_1\}_{sK_O}) \in \mathcal{T}$ and
$\underline{notes}(TTP, \mathtt{resolve}\|me_1\|me_2) \in \mathcal{T}$
then $\underline{says}(TTP, O, \{me_1\|me_2\}_{sK_{TTP}})\#\mathcal{T} \in \mathcal{P}_{fe}$

AbortNotResolved.
if $\mathcal{T} \in \mathcal{P}_{fe}$ and $\underline{gets}(TTP, \{\mathtt{abort}\|me_1\}_{sK_O}) \in \mathcal{T}$ and
$\underline{notes}(TTP, \mathtt{resolve}\|me_1\|me_2) \notin \mathcal{T}$
then $\underline{says}(TTP, O, \{\mathtt{abort}\|ma_1\}_{sK_{TTP}})\#\underline{notes}(TTP, \mathtt{abort}\|ma_1)\#\mathcal{T}\in\mathcal{P}_{fe}$

ResolveAborted.
if $\mathcal{T} \in \mathcal{P}_{fe}$ and $\underline{gets}(TTP, me_1\|me_2) \in \mathcal{T}$ and $\underline{notes}(TTP, \mathtt{abort}\|ma_1) \in \mathcal{T}$
then $\underline{says}(TTP, O, \{\mathtt{abort}\|ma_1\}_{sK_{TTP}})\#\mathcal{T} \in \mathcal{P}_{fe}$

ResolveNotAborted.
if $\mathcal{T} \in \mathcal{P}_{fe}$ and $\underline{gets}(TTP, me_1\|me_2) \in \mathcal{T}$ and $\underline{notes}(TTP, \mathtt{abort}\|ma_1) \notin \mathcal{T}$
then $\underline{says}(TTP, O, \{me_1\|me_2\}_{sK_{TTP}})\#\underline{notes}(TTP, \mathtt{resolve}\|me_1\|me_2)\#\mathcal{T}\in\mathcal{P}_{fe}$

Fig. 3. Inductive Rules for the TTP of the Abort-Resolve subprotocols

% AbortResolved

$$\{says(TTP, A, \{me_1\|me_2\}_{sK_{TTP}}, T)\} \longleftarrow \mathsf{got}(TTP, \{\mathtt{abort}\|me_1\}_{sK_A}, T), \mathsf{noted}(TTP, \mathtt{resolve}\|me_1\|me_2, T)$$

% AbortNotResolved:

$$\{notes(TTP, \mathtt{abort}\|ma_1, T)\} \longleftarrow \mathsf{got}(TTP, \{\mathtt{abort}\|me_1\}_{sK_A}, T), \; not\ \mathsf{noted}(TTP, \mathtt{resolve}\|me_1\|me_1, T)$$

$$says(TTP, A, \{\mathtt{abort}\|ma_1\}_{sK_{TTP}}, T+1) \longleftarrow notes(TTP, \mathtt{abort}\|ma_1, T)$$

Fig. 4. $\mathcal{AL}_{SP}$ "Natural" Definition of Abort Subprotocol

every trace either $\underline{notes}(TTP, \mathtt{abort})$may occur or $\underline{notes}(TTP, \mathtt{resolve})$ may occur, but not both.

The "natural" $\mathcal{AL}_{SP}$ translation of the above Abort steps (the resolve part is dual), obtained by extending Definition 4.1 is shown in Figure 4. Notice that we used a choice rule only for the "Notes" subaction and the normal rule for the "Says" subaction. This is due to the fact that the second step is compulsory: the TTP, following the protocol, must store the outcome of the abort action and then communicate the result to O.

Now we can look for models satisfying the goal:

$$\begin{aligned} \mathit{attack} \longleftarrow\ & \mathsf{said}(TTP, O, \{\mathtt{abort} \| ma_1\}_{sK_{TTP}}, T), \\ & \mathsf{said}(TTP, R, \{me_1 \| me_2\}_{sK_{TTP}}, T) \end{aligned}$$

we find that a model exists, and this model can be transformed into an attack to the actual protocol from [2, Fig. 2.2], even in presence of synchronization primitives on the individual variables `abort`and `resolve`.

To carry on the attack, we must deliver to the TTP a resolve and an abort message in parallel. This may be obtained in a variety of ways: the intruder can delay the abort message by O until R issues a resolve, R herself can send an abort and a resolve ticket at the same time, etc. At this point, since the standard "adequate" preconditions refer to the past, both the rule for abort and the rule for resolve fire, and we are done.

From a practical point of view, the fact that the values of the precondition are examined in parallel is reasonable: it corresponds to synchronizing on individual variables, which is coherent with current web and internet servers technology. Obviously it is possible to force the abort and resolve subprotocols to be transactions, by using a syncronizing primitive at the beginning and end of the entire subprotocol. However, this would make the entire protocol unmanageable for a large number of users.

It is possible to change the rule, but now we have no clear cut way to do it: should resolution be given priority over abortion? Should one of them be chosen non-deterministically? Should the TTP deadlock? Each choice leads to a different $\mathcal{AL}_{SP}$ specification, which can be motivated on different grounds.

6 Monotone and nonmonotone protocols

At this stage, one may wonder whether this problem — the difference between trace and parallel models — is just present in fair-exchange protocols, as no other protocol seems to have suffered from this problem before.

We argue that the problem is more general, and indeed that it is in the definition of protocol from Definition 2.1. A similar definition can be derived from the strand-space model by Guttman et al. [28], from Paulson's models [19] and from the current CSP [22,14] and CCS [12] modelling of protocols.

The key issue is that one usually imposes constraints on *what is* in a trace, but has no constraint on *what is not* in a trace. This didn't seem to be a problem as all protocols in Clark and Jacob library [11], i.e. practically all authentication and key exchange protocols, can be framed within this definition, and industrial protocols such as SET [15] or TLS follow the rule [4,20].

Fair-exchange protocols, and all auction protocols, contravene this rule: a protocol is resolved only if there is not already an abort message in the trace, a bid is winning if there is not already another winner in the trace, etc.

This makes it possible to define two major classes of protocols: *monotone*

$$\begin{array}{lll}
\textit{bid} & A \longrightarrow M : & \{A \| N_A\}_{sK_A} \\
\textit{assign} & M \longrightarrow A : & \text{if} \quad A \text{ arrived first with } N'_A \\
 & & \qquad \{A \| N'_A \| N_A\}_{sK_M} \\
 & & \text{else} \quad \text{some } B \text{ arrived first with } N_B \\
 & & \qquad \{B \| N_B \| A \| N_A\}_{sK_M}
\end{array}$$

Fig. 5. A Simple Nonmonotone Protocol: a first-pass-the-post auction

protocols and *nonmonotone protocols*. To execute an action in a monotone protocol an agent must only look at what did happen, whereas an agent running a nonmonotone protocol must consider what did happen, what did not happen, and what may be happening in parallel with his intended action.

We chose the wording monotone and nonmonotone, because they can also be distinguished by the following procedure:

(i) take a valid trace of a protocol,

(ii) replace all nonces or timestamps with fresh ones and swap freely agents that play the same roles;

(iii) interleave the original and the modified trace (respecting timestamp ordering if needed);

(iv) is the resulting trace still a valid trace of the protocol?

In a monotone protocol the answer is yes: the protocol is monotone because adding (interleaving) a valid trace with another (different) valid trace is still a valid trace. With a nonmonotone protocol, we can no longer do so.

For instance, in role-based protocol models, like those used by Lowe [14], Song [25], and Stoller [26], a bounded degree of (unsynchronized) parallellism can be modeled by a protocol transformation, which statically takes, for instance, two copies of a role of the original protocol, and interleaves the copies in various ways. However, this only works for monotone protocols.

In Figure 5 we show a simple auction protocol where interleaving cannot be done. A and B bid for a good and M simply certifies who arrived first. Any of the participants might want to refresh the "winning" certificates, so the protocol does not necessarily terminate with the first announcement.

It is easy to see that if we take the trace where *alice* sends the first bid, followed by *bob*, and then by another bid from *alice*, and just rename nonces and change *alice* with *bob*, there is no way in which the two traces can be interleaved. In any trace there can only be *one* message of the form $\{A \| N'_A\}_{sK_M}$. Thus, in a trace-based model it is not possible to attack this protocol. However, as soon as we allow for parallel messages to arrive, the way in which we implement the check "A arrived first" makes a difference as M may be led into an inconsistent state.

This classification is not just important for formal verification purposes but also for practical implementations. Monotone protocols can have parallel implementations — just fork a new thread for each TCP/IP request and forget all the rest —, for nonmonotone ones things are not so simple.

For instance, for the optimistic fair exchange protocol, if we assume a local implementation (e.g. one single Trusted Third Party for the whole Internet) even a local synchronization on individual variables (resolve/abort) would not be sufficient. We would need to have a synchronization step at the beginning and ending of each protocol step. If we go for more realistic distributed implementations, more sophisticated low level protocols and synchronization primitives such as write-once registers are needed.

It is an intriguing goal to determine which formalisms can cope with nonmonotone protocols in their full generality. For instance, we conjecture that strand-spaces, where protocols are defined by roles, e.g. as done by Song [25] or Stoller [26], can only cope with monotone protocols. Moreover, it remains to be seen which are adequate $\mathcal{AL}_{SP}$ rules for dealing with nonmonotone protocols. We leave these issues open for future investigations.

References

[1] K. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. Elsevier Science Publishers (North-Holland), Amsterdam, 1990.

[2] N. Asokan. *Fairness in Electronic Commerce*. PhD thesis, University of Waterloo, Canada, 1998.

[3] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In *Proceedings of the Fourth ACM Conference on Communications and Computer Security (CCS'97)*, pages 8–17. ACM Press and Addison Wesley, 1997.

[4] G. Bella, F. Massacci, L. Paulson, and P. Tramontano. Formal verification of Card-Holder Registration in SET. In F. Cuppens, Y. Deswarte, and D. Gollman, editors, *Proceedings of the Sixth European Symposium on Research in Computer Security (ESORICS 2000)*, volume 1895 of *Lecture Notes in Computer Science*, pages 159–174. Springer-Verlag, 2000.

[5] G. Bella and L. C. Paulson. Kerberos version IV: Inductive analysis of the secrecy goals. In *Proceedings of the Fifth European Symposium on Research in Computer Security (ESORICS'98)*, volume 1485 of *Lecture Notes in Computer Science*, pages 361–375. Springer-Verlag, 1998.

[6] D. Bolignano. An approach to the formal verification of cryptographic protocols. In *Proceedings of the Third ACM Conference on Communications and Computer Security (CCS'96)*, pages 106–118, 1996.

[7] S. Brackin. Automatic formal analyses of two large commercial protocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, September 1997.

[8] M. Burrows, M. Abadi, and R. Needham. A logic for authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[9] L. Carlucci Aiello and F. Massacci. An executable specification language for planning attacks to security protocols. In P. Syverson, editor, *IEEE Computer Security Foundation Workshop*, pages 88–103. IEEE Computer Society Press, 2000.

[10] L. Carlucci Aiello and F. Massacci. Verifying security protocols as planning in logic programming. *ACM Transactions on Computational Logic*, 2001. Accepted for publication November 2000. Available on the web.

[11] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. Technical report, University of York, Department of Computer Science, November 1997. Available on the web at `http://www-users.cs.york.ac.uk/∼jac/`.

[12] R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9):550–571, 1997.

[13] G. Lowe. Some new attacks upon security protocols. In *Proceedings of the Ninth IEEE Computer Security Foundations Workshop (CSFW'96)*, pages 162–169. IEEE Computer Society Press, 1996.

[14] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the Tenth IEEE Computer Security Foundations Workshop (CSFW'97)*, pages 31–43. IEEE Computer Society Press, 1997.

[15] Mastercard & VISA. *SET Secure Electronic Transaction Specification: Business Description*, May 1997. Available electronically at `http://www.setco.org/set_specifications.html`.

[16] C. Meadows. The NRL Protocol Analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1994.

[17] C. A. Meadows. Analyzing the Needham-Schroeder public key protocol: A comparison of two approaches. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *Proceedings of the Fourth European Symposium on Research in Computer Security (ESORICS'96)*, volume 1146 of *Lecture Notes in Computer Science*, pages 351–364. Springer-Verlag, 1996.

[18] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murphi. In *Proceedings of the Sixteenth IEEE Symposium on Security and Privacy (SSP'97)*, pages 141–151. IEEE Computer Society Press, 1997.

[19] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[20] L. C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Transactions on Information and System Security*, 2(3):332–351, 1999.

[21] R. Reiter. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems.* Springer-Verlag, 2001. Draft monograph available at `http://www.cs.toronto.edu/~cogrobo`.

[22] S. Schneider. Verifying authentication protocols in CSP. *IEEE Transactions on Software Engineering*, 24(9):741–758, 1998.

[23] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, 1994.

[24] V. Shmatikov and J. Mitchell. Analysis of a fair exchange protocol. In *Proc. of NDSS 2000*, pages 119–128, San Diego, 2000.

[25] D. Song. Athena: An automatic checker for security protocol analysis. In *Proceedings of the Twelfth IEEE Computer Security Foundations Workshop (CSFW'99)*. IEEE Computer Society Press, 1999.

[26] S. Stoller. A bound on attacks on payment protocols. In *Proceedings of the Sixteenth IEEE Symposium on Logic in Computer Science (LICS 2001)*. IEEE Computer Society Press, 2001.

[27] P. F. Syverson and P. C. van Oorschot. On unifying some cryptographic protocols logics. In *Proceedings of the Thirteenth IEEE Symposium on Security and Privacy (SSP'94)*. IEEE Computer Society Press, 1994.

[28] F. Thayer Fabrega, J. Herzog, and J. Guttman. Honest ideals on strand spaces. In *Proceedings of the Eleventh IEEE Computer Security Foundations Workshop (CSFW'98)*. IEEE Computer Society Press, 1998.