Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 118 (2005) 129–143
www.elsevier.com/locate/entcs

A Trace Logic for Local Security Properties

Ricardo Corin, Sandro Etalle, Pieter Hartel[1,2]

Faculty of Computer Science,
University of Twente, P.O.Box 217, 7500AE Enschede, The Netherlands

Antonio Durante [1,3]

Università di Roma "La Sapienza", Via Salaria 113, Rome 00198, Italy

Abstract

We propose a new simple *trace* logic that can be used to specify *local security properties*, i.e. security properties that refer to a single participant of the protocol specification. Our technique allows a protocol designer to provide a formal specification of the desired security properties, and integrate it naturally into the design process of cryptographic protocols. Furthermore, the logic can be used for formal verification. We illustrate the utility of our technique by exposing new attacks on the well studied TMN protocol.

Keywords: Trace logic, local security property, specification, TMN protocol

1 Introduction

Cryptographic protocols are typically designed to meet security goals such as authentication and confidential key exchange. These goals, usually called *security properties*, can be correctly accomplished if some of the values exchanged during the protocol run satisfy, for instance, classical properties like authenticity, confidentiality, or freshness.

[1] We would like to thank Cabernet and the EYES Project (IST- 2001-34734) for their support of this work.

[2] Email: {corin,etalle,pieter}@cs.utwente.nl

[3] Email: durante@dsi.uniroma1.it

doi:10.1016/j.entcs.2004.12.019

Often, the specification of security properties is carried out by writing of "global" security properties. These security specifications do not depend from any principal's point of view. Thus, to refer to a specific principal, global security properties are usually defined using extra protocol events [8,11].

In this paper, on the other hand, we propose a logic that can be used to express *local security properties*, i.e. properties that refer to the specification of *one* agent, namely the agent which they belong to. As we show in the following sections, local security properties are expressive enough to assert the properties that are commonly desired for cryptographic protocols (e.g., freshness of a nonce.)

The advantage of local properties is that they allow a designer to specify the security properties that should hold, according to each participant, at each protocol execution point. For instance, a property like freshness of a nonce can be specified as a formula that is connected directly to the corresponding participant who receives that nonce. Furthermore, since these formulae correspond to each principal, they depend *only* on information of that principal, as opposed to a global formula that can depend on the whole network state. Thus, a local formula can be bound to each principal and then be "plugged in" into any other network specification. This enables potential composability of the specifications.

Consequently, using local properties, it is possible to integrate the specification of the (logical) security properties that a protocol has to meet *within* the (algorithmic) specification of the protocol itself. This yields an integrated technique for protocol engineering that combines tightly the design and the analysis phase, resulting in a shorter design-verification feedback loop.

We illustrate our approach by studying the TMN protocol [10] for which we have found two new attacks.

Plan of the paper. In Section 2, we describe our security protocol model. Then, in Section 3 we introduce our trace logic language. In Section 4, the TMN protocol is studied and some novel attacks upon it are presented. In Section 5 elaborates the related work and finally conclusions and future work are discussed in Section 6.

2 Protocol Model

A protocol step is usually specified using the standard notation $A \to B : M$. Here, M is a message built from:

- atomic terms, that is constants (written in lowercase) and variables (which are capitalized). Constants may be nonces (e.g. na) or agent identities (e.g. a). A special constant ε denotes the intruder.

- constructed terms, that is a finite application of operators *encryption* M_K, *pairing* M_1, M_2, *hashing* $h(M)$ and finally *public key* $pk(M)$ over atomic terms.

However, the $A \rightarrow B : M$ notation is unsuitable for formal verification. In fact, in a protocol step, two different events take place: A sends message M, and B receives message M'. In presence of an intruder, M might not be equal to M'. Moreover, not even the identities of the correspondent communication parties may be the same (i.e., A sends to B' and B receives from A'.) It is therefore convenient to take an approach that considers separately each agent's point of view; this is the idea of *protocol roles*.

Definition 2.1 A *protocol role* is a pair $\langle A, [M_1 \diamond B_1, ..., M_n \diamond B_n]\rangle$, where $A, B_1, ..., B_n$ are variables, $\diamond \in \{\triangleleft, \triangleright\}$ and $M_1, ..., M_n$ are messages. □

Given a protocol role $\langle A, [M_1 \diamond B_1, ..., M_n \diamond B_n]\rangle$, A is called the *identity* of the role, while elements $M_i \diamond B_i, i = 1..n$ are the *actions* of the role: $M \triangleright B$ is a *send* action, while $M \triangleleft B$ is a *receive* action.

Protocol roles in a security protocol often receive (self explanatory) names such as *initiator, responder* and *server*. For example,

$$\mathit{responder(A,B,Na)} = \langle B, [pk(Na) \triangleleft A]\rangle \tag{1}$$

defines a responder role in which there is only one action, the receipt of Na from A.

Notice that in (1), the variables A, B, Na are still uninstantiated (we borrow this concept from logic programming: as long as no value is assigned to a variable, we call it *uninstantiated*, and *instantiated* otherwise.) In fact, a protocol role is *parametric*, thus representing a template. By appropriately (partially) instantiating a finite number of protocol roles, a *system scenario* can be obtained:

Definition 2.2 A system scenario is a multiset of (partially) intantiated protocol roles.

Typically, a system scenario determines how many sessions are present and which agents play which roles. For instance, the system scenario

$$\{\mathit{responder(A,b,Na),\ responder(C,d,Nc)}\}$$

(where *responder* is the role defined above) defines a system scenario with two responders (notice that there are no corresponding *initiators*), one played by b and the other by d. Uninstantiated variables represent unknown values:

for example, variable A in the first responder role represents the (unknown) communicating party of b.

2.1 Trace Semantics

Executions of system scenarios are described using *traces*, which are in turn composed of *events*, i.e. single actions performed by an agent.

Definition 2.3 An *event* is a pair $\langle A : M \diamond B\rangle$ where A, B are agent's names, $\diamond \in \{\triangleleft, \triangleright\}$ and M is a message. □

The event $\langle A : M \triangleright B\rangle$ should be read as "agent A sends message M with *intended destination* B". On the other hand, $\langle B : M \triangleleft A\rangle$ stands for "agent B receives message M *apparently from* A".

To analyze the protocol, we combine the system scenario with the usual Dolev-Yao intruder [5], who can perform the usual actions: intercept and learn any sent message, store the information contained in intercepted messages for later use, and introduce into the system new messages forged using information the intruder knows. The information obtained by the intruder is stored in a set of terms K called the *intruder's knowledge* [4]

Now we are ready to describe the execution of a system scenario, represented by the notion of a *run*.

Definition 2.4 Let S be a system scenario, and K be the intruder's initial knowledge, consisting of constants representing agents identities and their public keys. Let tr be an initial empty trace. A *run* of S is a trace obtained by a reiterated sequence of the following steps:

(i) a non-empty role in S is chosen nondeterministically, and its first action p is removed from it. Let a be the identity of the chosen role.

(ii) if $p = t \triangleright y$, then:
  (a) t is added to the knowledge of the intruder, $K := K \cup \{t\}$
  (b) event $e = \langle a : p\rangle$ is added to tr, $tr := tr \cdot e$

(iii) if $p = t \triangleleft y$, then:
  (a) it is checked if the intruder ε can generate t using the knowledge K [5], if so, then event $e = \langle a : p\rangle$ is added to the trace: $tr := tr \cdot e$.
  (b) If ε cannot generate such a message, then the run stops.

[4] Because of the symbolic nature of the analyzer, in practice an event can contain variables, which stand for something the intruder can generate (see [3] for details.)

[5] We adopt Millen and Shmatikov's constraint solving procedure [13] for checking if the intruder can generate a term t using knowledge K. This procedure may involve instantiation of variables in t or K; for example, t may unify with a term in K, representing that t is already in K, i.e., is already known by the intruder (see [13] for details.)

□

3 A Trace Logic

In this section we introduce a trace logic language for defining local security properties.

Definition 3.1 A trace logic formula is generated according to the following grammar:

$$\begin{aligned} F ::= \;& \texttt{true} \mid \texttt{false} \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid F_1 \rightarrow F_2 \mid \forall e \in tr\ : F \\ & \mid \exists e \in tr\ : F \mid \exists t : F \mid \neg F \mid e_1 = e_2 \mid e_1 \preceq e_2 \end{aligned}$$

where e, e_1 and e_2 are events. □

The conjunction of two formulae has the usual significance: $F_1 \wedge F_2$ is *true* if both F_1 and F_2 are *true*; the disjunction operator $\vee$ and implication $\rightarrow$ are analogous. On the other hand, the meaning of constructors $\forall e \in tr\ : F$ and $\exists e \in tr\ : F$ is non-standard. Since a trace formula is going to be evaluated on a certain input trace, constructors $\forall$ and $\exists$ allow us to reason about the events in the input trace: $\forall e \in tr\ : F$ asserts that every event e in the input trace satisfies formula F, while $\exists e \in tr\ : F$ express that some event in the input trace satisfies formula F. Notice that tr is not a variable, it is just part of the operators name to emphasize that e ranges over the system trace. Even though this gives a "temporal" flavor to our logic, we anticipate that these constructors only operate on *past* events, recorded in the input trace (see later). Formula $\neg F$ has the usual meaning of negation. Differently from the above operators, $\exists t : F$ quantifies t over all messages and agents space. Finally, predicates $e_1 = e_2$ and $e_1 \preceq e_2$ allow us to compare events: the former asserts equality, and the latter *subterm* inclusion.

While the choice of these constructors may seem rather *ad hoc* for our purposes, we believe this logic can in fact be quite expressive, and allow us to assert a fairly large set of interesting security properties, as will be shown later.

Next, we define the precise meaning of a trace logic formula.

Definition 3.2 Let $\mathcal{F}$ be the set of well-formed trace logic formulae, and TR be the set of traces, then the semantic function $[\![\cdot]\!]\cdot : \mathcal{F} \times TR \rightarrow \{true, false\}$ is defined as follows:

$$
\begin{array}{ll}
[\![\texttt{true}]\!]\ tr & = true \\
[\![\texttt{false}]\!]\ tr & = false \\
[\![F_1 \wedge F_2]\!]\ tr & = true\ \mathit{iff}\ [\![F_1]\!]\ tr = [\![F_2]\!]\ tr = true \\
[\![F_1 \vee F_2]\!]\ tr & = true\ \mathit{iff}\ [\![F_1]\!]\ tr = true\ or\ [\![F_2]\!]\ tr = true \\
[\![F_1 \rightarrow F_2]\!]\ tr & = true\ \mathit{iff}\ [\![F_1]\!]\ tr\ implies\ [\![F_2]\!]\ tr \\
[\![\forall e \in tr\ : F]\!]\ tr & =\text{true}\ \ \mathit{iff,\ for\ each\ event}\ x\ \text{of}\ tr,\ [\![F[^x/e]]\!]\ tr = true \\
[\![\exists e \in tr\ : F]\!]\ tr & =\text{true}\ \ \mathit{iff,\ for\ some\ event}\ x\ \text{of}\ tr,\ [\![F[^x/e]]\!]\ tr = true \\
[\![\exists t\ : F]\!]\ tr & =\text{true}\ \ \mathit{iff,\ for\ some\ message\ or\ agent}\ x,\ [\![F[^x/t]]\!]\ tr = true \\
[\![\neg F]\!]\ tr & = true\ \mathit{iff}\ [\![F]\!]\ tr = false \\
[\![e_1 = e_2]\!]\ tr & = true\ \mathit{iff\ event}\ e_1\ is\ equal\ to\ event\ e_2 \\
[\![t_1 \preceq t_2]\!]\ tr & = true\ \mathit{iff,\ if}\ t_1\ is\ a\ subterm\ of\ t_2
\end{array}
$$

□

Here, $F[x/y]$ is the result of substituting each occurrence of y with x in F.

For the sake of notation's simplicity, we assume that all variables that are not explicitly quantified are *existentially* quantified (over the set of messages and agents). This simplifies the notation considerably.

In the future, we plan to endow our logic with a proof system that allow us to relate proofs of formulae with the intended meaning given by $[\![\cdot]\!]\cdot$. In the present work, we are more interested in exploring the expressive power of security specifications; We plan to continue this work by addressing the issue of using our logic for automatic formal verification.

3.1 Appending local security properties to protocol roles

Now, we are ready to combine the definition of protocol roles and local security properties to obtain *extended protocol roles* and *extended system scenarios*. Intuitively, the idea is to embed the logical security properties within the protocol specification.

Definition 3.3 An *extended protocol role* is a triple $\langle A, [M_1 \diamond B_1 : F_1, ..., M_n \diamond B_n : F_n]\rangle$, where $\{A, B_1, ..., B_n\} \subset Var$, $M_1, ..., M_n$ are messages, $\diamond \in \{\triangleleft, \triangleright\}$ and $F_1, ..., F_n$ are trace logic formulae. □

Intuitively, adding a formula F_i after a protocol role action means that F_i must hold after the execution of the action. Notice that instantiation of an extended protocol role also affects the variables of an attached local security property. This formalizes the notion of a security property being 'local', that is a security specification that takes into account the principal's point of view. Also, F_i is going to be evaluated w.r.t. the system trace, which contains the events *up to* at that precise execution time. This, as we already mentioned, illustrates the "past flavour" nature of our formulae.

Similarly, we can define an extended system scenario as a multiset of (partially instantiated) extended protocol roles.

3.2 Verifying the local security properties

To evaluate the local security properties, we extend the Definition 3.4 to the extended system scenarios introduced in last section:

Definition 3.4 Let S be an *extended* system scenario, and K be the intruder's initial knowledge, consisting of constants representing agents identities and their public keys. Let tr be an initial empty trace. A *run* of S is a trace obtained by a reiterated sequence of the following steps:

(i) a non-empty role in S is chosen nondeterministically, and its first action p is removed from it. Let a be the identity of the chosen role.

(ii) if $p = t \triangleright y : F$, then:
  - (a) if $[\![F]\!]tr$ holds, then continue. Otherwise, the run stops.
  - (b) t is added to the knowledge of the intruder ε, $K := K \cup \{t\}$
  - (c) event $e = \langle a : p\rangle$ is added to tr, $tr := tr \cdot e$

(iii) if $p = t \triangleleft y : F$, then:
  - (a) it is checked if the intruder ε can generate t using the knowledge K (see below), if so, then event $e = \langle a : p\rangle$ is added to the trace: $tr := tr \cdot e$.
  - (b) if $[\![F]\!]tr$ holds, then continue. Otherwise, the run stops.
  - (c) If ε cannot generate such a message, then the run stops.

□

For example, consider the role:

$$\mathit{responder}(B,\ A,\ Na) = \langle B, [Na \triangleleft A : F]\rangle$$

where $F = \exists e : e = \langle A : Na \triangleright B\rangle$.

After the responder B receives the nonce Na, F checks that A had sent Na to B before. Now, consider the singleton scenario $\{\mathit{responder}(b,A,Na)\}$. In this scenario, there is only one honest responder role, played by b. Now, suppose this responder role receives, from the intruder ε, a nonce ni as Na. Therefore, according to Definition 3.4, we have trace $tr = \langle \varepsilon : ni \triangleright b\rangle$. The next step involves evaluation of $[\![F]\!]tr$ to see if the local security property F holds: clearly, we can see that $[\![\exists e : e = \langle A : Na \triangleright b\rangle]\!]\langle \varepsilon : ni \triangleright b\rangle$ evaluates to *true*, unifying A with ε and Na with ni.

3.2.1 Implementation.

We have a (beta version) implementation of $[\![\cdot]\!]\cdot$, encoded into our verifier of [3]. Using it, we were able to perform the verification of the TMN protocol, illustrated in the following section.

4 A Case Study: the TMN protocol

We apply our technique to a well known case study, the TMN protocol [10]. This protocol has been thoroughly studied, see for example [16,14,9]. However, in this section we present some vulnerabilities that we believe no one has noticed before.

4.1 Original Version

The original version of TMN was proposed for achieving key distribution between two users:

$$\begin{array}{lll} \text{Message 1.} & A \rightarrow S & : \; A, S, B, \{R_1\}_{pk(S)} \\ \text{Message 2.} & S \rightarrow B & : \; S, B, A \\ \text{Message 3.} & B \rightarrow S & : \; B, S, A, \{R_2\}_{pk(S)} \\ \text{Message 4.} & S \rightarrow A & : \; S, A, B, v(R_1, R_2) \end{array}$$

We denote Vernam encryption by $v(t_1, t_2)$ [6]. Here, keys R_1 and R_2 are sent from A and B to S, respectively. After Message 4 is received, A can obtain R_2, thus making R_2 the shared key between A and B.

4.1.1 TMN protocol roles.

The first step in our design and verification technique is to obtain the protocol roles from the standard notation:

- Initiator: $\langle A, [A, S, B, \{R_1\}_{pk(S)} \triangleright S : F_1, \quad S, A, B, v(R_1, R_2) \triangleleft S : F_2] \rangle$
- Responder: $\langle B, [S, B, A \triangleleft \; S : F_3, \quad B, S, A, \{R_2\}_{pk(S)} \triangleright S : F_4] \rangle$
- Server: $\langle S, [A, S, B, \{R_1\}_{pk(S)} \triangleleft A : F_5, \quad S, B, A \triangleright B : F_6, \quad B, S, A, \{R_2\}_{pk(S)} \triangleleft B : F_7, \quad S, A, B, v(R_1, R_2) \triangleright A : F_8] \rangle$

This translation can be tedious and error-prone when protocols get large; however, we believe this step can be mostly automated (eg. by a tool assisting the user.)

[6] We currently model Vernam encryption as normal symmetric encryption, and not as full exclusive `xor`.

The original version of TMN suffers from several secrecy attacks over R_2 above, as exposed for instance in [9]. Thus, we will concentrate on two modified versions of the protocol.

4.2 First modification

A replay attack against TMN was exposed by Simmons [18]. The attack exploits the fact that the messages to the server from A and B (Message 1 and Message 3) can be replayed. To solve this defficiency, Tatebayashi and Matsuzaki introduce timestamps in messages 1 and 3 [10]:

Message 1. $A \rightarrow S \ : \ A, S, B, \{T_A, R_1\}_{pk(S)}$

Message 2. $S \rightarrow B \ : \ S, B, A$

Message 3. $B \rightarrow S \ : \ B, S, A, \{T_B, R_2\}_{pk(S)}$

Message 4. $S \rightarrow A \ : \ S, A, B, v(R_1, R_2)$

In this new protocol, after receiving T_A and T_B, the server can check for the timeliness of these timestamps. According to Tatebayashi and Matsuzaki, this new protocol version guarantess the freshness of R_1 and R_2. To check if this is true, we can specify the freshness requirements of R_1 and R_2 as a local security properties of server S:

$$Fresh_{R_i} = \forall e \in tr : last\_event(e) \ \vee \ \neg(R_i \preceq msg(e)) \text{ (for } i = 1, 2)$$

Where primitive $msg(\cdot)$ projects the message of an event, defined as $msg(\langle x : m \diamond y \rangle) = m$ and predicate $last\_event(e)$ is a primitive that is true iff e is the last event of trace tr. The definition of this primitive is straightforward: $[\![last\_event(e)]\!] \ tr = true$ iff $tr = tr' \cdot e$. $Fresh_{R_1}$ and $Fresh_{R_2}$ are expressing that R_1 and R_2, respectively, are fresh.

The last step involves deciding where to put $Fresh_{R_1}$ and $Fresh_{R_2}$ in the server role. This is easy: we make the decision that the formulae for checking the freshness of the received values should be placed *as soon as the values are received*. Thus, $Fresh_{R_1}$ can be put as F_5, that is, after R_1 is received. Similarly, we set $Fresh_{R_2}$ as F_7.

4.3 First novel attack

After verification, we found a violation of formula F_5 (that is, freshness of R_1). The attack is reported in Table 1.

In this attack, the intruder starts replacing messages $\alpha.1$ with $\alpha.1'$ and $\alpha.3$ with $\alpha.3'$, and finally obtains r_1 from message $\alpha.4$. But, when it wants to use

Table 1
R_1 freshness attack. $\varepsilon(s)$ is ε masquerading as s. α and β denote two different runs.

Message α.1.	$a \to \varepsilon(s) \ : \ a, s, b, \{t_a, r_1\}_{pk(s)}$
Message α.1′.	$\varepsilon(a) \to s \ : \ a, s, b, \{t_{e1}, r_e\}_{pk(s)}$
Message α.2.	$s \to b \ : \ s, b, a$
Message α.3.	$b \to \varepsilon(s) \ : \ b, s, a, \{t_b, r_2\}_{pk(s)}$
Message α.3′.	$\varepsilon(b) \to s \ : \ b, s, a, \{t_a, r_1\}_{pk(s)}$
Message α.4.	$s \to \varepsilon(a) \ : \ s, a, b, v(r_e, r_1)$
Message β.1.	$\varepsilon(a) \to s \ : \ a, s, b, \{t_{e2}, r_1\}_{pk(s)}$

it in a new run β, even if the intruder uses a new (not expired) timestamp t_{e2}, the formula F_5 does not hold since r_1 is not fresh (note that s is the *same* server, involved in both runs α and β). It is important to notice why this attack represents a vulnerability of the protocol. According to Tatebayashi and Matsuzaki, the server has to check for the validity of the timestamps in order to guarantee the freshness of R_1 and R_2; as we can see in this attack, this is not sufficient. To the best of our knowledge, this vulnerability was never exposed before.

4.4 Second modification

A modification to assure authentication of the initiator and responder to the server consists in using S_A and S_B, shared secrets between S and A and B respectively, in the following manner:

$$
\begin{array}{lll}
\text{Message 1.} & A \to S & : \ A, S, B, \{T_A, S_A, R_1\}_{pk(S)} \\
\text{Message 2.} & S \to B & : \ S, B, A \\
\text{Message 3.} & B \to S & : \ B, S, A, \{T_B, S_B, R_2\}_{pk(S)} \\
\text{Message 4.} & S \to A & : \ S, A, B, v(R_1, R_2)
\end{array}
$$

After receiving messages 1 and 3, the server can authenticate A and B, respectively, since (by assumption) secrets S_A and S_B are shared *only* between the server and the respective agents. To check if the protocol accomplishes the authentication goal of A and B to S, we translate this in a formula that states that if S received a message M apparently from A (resp. B), then it was *really* sent by A (B). Server S authenticates A after receiving the first message, so at that point we set our formula: $F_5 = \exists e : e = \langle A : A, S, B, \{T_A, S_A, R_1\}_{pk(S)} \triangleright S\rangle$. Similarly, S authenticates B after the third message: $F_7 = \exists e : e = \langle B : B, S, A, \{T_B, S_B, R_2\}_{pk(S)} \triangleright S\rangle$.

We performed verification with some test scenarios and did not find any trace that violates the above security requirements. Thus, we can regard the protocol to be secure for the system scenarios we tested; of course, bigger scenarios can be tested to increase confidence about the protocol security.

4.5 Mutual authentication

Even though Tatebayashi and Matsuzaki do not state the mutual authentication of A and B, it is interesting to consider this case (Lowe and Roscoe [9] also discuss this.) We can translate this requirement by redefining two formulae, namely F_3 and F_2. We define F_3 to express the local security property of A to B and F_2 expressing the authentication of B to A:

- M authenticity of A to B: $F_3 = \exists e : e = \langle A : A, S, B, \{T_A, S_A, R_1\}_{pk(S)} \triangleright S\rangle$;
- M authenticity of B to A: $F_2 = \exists e : e = \langle B : B, S, A, \{T_B, S_B, R_2\}_{pk(S)} \triangleright S\rangle$.

Proceeding with verification, we found traces that violate F_2 and F_3. The attack trace for F_3 is straightforward, consisting in only one message, sent from $\varepsilon(s)$ to b: s, b, a. But this is sufficient to violate formula F_3, since when b receives s, b, a she wants to check if a sent $a, s, b, \{t_a, s_a, r_1\}_{pk(s)}$, which she did not (this attack is similar to attack 7.1 in [9].)

4.6 Novel authentication attacks

In Table 2 we report two attacks that violate F_2.

Table 2
B to A authentication attacks

$\alpha.1.\quad a \to \varepsilon(s) \;:\; a, s, b, \{t_a, s_a, r_1\}_{pk(s)}$	$\alpha.1. a \to \varepsilon(s) \;:\; a, s, b, \{t_a, s_a, r_1\}_{pk(s)}$
$\alpha.2.\quad \varepsilon(s) \to b \;:\; s, b, \varepsilon$	$\beta.1. \varepsilon \to s \;:\; \varepsilon, s, a, \{t_e, s_e, re_1\}_{pk(s)}$
$\alpha.3.\quad b \to \varepsilon(s) \;:\; b, s, \varepsilon, \{t_b, s_b, r_2\}_{pk(s)}$	$\beta.2. s \to \varepsilon(a) \;:\; s, a, \varepsilon$
$\beta.1.\quad \varepsilon(a) \to s \;:\; a, s, b, \{t_a, s_a, r_1\}_{pk(s)}$	$\beta.3. \varepsilon(a) \to s \;:\; a, s, \varepsilon, \{t_a, s_a, r_1\}_{pk(s)}$
$\beta.2.\quad s \to \varepsilon(b) \;:\; s, b, a$	$\beta.4. s \to \varepsilon \;:\; s, \varepsilon, a, v(re_1, r_1)$
$\beta.3.\quad \varepsilon(b) \to s \;:\; b, s, a, \{t_b, s_b, r_2\}_{pk(s)}$	$\alpha.4. \varepsilon(s) \to a \;:\; s, a, b, v(r_1, re_1)$
$\beta.4.\quad s \to \varepsilon(a) \;:\; s, a, b, v(r_1, r_2)$	
$\alpha.4.\quad \varepsilon(s) \to a \;:\; s, a, b, v(r_1, r_2)$	

The attack of Table 2 (left side) is successful since the intruder can manipulate the first three non-encrypted fields. Notice how F_2 is violated: when a receives message $\alpha.4$, b *never* sent message $b, s, a, \{t_b, s_b, r_2\}_{pk(b)}$. The attack reported in Table 2 (right side) is stronger, since the principal b is not alive in the run of a.

We believe these attacks over this modified version of the TMN protocol have never been reported before in the literature.

5 Related Work

In this section we discuss some related work. In [16], Roscoe identifies two ways of specifying protocol security goals: firstly, using *extensional* specifications, and secondly using *intensional* specifications. An extensional specification describes the intended service provided by the protocol in terms of behavioural equivalence [6,1,17]. On the other hand, an intensional specification describes the underlying mechanism of a procotol, in terms of states or events [2,21,16,19,15,7].

The approach presented in this paper belongs to the spectrum of intensional specifications, and is related to [16,19]. In [19], a requirement specification language is proposed. This language is useful for specifying sets of requirements for classes of protocol; the requirements can be mapped onto a particular protocol instance, which can be later verified using their tool, called NRL Protocol Analyzer. This approach has been subsequently used to specify the GDOI secure multicast protocol [12].

In [16], Roscoe presents a method for describing the underlying mechanism of a protocol, using a CSP specification. The method consists of four steps: Firstly, one identifies an execution point of the protocol that should not be reached without a corresponding legitimate run having occurred. Secondly, one describes the possible sequences of messages that should have occurred before this execution point; thirdly, one creates a specification which groups all the CSP processes modelling the protocol participants (this step is similar to our scenario setting). Finally, one verifies the specification using FDR. This method has been also used by Lowe in [9].

The approaches just mentioned employ languages specifying security properties in a global fashion, as opposed to our technique which deals with local security properties.

In [4], Cremers, Mauw and de Vink present another logic for specifying local security properties. Similarly to us, in [4] the authors define the message authenticity property by referring to the variables occurring in the protocol role. In addition, in [4], it is defined a new kind of authentication, called *synchonization*, which is then compared with the Lowe's intensional specification. The logic presented in this paper cannot handle the specification of the synchronization authentication. In fact, we cannot handle the weaker notion of injective authentication, since we cannot match corresponding events in a trace. However, we believe we can extend our logic to support these properties. Briefly, this could be achieved by decorating the different runs with label identifiers and adding a primitive to reason about events that happenned before others in a trace.

6 Conclusions and Future Work

We have developed a *trace logic* for expressing *local security properties*. Using this trace logic, the protocol designer can specify precisely the (local) security properties a protocol should satisfy to accomplish the security goals for which it has been designed.

The main differences between our approach and the ones mentioned in Section 5 can be summarized as follows:

(i) Our trace logic formulae are *local* to the participants, in the sense that are dependent to the principal's point of view, instead of *global* to the protocol specification. This allow us to define properties more precisely, in the sense of what should hold for each principal at each execution step.

Furthermore, our technique can be used to integrate the specification within the design of a cryptographic protocol. Methodologically, this allows for the integration of the verification phase within the design one, speeding up the feedback from the verification, and providing the basis for an integrated environment for protocol engineering.

(ii) Without having to use temporal operators, the logic we presented can express classical security properties including freshness and authenticity of the exchanged values during a protocol run.

(iii) Our logic is applied *directly* to the protocol messages. This allow us to reason about (local) security properties without having to use artificial *event* messages.

As future work, we plan to apply the methodology to more complex case studies, such as multicast protocols e.g. LKH group communications protocol [20]. We also plan to study how to *compose* local security specifications: we believe this is a very important advantage of our approach over the other global ones.

Acknowledgements. We would like to thank the anonymous reviewers for useful comments.

References

[1] M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, 1999.

[2] M. Abadi and B. Blanchet. Secrecy types for asymmetric communication. In *In Foundations of Software Science and Computation Structures*, volume 2030, pages 25–41. Springer-Verlag LNCS, 2001.

[3] R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In G. Puebla M. Hermenegildo, editor, *9th International Symposium, SAS 2002*, volume 2477 of *LNCS*, pages 326–342. Springer, 2002.

[4] C.J.F. Cremers, S. Mauw, and E.P. de Vink. Defining authentication in a trace model. In *Workshop on Formal Aspects in Security and Trust*, 2003.

[5] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

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

[7] J.D. Guttman and F.J. Thayer. Au thentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, 2002.

[8] G. Lowe. Casper: A compiler for the analysis of security protocols. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW '97)*, pages 18–30. IEEE, 1997.

[9] G. Lowe and A. W. Roscoe. Using CSP to detect errors in the TMN protocol. *Software Engineering*, 23(10):659–669, 1997.

[10] D. B. Newman Jr. M. Tatebayashi, N. Matsuzaki. Key distribution protocol for digital mobile communication systems. *Advances in Cryptology: Proceedings of Crypto '89*, 435:324–333, 1990.

[11] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[12] C. Meadows, P. Syverson, and I. Cervesato. Formalizing GDOI group key management requirements in NPATRL. In *Proceedings of the ACM Conference on Computer and Communications*, pages 235–244. ACM, November 2001.

[13] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 2001 ACM Conference on Computer and Communication Security*, pages 166 – 175. ACM press, 2001.

[14] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murϕ. In *Proceedings of the 1997 Conference on Security and Privacy*, pages 141–153. IEEE Press, 1997.

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

[16] A. W. Roscoe. Intensional specification of security protocols. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop*, pages 28–38. IEEE Computer Society Press, June 10-12 1996.

[17] S. Schneider. Security properties and CSP. In *IEEE Computer Society Symposium on Security and Privacy*, pages 174–190. IEEE Computer Society Press, 1996.

[18] G. J. Simmons. Cryptoanalysis and protocol failures. *Communications of the ACM*, 37(11):56–65, 1994.

[19] P. Syverson and C. Meadows. A formal language for cryptographic protocol requirements. *Designs, Codes, and Cryptography*, 7(1/2):27–59, 1996.

[20] C. K. Wong, M. Gouda, and S. S. Lam. Secure group communications using key graphs. *IEEE/ACM Transaction on Networking*, 8(1):16–30, 2000.

[21] T.Y.C. Woo and S.S. Lam. A semantic model for authentication protocols. In *In IEEE Computer Society Symposium on Research in Security and Privacy*, pages 178–194, 1993.