Automatic Abstraction Generation : How to Make an Expert Verification Technique for Security Protocols available to Non-expert Users
Yohan Boichut, Pierre-Cyrille Héam, Olga Kouchnarenko

To cite this version:

HAL Id: inria-00116918
https://hal.inria.fr/inria-00116918v2
Submitted on 30 Nov 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Automatic Abstraction Generation

—

How to Make an Expert Verification Technique for Security Protocols available to Non-expert Users

Yohan Boichut — Pierre-Cyrille Héam — Olga Kouchnarenko

N° 6039
Novembre 2006
Automatic Abstraction Generation
—
How to Make an Expert Verification Technique for
Security Protocols available to Non-expert Users

Yohan Boichut* , Pierre-Cyrille Héam**, Olga Kouchnarenko**

Thème SYM — Systèmes symboliques
Projet LANDE et CASSIS

Rapport de recherche n° 6039 — Novembre 2006 — 18 pages

Abstract: Reachability analysis is central for deciding the security problem for protocols
with an unbounded number of sessions. This problem is in general undecidable for infinite-
state systems. However, a solution may consist of performing reachability analysis on safety-
preserving abstractions of security protocols.

In order to make this technique available for high level specification languages like HLPSL
and PROUVÉ, we define safe and sound abstractions of protocol transition systems into
rewriting systems. These abstractions allow the automated generation of approximation
functions to ensure soundness of the reachability analysis. In addition, we propose a new
representation of secrecy properties, suitable to semi-decide the security problem.

As our main purpose is to automate as much as possible the analysis of protocols for
an unbounded number of sessions, our abstraction/approximation based approach provides
an efficient verification tool, TA4SP. This way, the requirement of an expert user can be
removed from the verification chain.

Key-words: Security protocols, safe and sound abstractions, verifications, approximations,
abstractions

* IRISA/Rennes, email: boichut@irisa.fr
** LIFC – Université de Franche-Comté, email: {heampc,kouchna}@lifc.univ-fcomte.fr
Génération automatique d’abstractions

Une technique experte de vérification de protocoles de sécurité rendue Boîte Noire

Résumé : L’analyse d’atteignabilité en réécriture a été adaptée à différents contextes de vérification, notamment, celui des protocoles de sécurité. Pour des systèmes infinis, le problème de sécurité (des protocoles) tout comme le problème d’atteignabilité est en général indécidable. L’utilisation d’approximations et d’abstractions est alors une alternative pour semi-décider ces problèmes. Mais leur définition et manipulation nécessitent souvent une certaine expertise.

Nous proposons dans cet article de connecter une technique de vérification fondée sur des approximations de réécriture sur de langages réguliers d’arbres aux langages de spécification de haut-niveau: HLPSL et PROUVÉ. Cette contribution a pour but de fournir une abstraction correcte d’un protocole HLPSL ou PROUVÉ pour ainsi appliquer la vérification que nous avons rendue automatique ultérieurement. Le tout est développé au sein de l’outil automatique TA4SP qui, à partir d’une spécification de protocole de haut-niveau peut établir un diagnostic de sûreté pour les propriétés de secret et ce pour un nombre quelconque d’exécutions.

Mots-clés : Protocoles de sécurité, vérification automatique, abstraction correcte, approximations, langages de haut-niveau
1 Introduction

Automated and computer-assisted verification and validation of cryptographic protocols are a fast-growing application field for formal methods and techniques in computer science. This is because cryptographic protocols are notably more difficult to understand for human designers and users. Indeed, any number of sessions (sequential or parallel executions) of protocols, sessions interleaving, any message size, algebraic properties of encryption or data structures give rise to infinite-state systems. In this case, when the number of sessions is unbounded, the security problem of cryptographic protocols is undecidable, even when the length of the messages is bounded [Da99].

Fortunately, today there exists a very successful approach to circumvent the infinite-state system analysis problem: to employ abstraction-based approximation methods [Mon99,GK00]. In fact, this approach uses regular tree languages to approximate the set of messages that the intruder might have acquired during an unbounded number of sessions of protocols. In this framework, most problems of protocol analysis reduce to various kinds of reachability problems on these models.

In [Ba05], we proposed and studied an abstract formal model for protocols which allows us to semi-decide the secrecy problem using automatically generated approximation functions. In practice, cryptographic protocols are often specified using high level protocol specification languages, for example PROUVÉ [Ka05] and HLPSL [Ca04]. Consequently, all works based on regular tree languages require the presence of an expert in order either to compute the approximations or to encode the protocol specification into tree automata and term rewriting systems.

Now a bridge must be built, linking real protocols specifications and theoretical problems on their sound abstractions. The main contribution of this paper consists of showing the feasibility of the automatic analysis of secrecy properties of protocols where the number of sessions is unbounded and when the protocol is given in a high level protocol specification language. The tool TA4SP - Tree Automata based on Automatic Approximations for the Analysis of Security Protocols - provides non-expert user a way to practically and easily verify security protocols in an unbounded context.

Layout of the paper The paper is organized as follows. Two high level protocol specification languages HLPSL and PROUVÉ are first introduced in Section 2. Then, some notions and notations are introduced in Section 3. In Section 4, we show how our verification technique is plugged on both languages and a new handling of secrecy properties as well for a technique based on [GK00] where they are specified as a tree automaton language. In Section 5, the soundness of the translation of an IF specification into our abstract model is shown. Finally, before concluding, the experiments, performed with the TA4SP tool, are presented in Section 6.

2 High Level Protocol Specification Languages

We consider cryptographic protocols specified either in HLPSL [Ca04] or in PROUVÉ [Ka05], two well-known high level protocol specification languages. The purpose of the PROUVÉ lan-
language is to give means to describe both protocols and the context in which they are used. The sub-language used to define protocols is oriented onto existing imperative programming languages and describes roles which are the programs executed by (legitimate) protocol participants. Roles are composed in so-called scenarios. The sub-language for scenarios contains constructs for parallel and sequential composition, and for non-deterministic choice of values. It has assignment statements, and allows the instantiations of roles that have been previously defined. The accompanying assertion language allows one to specify safety properties on execution traces of protocol scenarios.

The language HLPES, developed in the framework of the European Union project AVISPA \(^1\), is modular and allows the specification of control-flow patterns, data structures and different intruder models. Its semantics are based on Lamport’s TLA (Temporal Logic of Actions [Lam94]) which makes an HLPES specification easily translatable into a declarative lower-level term rewriting based language \(\text{IF}(\text{Intermediate Format}, [AVI03])\) well-suited to automated analysis tools [Aa05].

In [Ba06b], a translation of a PROUVÉ specification into an IF specification is described. As the IF language is connected to the high level languages PROUVÉ and HLPES, we choose the language IF as the input language of our verification technique.

3 Preliminaries

This section recalls some usual notions on terms, tree automata and term rewriting systems (TRSs for short) required to describe our verification technique. A comprehensive survey on TRSs and tree automata can be found for example in [Ca02].

3.1 Terms, Term Rewriting Systems and Tree Automata

Let \(\mathcal{F}\) be a finite set of symbols, each one with an arity, denoted \(\text{Arity}(f)\) for \(f \in \mathcal{F}\), and let \(\mathcal{X}\) be a countable set of variables. \(\text{T}(\mathcal{F}, \mathcal{X})\) denotes the set of terms, and \(\text{T}(\mathcal{F})\) denotes the set of ground terms (terms without variables). The set of variables of a term \(t\) is denoted by \(\text{Var}(t)\). A substitution is a function \(\sigma\) from \(\mathcal{X}\) into \(\text{T}(\mathcal{F}, \mathcal{X})\), which can be uniquely extended to an endomorphism of \(\text{T}(\mathcal{F}, \mathcal{X})\). A position \(p\) for a term \(t\) is a word over \(\mathbb{N}\). The empty sequence \(\epsilon\) denotes the top-most position. The set \(\text{Pos}(t)\) of positions of a term \(t\) is inductively defined by:

- \(\text{Pos}(t) = \{\epsilon\}\) if \(t \in \mathcal{X}\)
- \(\text{Pos}(f(t_1, \ldots, t_n)) = \{\epsilon\} \cup \{i.p \mid 1 \leq i \leq n = \text{Arity}(f)\text{ and } p \in \text{Pos}(t_i)\}\)

If \(p \in \text{Pos}(t)\), then \(t|_p\) denotes the subterm of \(t\) at position \(p\) and \(t[s]_p\) denotes the term obtained by replacement of the subterm \(t|_p\) at position \(p\) by the term \(s\). For any term \(s \in \text{T}(\mathcal{F}, \mathcal{X})\), we denote by \(\text{Pos}_f(s)\) the set of functional positions in \(s\), i.e. \(\{p \in \text{Pos}(s) \mid \text{Root}(s|_p) \in \mathcal{F}\}\) where \(\text{Root}(t)\) denotes the symbol at position \(\epsilon\) in \(t\). More generally,

\(^1\) [http://www.avispa-project.org/]

INRIA
we denote \( t(p) \) the functional symbol of \( t \) located at position \( p \). We denote by \( \text{Pos}_x(s) \) the set of positions of variable \( x \in \mathcal{X} \) occurring in \( s \), i.e. the set \( \{ p \in \text{Pos}(s) \mid s|_p = x \} \).

A TRS \( \mathcal{R} \) is a set of rewrite rules \( l \rightarrow r \), where \( l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X}), l \notin \mathcal{X} \), and \( \text{Var}(l) \supseteq \text{Var}(r) \). A rewrite rule \( l \rightarrow r \) is left-linear (resp. right-linear) if each variable of \( l \) (resp. \( r \)) occurs only once in \( l \) (resp. in \( r \)). A rule is linear if it is both left and right-linear. A TRS \( \mathcal{R} \) is linear (resp. left-linear, right-linear) if every rewrite rule \( l \rightarrow r \) of \( \mathcal{R} \) is linear (resp. left-linear, right-linear). The TRS \( \mathcal{R} \) induces a rewriting relation \( \rightarrow_{\mathcal{R}} \) on terms whose reflexive transitive closure is denoted by \( \rightarrow_{\mathcal{R}}^* \). The set of \( \mathcal{R} \)-descendants of a set of ground terms \( E \) is \( \mathcal{R}^*(E) = \{ t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in E \text{ s.t. } s \rightarrow_{\mathcal{R}}^* t \} \).

Let \( \mathcal{Q} \) be a possibly infinite set of symbols of arity 0, called states such that \( \mathcal{Q} \cap \mathcal{F} = \emptyset \). \( \mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \) is called the set of configurations.

**Definition 1 (Transition and normalized transition).** A transition is a rewrite rule \( c \rightarrow q \), where \( c \) is a configuration i.e. \( c \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \) and \( q \in \mathcal{Q} \). A normalized transition is a transition \( c \rightarrow q \) where \( c = f(q_1, \ldots, q_n) \), \( f \in \mathcal{F} \), \( \text{Arity}(f) = n \), and \( q_1, \ldots, q_n \in \mathcal{Q} \), or \( c \in \mathcal{F}_0 \).

Bottom-up non-deterministic finite tree automata are then defined as follows.

**Definition 2.** A bottom-up non-deterministic finite tree automaton (tree automaton for short) is a quadruple \( \mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta) \), where \( \mathcal{Q}_f \subseteq \mathcal{Q} \) and \( \Delta \) is a set of normalized transitions.

The rewriting relation on \( \mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \) induced by the transitions of \( \mathcal{A} \) (the set \( \Delta \)) is denoted by \( \rightarrow_{\Delta} \). When \( \Delta \) is clear from the context, \( \rightarrow_{\Delta} \) will also be denoted by \( \rightarrow_{\mathcal{A}} \). Similarly, by notation abuse, we often note \( q \in \mathcal{A} \) and \( t \rightarrow q \in \mathcal{A} \) respectively for \( q \in \mathcal{Q} \) and \( t \rightarrow q \in \Delta \).

**Definition 3 (Recognized language).** The tree language recognized by \( \mathcal{A} \) in a state \( q \) is \( \mathcal{L}(\mathcal{A}, q) = \{ t \in \mathcal{T}(\mathcal{F}) \mid t \rightarrow_{\Delta}^* q \} \). The language recognized by \( \mathcal{A} \) is \( \mathcal{L}(\mathcal{A}) = \bigcup_{q \in \mathcal{Q}_f} \mathcal{L}(\mathcal{A}, q) \). A tree language is regular if and only if it can be recognized by a tree automaton.

### 3.2 Security Protocols Verification over Reachability Analysis

This section recalls the approximation-based framework we have been developing. Given a tree automaton \( \mathcal{A} \) and a TRS, the tree automata completion algorithm, initially presented in [GK00] and then used in [Fa04], computes a tree automaton \( \mathcal{A}_k \) such that \( \mathcal{L}(\mathcal{A}_k) = \mathcal{R}^*(\mathcal{L}(\mathcal{A})) \) when it is possible (for the classes of TRSs covered by this algorithm see [Fa04]), and such that \( \mathcal{L}(\mathcal{A}_k) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A})) \) otherwise.

The tree automata completion works as follows. From \( \mathcal{A} = \mathcal{A}_0 \), the completion builds a sequence \( \mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_k \) of automata such that if \( s \in \mathcal{L}(\mathcal{A}_i) \) and \( s \rightarrow_{\mathcal{A}} t \) then \( t \in \mathcal{L}(\mathcal{A}_{i+1}) \). If we find a fix-point automaton \( \mathcal{A}_k \) such that \( \mathcal{R}^*(\mathcal{L}(\mathcal{A}_k)) = \mathcal{L}(\mathcal{A}_k) \), then we have \( \mathcal{L}(\mathcal{A}_k) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \) (or \( \mathcal{L}(\mathcal{A}_k) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A})) \) if \( \mathcal{R} \) is not in any class of [Fa04]). To build \( \mathcal{A}_{i+1} \) from \( \mathcal{A}_i \), we achieve a completion step which consists of finding critical pairs between \( \rightarrow_{\mathcal{R}} \) and \( \rightarrow_{\mathcal{A}_i} \). For a substitution \( \sigma : \mathcal{X} \rightarrow \mathcal{Q} \) and a rule \( l \rightarrow r \in \mathcal{R} \), a critical pair is an instance \( l \sigma \rightarrow_{\mathcal{A}_i} q \) and \( r \sigma \rightarrow_{\mathcal{A}_i} q \). For every critical pair \( l \sigma \rightarrow_{\mathcal{A}_i} q \) and
\( r \sigma \xrightarrow{*} q \) detected between \( \mathcal{R} \) and \( \mathcal{A}_i \), \( \mathcal{A}_{i+1} \) is constructed by adding new transitions to \( \mathcal{A}_i \) such that it recognizes \( r \sigma \) in \( q \), i.e. \( r \sigma \rightarrow \mathcal{A}_{i+1} \).

\[
\begin{array}{c}
\mathcal{R} \\
\mathcal{A}_i \\
q \\
\end{array} 
\xrightarrow{r \sigma} 
\begin{array}{c}
\mathcal{A}_{i+1} \\
n \\
\end{array}
\]

However, the transition \( r \sigma \rightarrow q \) is not necessarily a normalized transition. The normalization of such a transition is performed by an approximation function \( \alpha \) and this may lead either to an exact computation of \( \mathcal{R}^*(\mathcal{A}) \), or to an over-approximation of \( \mathcal{R}^*(\mathcal{A}) \).

The method presented in [GK00] requires as input a tree automaton \( \mathcal{A}_0 \), a TRS \( \mathcal{R} \) and a tree automaton \( \mathcal{A}_{secret} \) where:

- the language of \( \mathcal{A}_0 \) represents both the initial knowledge of the intruder and the initial configuration of the network;
- \( \mathcal{R} \) specifies the intruder abilities to compose and analyze messages broadcast over the network, and each step of the studied protocol also;
- the language of \( \mathcal{A}_{secret} \) represents the set of terms supposed to remain secret while executing the protocol.

In the context of the verification of protocols, the security problem consists of deciding whether a protocol preserves secrecy against an intruder, or not. The finite tree automata permit us to ensure that some states are unreachable, and hence that the intruder will never be able to know certain terms.

Our previous work [Ba05] describes an automated method for computing over-approximations or under-approximations of the intruder knowledge from a tree automaton \( \mathcal{A}_0 \) and a TRS \( \mathcal{R} \). To ensure the soundness of the verification, some criteria are defined for both \( \mathcal{A}_0 \) and \( \mathcal{R} \), and two classes of approximations are defined also such that the criteria keep holding during the verification.

These criteria concern a particular class of terms which are called atomic terms or atomic data.

In [Ba05], the set of constants and variables are finite, and each constant and variable is associated to one \( \text{lF} \)-type which can be \text{agent}, \text{public_key}, \text{symmetric_key}, \text{text}, \text{nat}, \text{function}, \text{bool}, \text{message}, \text{fact}, \text{set} \) or \text{identifier}.

\textbf{Definition 4.} Let \( \mathcal{F}_{\text{abs}} \) be a finite set of special functional symbols used for data abstraction. An atomic term or atomic data is either a constant or a variable whose type is different from message and fact, or a term \( f(t_1, \ldots, t_n) \) representing fresh data of the types text, symmetric_key or public_key, where \( t_1, \ldots, t_n \) are of type agent or nat, and \( f \in \mathcal{F}_{\text{abs}} \).

By definition and by hypothesis on the finiteness of the sets of variables and constants, the set of atomic data is finite too. Each atomic type is associated to one particular safety functional symbol. We denote \( \mathcal{F}_{\text{prct}} \) the set of safety symbols, and \( \mathcal{F}_{\text{abs}} \cap \mathcal{F}_{\text{prct}} = \emptyset \). For a term \( t \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \), the notion of greatest atomic data (GAD), given in Definition 5, allows one to locate each atomic datum in the term \( t \).

INRIA
Definition 5. Let $t$ be a term of $T(F, X)$ such that there exists a position $p$ such that $t|_p$ is an atomic datum. If there is $p' \in \text{Pos}(t)$ such that $p = p'.i$ with $i \in \mathbb{N}$ and $t|_{p'}$ is not an atomic datum, then $t|_{p'}$ is a greatest atomic datum of $t$. The set of positions of the greatest atomic data of the term $t$ is denoted $\text{Pos}_{GAD}(t)$.

To illustrate the above definition, let us consider a simple example. Let $t$ be the term $pair(a, n(a, b))$ where $a$, $b$ and $n(a, b)$ are atomic data. Thus, $\text{Pos}_{GAD}(t) = \{1, 2\}$ where the position 1 is related to the GAD $a$, and 2 is related to the GAD $n(a, b)$.

Now, the intuition behind the criteria mentioned above and used in the remainder of the paper is:

- An atomic term can be rewritten into a unique special allotted state within the initial tree automaton (conditions on $A_0$);
- An atomic term must be protected by a special allotted functional symbol (conditions on $R$, $A_0$ and the approximation function).

Formally, let $t_{\text{atomic}}$ be an atomic datum of type $ty$ whose safety symbol is denoted $sf_{ty}(\in F_{\text{prot}})$. Let $q_{t_{\text{atomic}}}$ be the allotted state of $t_{\text{atomic}}$ when $t_{\text{atomic}} \in T(F)$. There are four criteria on $A_0$. If $t_{\text{atomic}} \in T(F)$,

[A-1] $\mathcal{L}(A_0, q_{t_{\text{atomic}}}) = \{t_{\text{atomic}}\}$;

[A-2] For all $q \in A_0$, if $t_{\text{atomic}} \in \mathcal{L}(A_0, q)$ then $q = q_{t_{\text{atomic}}}$.

[A-3] For all $q \in A_0$, $t \in \mathcal{L}(A_0, q)$ and $p \in \{p' \in \text{Pos}(t) \mid p' \in \text{Pos}_{GAD}(t) \land t|_{p'} = t_{\text{atomic}}\}$

either $p = \epsilon$ and $q = q_{t_{\text{atomic}}}$, or $p = s.1$, $t(s) = sf_{ty}$.

[A-4] For all $q \in A_0$, $t \in \mathcal{L}(A_0, q)$, if there exists $p \in \text{Pos}(t)$ such that $t(p) = sf_{ty}$ then $p.1 \in \text{Pos}_{GAD}(t)$ and the type of $t|_{p.1}$ is $ty$.

Concerning $R$, the criteria are as follows: for all $l \rightarrow r \in R$, $t \in \{l, r\}$ and $p \in \{p' \in \text{Pos}(t) \mid p' \in \text{Pos}_{GAD}(t) \land t|_{p'} = t_{\text{atomic}}\}$,

[R-1] $p \neq \epsilon$,

[R-2] $t(s) = sf_{ty}$ and $p = s.1$.

When the conditions [A-1]-[A-4] and [R-1]-[R-2] are satisfied on both $A_0$ and $R$, two kinds of approximation functions can be generated, which lead towards either a sound under-approximation or a sound over-approximation of the intruder knowledge.

4 How to Verify Security Protocols from High Level Protocols

Specification Languages

As mentioned in Section 2, I1F is directly connected to the high level specification languages HLPSL and PROUVÉ. The method presented in [GK00, Fa04] and [Ba05] requires an input in the form of a tree automaton $A_0$, a TRS $R$ and a tree automaton $A_{\text{secret}}$.

To link this verification method to I1F, we explain in this section how to automatically generate the tree automaton $A_0$ and the TRS $R$ from an I1F specification. This way, an abstract model of the protocol to be checked can be built automatically. Moreover, in order to express secrecy properties, a new technique is developed and a new means to check them is given.
<table>
<thead>
<tr>
<th>Type of t</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent</td>
<td>agt(t)</td>
</tr>
<tr>
<td>public_key</td>
<td>pk(t)</td>
</tr>
<tr>
<td>symmetric_key</td>
<td>sk(t)</td>
</tr>
<tr>
<td>text</td>
<td>text(t)</td>
</tr>
<tr>
<td>function</td>
<td>func(t)</td>
</tr>
<tr>
<td>identifier</td>
<td>ident(t)</td>
</tr>
<tr>
<td>nat</td>
<td>nat(t)</td>
</tr>
<tr>
<td>set</td>
<td>set(t)</td>
</tr>
<tr>
<td>bool</td>
<td>bool(t)</td>
</tr>
</tbody>
</table>

Table 1. Translating an IF atomic datum $t$

4.1 A Particular Processing for Atomic Data

Recall that for an IF specification of a protocol, the sets of constants and variables are both finite. In IF and in [Ba05] as well, there is a particular type for each constant and variable. The available types are agent, public_key, symmetric_key, text, nat, function, bool, message, fact, set or identifier.

Following [Ba05], we consider here the notion of atomic data given in Definition 4, and we adapt this notion to fresh data in IF, which are specified by existentially quantified variables as presented in Section 4.2.

A first step towards an abstract model consists of translating atomic data as shown in Table 1. Using Table 1, a functional symbol per type is introduced. Thus the set $\mathcal{F}_{\text{prot}}$ is built.

The terms of the type message or fact are built using the constructors whose signatures are listed in Table 2.

Notice that all types corresponding to atomic data are subtypes of the type message. Consequently, a term $\text{iknows}(a)$ where the type of $a$ is agent, is well-formed, i.e., it complies with the relation between types. The term $\text{iknows}(\text{iknows}(a))$ is not a well-formed term. In order to define the local state of an agent, a constructor complying with the last line of Table 2 is used.

To describe the evolution of all agents during a protocol run and the whole protocol, it remains to generate an abstract TRS $\mathcal{R}$ from an IF TRS.

4.2 Generation of a TRS $\mathcal{R}$ from an IF TRS

In an IF TRS, there are two kinds of rules: 1) protocol-dependent rules, and 2) protocol-independent rules. The rules in 1) describe the protocol, and the rules in 2) specify the intruder model. Notice that the default intruder model used in HPLS!, and in IF also, is the Dolev and Yao model [DY83]. The user does not specify the protocol-independent rules.
Concerning the protocol dependent part of the IF TRS, i.e., the description of the protocol, the protocol-dependent rules specify the reception or the sending of a message and a computation performed on his own knowledge from the point of view of an agent.

\[
\text{state}_{\text{Alice}}(A, B, \text{Kab}, D, M, 0).
\text{iknows}(\text{start})
= \text{[exists M]} \Rightarrow
\text{state}_{\text{Alice}}(A, B, \text{Kab}, M, 1).
\text{iknows} (\text{pair(Kab, scrypt(Kab, M))}).
\text{secret}(M, id_1, \{A, B\})
\]

\[
\text{state}_{\text{Bob}}(B, A, \text{Kab}, D, M, 0).
\text{iknows}(\text{pair(Kab, scrypt(Kab, M))})
= \Rightarrow
\text{state}_{\text{Bob}}(A, B, \text{Kab}, M, 1)
\]

crypt: message \times message \rightarrow message
scrypt: message \times message \rightarrow message
pair: message \times message \rightarrow message
apply: message \times message \rightarrow message
inv: message \rightarrow message
exp: message \times message \rightarrow message
xor: message \times message \rightarrow message
iknows: message \rightarrow \text{fact}
secret: message \times \text{identifier} \times \text{SET} \rightarrow \text{fact}
\text{SET} \quad \text{is a set of variables whose type is agent}
f \in \mathcal{F}: \text{ty}_1 \times \ldots \times \text{ty}_n \rightarrow \text{fact}
\text{with } \text{ty}_1, \ldots, \text{ty}_n \neq \text{fact}

Table 2. Signatures of constructors

For example, the two rules above specify a protocol where an agent A sends a message to the agent B. This message contains a symmetric key Kab and a nonce M encoded with Kab. In this example, A, B, Kab, D, M, 0, 1, id1 and start are atomic data. Notice that the first rule is existentially quantified with the notation \text{[exists M]} \Rightarrow, specifying the nonce generation.

As stated in Section 4.1, the translation of all atomic data according to Table 1 is performed. However, the rules obtained may not satisfy the condition \( \text{Var}(r) \subseteq \text{Var}(l) \) for a rewrite rule \( l \rightarrow r \). For example, this is the case when a variable is existentially quantified. The condition above being crucial for our verification technique [Ba05], the notion of valid abstraction is introduced in Definition 6.
Definition 6. Let $\mathcal{R}_{\text{IF}}$ be a set of $\text{IF}$ rules. Let $m \xrightarrow{\text{exists } X_1 \ldots X_n} m'$ be a rule of $\mathcal{R}_{\text{IF}}$. For $i = 1, \ldots, n$, $f_i(t_1, \ldots, t_k)$ is a valid abstraction of $X_i$ if $f_i \in \mathcal{F}_{\text{abs}}$ is not an $\text{IF}$ functional symbol, and $t_1, \ldots, t_k$ are some atomic data in $m$ of type agent or nat.

Using this definition in our running example, the variable $M$ is replaced by a valid abstraction. This could be $n(A, B)$ in the first rule. Moreover, since the number of naturals and agents in an $\text{IF}$ specification is bounded, there exists a finite number of instantiations of the valid abstraction of the data $M$. Proposition 1 claims that it is always possible to define a valid abstraction for a fresh variable.

**Proposition 1.** Let $\mathcal{R}_{\text{IF}}$ be a set of $\text{IF}$ rules. Let $m \xrightarrow{\text{exists } X_1 \ldots X_n} m$ be a rule of $\mathcal{R}_{\text{IF}}$. For each $i=1, \ldots, n$, there exists $t_{X_i} \in T(\mathcal{F}, \mathcal{X})$ such that $t_{X_i}$ is a valid abstraction of $X_i$.

**Proof (Sketched).** Let $\text{Atomic}(m)$ be the set of atomic terms of $m$ built as follows:

$$\text{AgtNats}(m) = \{ t \in T(\mathcal{F}, \mathcal{X}) \mid t = m|_p, p \in \mathcal{P}_{\text{GAD}}(m) \text{ and the type of } t \text{ is either agent or nat}. \}
$$

If $\text{AgtNats} = \emptyset$ then the valid abstraction is a constant. Otherwise, one can build a valid abstraction using some elements of $\text{AgtNats}$.

The following proposition shows that the number of instantiations of a valid abstraction is finite.

**Proposition 2.** Let $\mathcal{R}_{\text{IF}}$ be a set of $\text{IF}$ rules. Let $m \xrightarrow{X_1 \ldots X_n} m$ be a rule of $\mathcal{R}_{\text{IF}}$ and $t_{X_i} \in T(\mathcal{F}, \mathcal{X})$ be a valid abstraction of the variable $X_i$. The set

$$\{ t \in T(\mathcal{F}) \mid \exists \rho : \text{Var}(t_{X_i}) \rightarrow T(\mathcal{F}), \text{ s.t. for } x \in \text{Var}(t_{X_i}), t_{X_i}\rho = t \}
$$

is finite.

**Proof.** A consequence of the finiteness of the set of the atomic terms.

From now on, we consider that $t'$, a valid abstraction of an atomic data $t$, inherits of the type of $t$. Consequently, $n(A, B)$ has the same type as the variable $M$, and $n(A, B)$ is thus considered as an atomic datum. For the running example, the rules below are obtained by substituting the fresh variable $M$ by its valid abstraction $n(A, B)$ on the TRS resulting from the first translation step.
state_Alice(agt(A), agt(B), sk(Kab), text(D_M), nat(0)).
\text{iknows(text(start))} \Rightarrow \text{state_Alice(agt(A), agt(B), sk(Kab), text(n(A, B)), nat(1)).}
\text{iknows(pair(sk(Kab), scrypt(sk(Kab), sk(n(A, B))))).}
\text{secret(text(n(A, B))), ident(id1), \{agt(A), agt(B)\}}
\text{state_Bob(agt(B), agt(A), sk(Kab), text(D_M), nat(0)).}
\text{iknows(pair(sk(Kab), scrypt(sk(Kab), text(n(\text{text(M))))))}
\Rightarrow \text{state_Bob(agt(A), agt(B), sk(Kab), text(M), nat(1))}

Finally, the TRS \( R \) below is a representation of both rules above where the operator '·' is replaced by the binary functional symbol \( \text{and} \), and the secrecy fact decorates the rule.

\[
1) \quad \text{and(state}_\_\text{Alice(agt(A), agt(B), sk(Kab), text(D}_\_\text{M)}, \text{nat(0))}, \text{iknows(text(start)))} \\
\quad \text{secret(text(n(A, B))), ident(id1), \{agt(A), agt(B)\}} \\
\quad \text{and(state}_\_\text{Alice(agt(A), agt(B), sk(Kab), text(n(A, B)), nat(1)),} \\
\quad \text{iknows(pair(sk(Kab), scrypt(sk(Kab), sk(n(A, B))))))} \\
\quad \text{and(state}_\_\text{Bob(agt(B), agt(A), sk(Kab), text(D}_\_\text{M)}, \text{nat(0)),} \\
\quad \text{iknows(pair(sk(Kab), scrypt(sk(Kab), text(M))))} \\
\quad \text{\( \bar{\delta} \rightarrow \text{state}_\_\text{Bob(agt(A), agt(B), sk(Kab), text(M), nat(1))} \)}
\]

The following proposition issues from the construction of \( R \) from an IF TRS \( R_{IF} \).

**Proposition 3.** Let \( \text{Spec}_{IF} \) be an IF specification. Let \( R_{IF} \) be the TRS of \( \text{Spec}_{IF} \) and \( R \) its translation. Then, \( R \) satisfies Conditions [R-1] and [R-2] in Section 3.2.

**Proof (Sketched).** By definition, for an IF rule \( m \rightarrow m' \), \( m \) and \( m' \) are not atomic data. Consequently, Condition [R-1] is trivially satisfied. According to Table 1, each atomic datum occurring either in \( m \) or \( m' \) corresponds to a particular safety functional symbol. Moreover, concerning the valid abstractions for fresh variables, a fresh variable is firstly protected with a safety functional symbol according to Table 1. Secondly, it is substituted by a valid abstraction. This implies Condition [R-2].

Let us emphasize the fact that not only does the TRS \( R \) represent the protocol, but it also specifies Dolev and Yao's intruder model [DY83]. Like in [Ba03c, Ca01a], non atomic keys (keys computed in several steps) are not handled here.

In this section, we have presented a way to automatically generate a TRS, an input for the verification procedure. Usually, this task is tedious and very error-prone. Our contribution removes this difficult task, by making this step fully invisible for a non-expert user.
4.3 Generation of $A_0$ from an $IF$ Initial State

For an $IF$ specification $Spec$, its initial state represents the intruder knowledge and the initial state of all agents. An example of an initial state for the made-up example introduced in Section 4.2 could be specified by the term below.

\[
i\text{knows}(a).
\]
\[
i\text{knows}(b).
\]
\[
i\text{knows}(i).
\]
\[
\text{state}_{Alice}(a,b,kab,dummy\_nonce,0).
\]
\[
\text{state}_{Bob}(b,a,kab,dummy\_nonce,0)
\]

The identity of the intruder is denoted by $i$. To generate the initial tree automaton, the first step is to translate this term into the term $t_0$ below using the syntax of Section 4.1.

\[
\text{and}(\text{knows}(\text{agt}(a)),
\text{and}(\text{knows}(\text{agt}(b)),
\text{and}(\text{agt}(i)),
\text{and}(\text{state}_{Alice}(\text{agt}(a),\text{agt}(b),\text{sk}(kab),\text{text}(\text{dummy\_nonce}),\text{nat}(0)),
\text{state}_{Bob}(\text{agt}(b),\text{agt}(a),\text{sk}(kab),\text{text}(\text{dummy\_nonce}),\text{nat}(0)))))
\]

Afterwards, for each subterm $t$ of $t_0$, a set of transitions is generated as follows. We distinguish two particular states $q_f$ (the final state of $A_0$) and $q_{intruder}$ (the state for the knowledge of the intruder).

- If $\text{Root}(t)$ is $\text{knows}$ then the transition $\text{knows}(q_{intruder}) \rightarrow q_f$ is added and for the direct subterm $t_1$, $t_1 = f(t_1, \ldots, t_n)$, a transition $f(q_{t1}, \ldots, q_{tn}) \rightarrow q_{intruder}$ is added. Next, the process is applied to the terms $t_1, \ldots, t_n$ by considering the following cases.
- If $\text{Root}(t)$ is $\text{state}_{Alice}$, or $\text{state}_{Bob}$, the transition $\text{Root}(t)(q_{t1}, \ldots, q_{tn}) \rightarrow q_f$ is produced, with $\text{Root}(t) \in \mathcal{F}$ and $\text{Arity} (\text{Root}(t)) = n$.
- For other subterms not handled in the first case and such that $t(e) \neq \text{and}$ with $\text{Arity}(\text{Root}(t)) = n$, the transitions complying with the pattern $\text{Root}(t)(q_{t1}, \ldots, q_{tn}) \rightarrow q_f$ are generated.

Finally, the transition $\text{and}(q_f, q_f)$ which aims at representing the conjunction of facts concerning the intruder or the local states of the agents, is added.

Other transitions concerning the atomic data are produced as follows. For each atomic constant $t$, the state $q_t$ is allotted to it, and the transition $t \rightarrow q_t$ is added. For $t_0$, this boils down to adding the following transitions: $a \rightarrow q_a, b \rightarrow q_b, i \rightarrow q_i, kab \rightarrow q_{kab}, 0 \rightarrow q_0, 1 \rightarrow q_1, dummy\_nonce \rightarrow q_{dummy\_nonce} \text{ and } idl \rightarrow q_{idl}$.

Concerning the instantiations of the valid abstractions, in order to get normalized transitions, the allotted states are used for the atomic data inside, and a new state is allotted for the whole term. For instance, for the valid abstraction $n(A, B)$, the possible instantiations are $\{n(A, B) \mid A, B \in \{a, b, i\}\}$ and this set is denoted by $\text{Inst}(n(A, B))$. For each term $t$ of $\text{Inst}(n(A, B))$, the transition $n(q_{t1}, q_{t2}) \rightarrow q_t$ is added. Concretely, for the instance $n(a, b)$, the transition $n(q_a, q_b) \rightarrow q_{n(a,b)}$ is added to $A_0$.

As mentioned in Section 3.2, the initial tree automaton represents the initial knowledge of the intruder and the initial state of each agent. Some abilities of the intruder are
also specified with the initial tree automaton. For example, to specify the intruder abilities to compose terms, the following transitions are added: pair(qintruder, qintruder) → qintruder, scrypt(qintruder, qintruder) → qintruder and iknows(qintruder) → qf.

The following proposition claims that the generated tree automaton complies with the conditions given in Section 3.2.

**Proposition 4.** Let SpecIF be an IF specification. Let tIF be the term representing the initial state of SpecIF and A0 be the tree automaton resulting from the translation. Then, A0 satisfies Conditions [A-1], [A-2], [A-3] and [A-4].

**Proof (Sketched).** The proof is done by cases on Conditions on tree automata given in Section 3.2:

[A-1 ] Trivially, the specific transition construction for the atomic data ensures this property: for an atomic datum t, the transition t → qf is built. This keeps holding for atomic terms representing fresh data.

[A-2 ] By construction, this condition is true since for an atomic term t, qf is used in the right hand side of a rule, only for recognizing t.

[A-3 ] This transition is trivially satisfied for a state qf where t is an atomic term. In addition, this condition must be checked for the transitions coming from the translation of the term tIF. Condition [A-3] is satisfied too, since the rules given in Table 1 are applied before beginning the translation into transitions.

[A-4 ] Finally, this condition needs to be checked for the transitions coming from the translation of the term tIF. Indeed, transitions complying with the pattern sf(tf(q1)) → q2 where q1, q2 ∈ A0 cannot be introduced elsewhere. The satisfaction of Condition [A-4] by A0 is the result of the application of the rules given in Table 1 before translating tIF into A0.

Finally, A0 satisfies all the conditions previously mentioned.

In this section, we have presented a way for automatically generating a tree automaton A0. In practice, this task is error-prone and requires a certain knowledge of tree automata. It also requires expertise in the verification technique itself [GK00] in order to perform a sound verification of the protocol. Again, our contribution removes these difficulties.

### 4.4 New Representation of Secrecy Properties

In [GK00], secrecy properties are represented as the language of a tree automaton Asecret. Thus computing the intersection between L(Asecret) and L(Ak), representing an over-approximation of the intruder knowledge, allows to decide the secrecy problem: the protocol is safe if the intersection is empty. For this, the set of secret terms must be previously known. However it is not always the case. Indeed, expert users are able to anticipate what the secret terms will be during the computation, but for non-trivial secrets, this task remains tedious.

In HPSL and in IF as well, a secrecy property p is defined using secret facts of the form secret(tsecret, p, Agents) where tsecret ∈ T(F, X) is a pattern for secret terms concerning the property p, and Agents is the set of agents allowed to know this secret.
Here, we propose a new general representation of secrecy properties based on decorated rewrite rules introduced in Section 4.2.

Let $A$ be a finite tree automaton. Let $R_{IF}$ be an IF TRS and $R$ be the TRS resulting from the translation of $R_{IF}$. For the decorated TRS $R$ and for the tree automaton $A$, the set of secret terms concerning a property $p$, denoted $KT(p)$, can be built in the following way:

$$KT(p) = \{ t \in T(F \cup Q) \mid \exists r \in R, \sigma : \text{Var}(l) \to Q, t' \in \text{Sec},$$

$$t' = \text{secret}(t_{\text{secret}}, p, \text{Agents}), \text{ and } q \in Q \text{ such that }$$

$$l_\sigma \rightarrow^* q, r_\sigma \rightarrow^* q \text{ and } t = \text{secret}(t_{\text{secret}} p, \text{Agents}_\sigma) \}.$$

Since $A$ is a finite tree automaton, the set of substitutions $\text{Var}(l) \to Q$, with $l \xrightarrow{\text{Sec}} r \in R$ is also finite. Consequently, $KT(p)$ is a finite set.

Given a tree automaton $A$, we say that $A$ satisfies a property $p$ if the intruder does not know a secret term in $KT(p)$ which it is not supposed to know. The new algorithm below allows the decision of the satisfaction of a secrecy property $p$ for a tree automaton $A$.

**Algorithm 1**  The function verification returns true if $A$ satisfies the property $p$.

```plaintext
verification(A, KT(p))
Begin
res := true;
For all secret(t_data,p,S_agent) \in KT(p) do
    If the intruder is in S_agent then
        If t_data \rightarrow^* q_{\text{intruder}} then res := false EndIf
    EndIf
EndFor
return(res)
End
```

The algorithm above terminates for every finite tree automaton $A$.

**Proposition 5.** Let $A = (F, Q, Q_f, \Delta)$ be a finite tree automaton. Let $R$ be a TRS resulting from the translation of an IF TRS. verification $(A, p, R)$ terminates for every property $p$.

For an unbounded number of sessions, semi-deciding the secrecy problem for a protocol is different from deciding the satisfaction of a property $p$ by a tree automaton $A$. Theorem 2 links these two problems according the language of $A$.

**Theorem 2.** Let $p$ be a secrecy property and $A$ an automaton obtained by the completion of the tree automaton $A_0$. According to the language of $A$, the secrecy problem is semi-decided in the following way:

- $L(A) \subseteq R^*(L(A_0))$: verification$(A, p, KT) = false$ means that there exists an attack against the property $p$ with the model $R$ and for the given initial automaton $A_0$. 

INRIA
- $R^*(\mathcal{L}(A_0)) \subseteq \mathcal{L}(A)$ and $A$ is a fix-point automaton: $\text{verification}(A, p, KT) = \text{true}$ means that the property $p$ is satisfied for the model $R$ and for the given initial automaton $A_0$.

With such a technique, the set of secret terms by construction might be larger than the real one when an over-approximation of the intruder knowledge is computed. However, even in this case, if no computed secret term is known by the intruder, then no real secret term is known either. The main asset of this method is that a general pattern of the secret terms has just to be written in the IF specification. All secret terms are then computed for a given tree automaton. This technique is also more precise than the one in [GK00], in the sense that only pertinent secret terms are verified, i.e., secret terms that must not be known by the intruder.

5 Soundness of the Translation of an IF Specification

In Section 3.2, the criteria ensuring the soundness of the verification have been presented. Propositions 3 and 4 show that for an IF specification $\text{Spec}_{IF}$, the tree automaton generated $A_0$ and the TRS $R$ satisfy respectively the conditions [A-1], [A-2], [A-3] and [A-4], and the conditions [R-1] and [R-2]. Consequently, by using an approximation function of $[Ba05]$, the verification soundness is ensured. In this section, we show which kind of result can be propagated on $\text{Spec}_{IF}$ once it has been shown on the abstract model $A_0$ and $R$.

As explained in Section 3, the computation of a tree automaton $A$ is performed from $A_0$ and $R$ using approximation functions as the ones given in $[Ba05]$. This leads either to an over-approximation or to an under-approximation of the knowledge of the intruder.

The main difference between an IF state/transition system and its translation into $A_0$ and $R$ is that, in IF, the application of a rule provides a new state while, in our verification technique, new data are added in the current state. Thus the same rule can be applied anew in our method, whereas it couldn’t be in IF.

Consequently, when we succeed in showing that the resulting model is secure, we can conclude that an IF protocol is secure as well. Moreover, if there exists an attack for the IF protocol specification, then there exists an attack in our abstract model as well.

**Theorem 3.** Let $\text{Spec}_{IF}$ be an IF specification. Let $R$ and $A_0$ be respectively the decorated TRS and the tree automaton generated from $\text{Spec}_{IF}$. Let $p$ be the secrecy property. Let $\alpha$ be an approximation function following $[Ba05]$ such that there exists $N > 0$ for which $R^*(\mathcal{L}(A_0)) \subseteq \mathcal{L}(A_N)$ and $A_N$ is a fix-point automation of the completion of $A_0$ by $R$ and using $\alpha$.

1. If $\text{verification}(A_N, p, R) = \text{true}$ then $\text{Spec}_{IF}$ satisfies the property $p$ as well.
2. If the property is not verified for $\text{Spec}_{IF}$ then $\text{verification}(A_N, p, R) = \text{false}$.

The method presented in this paper is not complete in the sense that, if there exists an attack in our under-approximated model, then this might be a false attack for the IF protocol specification. However, such an attack may be worth detecting. Indeed, even if it can be unrealistic with the verification hypothesis (perfect cryptography and all nonces are different), in the real world, one might be interested in taking this kind of attack into account.
<table>
<thead>
<tr>
<th>Protocol</th>
<th>Computation time (s)</th>
<th>Diagnostic</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSPKL</td>
<td>1.45</td>
<td>SAFE</td>
</tr>
<tr>
<td>NSPK</td>
<td>4.81</td>
<td>FLAWED</td>
</tr>
<tr>
<td>RSA</td>
<td>5.93</td>
<td>FLAWED</td>
</tr>
<tr>
<td>NSK</td>
<td>115.34</td>
<td>SAFE</td>
</tr>
<tr>
<td>Denning-SAco shared key</td>
<td>8.82</td>
<td>SAFE</td>
</tr>
<tr>
<td>Yahalom</td>
<td>97.68</td>
<td>SAFE</td>
</tr>
<tr>
<td>Andrew Secure RPC</td>
<td>23.97</td>
<td>SAFE</td>
</tr>
<tr>
<td>Wide Mouthed Frog</td>
<td>7.20</td>
<td>SAFE</td>
</tr>
<tr>
<td>Kaochow v1</td>
<td>209.60</td>
<td>SAFE</td>
</tr>
<tr>
<td>Kaochow v2</td>
<td>353.91</td>
<td>??</td>
</tr>
<tr>
<td>TMN</td>
<td>8.02</td>
<td>FLAWED</td>
</tr>
<tr>
<td>Neumann</td>
<td>66.45</td>
<td>SAFE</td>
</tr>
<tr>
<td>AAA Mobile IP</td>
<td>754.19</td>
<td>SAFE</td>
</tr>
<tr>
<td>UMT-AKA</td>
<td>0.55</td>
<td>SAFE</td>
</tr>
<tr>
<td>CHAPv2</td>
<td>16.46</td>
<td>SAFE</td>
</tr>
<tr>
<td>CRAM-MD5</td>
<td>0.37</td>
<td>SAFE</td>
</tr>
<tr>
<td>DHCP-Delayed-Auth</td>
<td>6.84</td>
<td>SAFE</td>
</tr>
<tr>
<td>EKE</td>
<td>2.87</td>
<td>SAFE</td>
</tr>
<tr>
<td>LPD-IMSR</td>
<td>3.25</td>
<td>SAFE</td>
</tr>
<tr>
<td>LPD-MSR</td>
<td>0.61</td>
<td>FLAWED</td>
</tr>
<tr>
<td>TSIG</td>
<td>4.46</td>
<td>SAFE</td>
</tr>
<tr>
<td>SHARE</td>
<td>1.90</td>
<td>SAFE</td>
</tr>
</tbody>
</table>

Table 3. Experimentations on some secrecy properties using TA4SP

6 Experimentations and Future works

With the contributions presented in this paper, the automatic TA4SP [Ba05] tool has been plugged to the high level protocol specification languages HLPSL and PROUVE. Consequently, 23 protocols, specified either in HLPSL or in PROUVE, have been successfully verified. These experimentations are listed in Table 3.

Some of them have been diagnosed as flawed and they have indeed been shown as attacked with other approaches (NSPK, TMN, RSA and LPD-MSR). Others have been shown secure: NSPKL, SHARE, TSIG, LPD-IMSR, EKE, DHCP-Delayed-Auth, CRAM-MD5, CHAPv2, AAA Mobile IP,....

For the protocol Kaochow v2, an over-approximation leads to an inconclusive result and no under-approximation large enough can be computed. This is the only inconclusive result we have obtained to date.

Following theoretical works [Ba03b, Ca00] showing the relation between different models, making verification techniques available from high level protocol specification languages has come as an evidence for the last ten years. Already in [Ba99], the authors have plugged the NRLPA analyser [Men94] into the high level protocol specification language CAPSL [DM99]. G. Lowe has defined his own-language CASPER [Low98] for his verification tool FDR [Low96].
in order to concern a larger community of users. Much more recently, in the European project AVISPA and in the French project PROUVE, the verification tools CL-AtSe [Tur06], SATMC [AC02], OFMC [Ba03a], Hermes [Ba03c] have been all connected either to the HLPSL or PROUVE languages.

In this paper, we have proposed a way to offer an expert verification technique to non expert users. The contributions of this paper allow the remove of the usual drawbacks encountered with such a verification technique. Indeed, specifying a protocol by a TRS and a tree automaton is a very error-prone task, and much expertise is needed for debugging a flawed specification. For an IF specification, the abstract model resulting from the translation is a sound abstraction of it in the sense that if the abstract model will stand an intruder, then the IF specification will stand it as well. When an intruder can attack the IF protocol, the abstract protocol is flawed as well.

In [Ba05], we have presented the TA4SP tool as a tool plugged into the IF. We have then presented in this paper, how TA4SP is connected to IF and therefore, what this tool can conclude for a given IF specification. Thus, the TA4SP tool has been successfully run against 23 protocols specified either in PROUVE or in HLPSL.

This development falls into the movement which aims make some state-of-the-art technologies reachable for a larger community of users, in particular protocol specifiers. Following some recent results [Ba06a], we would like to make TA4SP handle much more algebraic properties and then to provide an automatic tool with a larger scope of verification. Another proposition is to extend TA4SP with a technique of attack reconstruction detailed in [BG06] in order to help a protocol specifier to discriminate false attacks from the true ones.

References


RR  n° 6039
[Ba06b] Y. Boichut and al. Validation of PROUVÉ protocols using the automatic tool TA4SP. to be attributed, INRIA-Lorraine, Loria, 2006.