Handling Algebraic Properties in Automatic Analysis of Security Protocols

Yohan Boichut, Pierre-Cyrille Héam, Olga Kouchnarenko

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Handling Algebraic Properties in Automatic Analysis of Security Protocols

Yohan Boichut

— Pierre-Cyrille Héam — Olga Kouchnarenko

N° 5857

Mars 2006

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--5857--FR+ENG

Handling Algebraic Properties in Automatic Analysis of Security Protocols

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

Thème COM — Systèmes communicants
Projets CASSIS

Rapport de recherche n° 5857 — Mars 2006 — 18 pages

Abstract: This paper extends the approximation-based theoretical framework in which the security problem – secrecy preservation against an intruder – may be semi-decided through a reachability verification.

We explain how to cope with algebraic properties for an automatic approximation-based analysis of security protocols. We prove that if the initial knowledge of the intruder is a regular tree language, then the security problem may by semi-decided for protocols using cryptographic primitives with algebraic properties. More precisely, an automatically generated approximation function enables us 1) an automatic normalization of transitions, and 2) an automatic completion procedure. The main advantage of our approach is that the approximation function makes it possible to verify security protocols with an arbitrary number of sessions.

The concepts are illustrated on an example of the *view-only* protocol using a cryptographic primitive with the exclusive or algebraic property.

Key-words: Verification, security protocols, approximations

* Laboratoire d'Informatique de l'Université de Franche Comté FRE 2661 CNRS, projet INRIA-CASSIS. This work has been supported by the European project AVISPA IST-2001-39252 and the French projects RNTL PROUVE and ACI SATIN.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,

Analyse automatique de protocoles de sécurité utilisant des propriétés algébriques

Résumé : Ce papier étend le travail basé sur des approximations permettant de semi-décider si un intrus a accès à un secret dans un protocole de sécurité.

Nous montrons comment prendre en compte les propriétés algébriques pour une analyse automatique par approximation de protocoles de sécurité. Nous prouvons que si la connaissance intiale de l'intrus est un langage régulier d'arbres, alors le problème peut être semi-décider pour des protocoles utilisant des primitives cryptographique ayant des propriétés algébriques. Plus précisément, une fonction d'approximation générée automatiquement nous permet 1) une normalisation automatique des transitions et 2) une procedure de complétion automatique. Le principal avantage de notre approche est de permettre la vérification de protocoles pour un nombre arbitraire de sessions.

Les concepts sont illustrés sur l'exemple du protocole *view-only* qui utilise les propriétés algébriques du ou-exclusif.

Mots-clés : Vérification, protocoles de sécurité, approximations

1 Introduction

Cryptographic protocols are widely used to secure the exchange of information over open modern networks. It is now widely accepted that formal analysis can provide the level of assurance required by both the developers and the users of the protocols. However, whatever formal model one uses, analyzing cryptographic protocols is a complex task because the set of configurations to consider is very large, and can even be infinite. Indeed, any number of sessions (sequential or parallel executions) of protocols, sessions interleaving, any size of messages, algebraic properties of encryption or data structures give rise to infinite-state systems.

Our main objective is to automate in so far as possible the analysis of protocols. More precisely, we are interested in a fully automatic method to (semi)decide the *security* problem. In the context of the verification of protocols, the security problem consists of deciding whether a protocol preserves secrecy against an intruder, or not.

For this problem, current verification methods based on model-checking can be applied whenever the number of participants and the number of sessions between the agents are bounded. In this case, the protocol security problem is co-NP-complete [21]. The recent work [22] presents new decidability results for a bounded number of sessions, when the initial knowledge of the intruder is a regular language under the assumption that the keys used in protocols are atomic.

When the number of sessions is unbounded, the security problem of cryptographic protocols becomes undecidable, even when the length of the messages is bounded [15]. Decidability can be recovered by adding restrictions as in [13] where tree automata with one memory are applied to decide secrecy for cryptographic protocols with single blind copying.

Another way to circumvent the problem is to employ abstraction-based approximation methods [19,17]. In fact, these methods use 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, the security problem may be semi-decided through a reachability verification. The finite tree automata permit to ensure that some states are unreachable, and hence that the intruder will never be able to know certain terms.

To achieve the goal of an automatic analysis of protocols, we have investigated, improved [7] and extended [6] the semi-algorithmic method by Genet and Klay. The main advantage of our approach is that the automatically generated symbolic approximation function makes it possible to automatically verify security protocols while considering an unbounded number of sessions. The efficiency and usefulness of our approach have been confirmed by the tool `TA4SP` (Tree Automata based on Automatic Approximations for the Analysis of Security Protocols), which has already been used for analyzing many real Internet security protocols as exemplified in the European Union project AVISPA [1] [2].

However, for some cryptographic protocols, the secrecy is not preserved even when used with strong encryption algorithms. The purpose of the work we present in this paper is to extend the symbolic approximation-based theoretical framework defined in [6] to security

[1] http://www.avispa-project.org/

protocols using cryptographic primitives with algebraic properties. To be more precise, the goal is to relax the perfect cryptography assumption in our symbolic approximation-based method. As explained in [14], such an assumption is too strong in general since some attacks are built using the interaction between protocol rules and properties of cryptographic operators.

The main contribution of this paper consists of showing the feasibility of the automatic analysis of protocols where the number of sessions is unbounded and some algebraic properties of the cryptographic primitives – e.g. the exclusive or – are taken into account.

The main result of the paper is that the automatically generated approximation function allows us to over-approximate the knowledge of the intruder. `TA4SP` we have been implementing is used for obtaining experimental results. The most important new feature is the exclusive or algebraic property, `XOR` for short, modulo which the protocol analysis is performed. The feasibility of our approach is illustrated on the example of the *view-only* protocol.

Related Work In [20] it has been shown that using equational tree automata under associativity and/or commutativity is relevant for security problems of cryptographic protocols with an equational property. For protocols modeled by associative-commutative TRSs, the authors announce the possibility for the analysis to be done automatically thanks to the tool ACTAS manipulating associative-commutative tree automata and using approximation algorithmes. However, the engine has still room to be modified and optimized to support an automated verification.

In [23], the authors investigate algebraic properties and timestamps in the approximation-based protocol analysis. Like in [6], there is no left-linearity condition on TRSs modeling protocols. However, the weakness of the work is that no tool is mentioned in [23].

In the recent survey [14], the authors give an overview of the existing methods in formal approaches to analyze cryptographic protocols. In the same work, a list of some relevant algebraic properties of cryptographic operators is established, and for each of them, the authors provide examples of protocols or attacks using these properties.

This survey lists two drawbacks with the recent results aiming at the analysis of protocols with algebraic properties. First, in most of the papers a particular decision procedure is proposed for a particular property. Second, the authors emphasize the fact that the results remain theoretical, and very few implementations automatically verify protocols with algebraic properties.

Following the result presented in [10], the authors have prototyped a new feature to handle the `XOR` operator in CL-AtSe (Constraint Logic based Attack Searcher), one of the four official tools of the `AVISPA` tool-set [2].

Layout of the paper The paper is organized as follows. After giving preliminary notions on tree-automata and term rewriting systems (TRSs), we introduce in Section 2 a substitution notion depending on rules of a TRS, and a notion of compatibility between that substitutions and finite tree-automata, both suitable for our work. Section 3 presents the completion theorem making the completion procedure stop even when protocols – modeled by non left-linear TRSs – use cryptographic primitives with algebraic properties. The

main result is then a consequence of the completion theorem allowing us to use an approximation function to obtain an over-approximation of the knowledge of the intruder. In Section 4, we explain how to apply the main theorem to verify the *view-only* protocol using a cryptographic primitive with the exclusive or algebraic property.

2 Background and notation

In this section basic notions on finite tree automata, term rewriting systems and approximations are reminded. The reader is refereed to [12] for more detail.

2.1 Notations

For any set A, we denote by 2^A the set of subsets of A. We denote by $\mathbb{N}$ the set of natural integers and $\mathbb{N}^*$ denotes the finite strings over $\mathbb{N}$.

Let $\mathcal{F}$ be a finite set of symbols with their arities. The set of symbols of $\mathcal{F}$ of arity i is denoted $\mathcal{F}_i$. Let $\mathcal{X}$ be a finite set whose elements are variables. We assume that $\mathcal{X} \cap \mathcal{F} = \emptyset$.

A finite ordered tree t over a set of labels $(\mathcal{F}, \mathcal{X})$ is a function from a prefix-closed set $\mathcal{P}\mathrm{os}(t) \subseteq \mathbb{N}^*$ to $\mathcal{F} \cup \mathcal{X}$. A term t over $\mathcal{F} \cup \mathcal{X}$ is a labeled tree whose domain $\mathcal{P}\mathrm{os}(t)$ satisfies the following properties:

- $\mathcal{P}\mathrm{os}(t)$ is non-empty and prefix closed,
- For each $p \in \mathcal{P}\mathrm{os}(t)$, if $t(p) \in \mathcal{F}_n$, then $\{i \mid p.i \in \mathcal{P}\mathrm{os}(t)\} = \{1, \ldots, n\}$,
- For each $p \in \mathcal{P}\mathrm{os}(t)$, if $t(p) \in \mathcal{X}$, then $\{i \mid p.i \in \mathcal{P}\mathrm{os}(t)\} = \emptyset$.

Each element of $\mathcal{P}\mathrm{os}(t)$ is called a position of t. For each subset $\mathcal{K}$ of $\mathcal{X} \cup \mathcal{F}$ and each term t we denote by $\mathcal{P}\mathrm{os}_{\mathcal{K}}(t)$ the subset of positions p's of t such that $t(p) \in \mathcal{K}$. Each position p of t such that $t(p) \in \mathcal{F}$, is called a functional position. The set of terms over $(\mathcal{F}, \mathcal{X})$ is denoted $\mathcal{T}(\mathcal{F}, \mathcal{X})$. A ground term is a term t such that $\mathcal{P}\mathrm{os}(t) = \mathcal{P}\mathrm{os}_{\mathcal{F}}(t)$ (i.e. such that $\mathcal{P}\mathrm{os}_{\mathcal{X}}(t) = \emptyset$). The set of ground terms is denoted $\mathcal{T}(\mathcal{F})$.

A subterm $t_{|p}$ of $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ at position p is defined by the following:

- $\mathcal{P}\mathrm{os}(t_{|p}) = \{i \mid p.i \in \mathcal{P}\mathrm{os}(t)\}$,
- For all $j \in \mathcal{P}\mathrm{os}(t_{|p})$, $t_{|p}(j) = t(p.j)$.

We denote by $t[s]_p$ the term obtained by replacing in t the subterm $t_{|p}$ by s.

If $\mathcal{X}$ contains n elements and is (arbitrarily) ordered, then a context C is an element of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ in which each element of $\mathcal{X}$ occurs exactly once. The expression $C[t_1, \ldots, t_n]$ for $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ denotes the term in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ obtained from C by replacing the i-th element of $\mathcal{X}$ by t_i for each $1 \leq i \leq n$.

For all sets A and B, we denote by $\Sigma(A, B)$ the set of functions from A to B. If $\sigma \in \Sigma(\mathcal{X}, B)$, then for each term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, we denote by $t\sigma$ the term obtained from t by replacing for each $x \in \mathcal{X}$, the variable x by $\sigma(x)$.

A term rewriting system (TRS for short) over $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is a finite set of pairs (l, r) from $\mathcal{T}(\mathcal{F}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}, \mathcal{X})$, denoted $l \to r$, such that the set of variables occurring in r is included

in the set of variables of l. A term rewriting system is left-linear if for each rule $l \to r$, every variable occurring in l occurs at most once. For each ground term t, we denote by $\mathcal{R}(t)$ the set of ground terms t' such that there exist a rule $l \to r$ of $\mathcal{R}$, a function $\mu \in \Sigma(\mathcal{X}, \mathcal{T}(\mathcal{F}))$ and a position p of t satisfying $t_{|}p = l\mu$ and $t' = t[r\mu]_p$. The relation $\{(t, t') \mid t' \in \mathcal{R}(t)\}$ is classicaly denoted $\to_{\mathcal{R}}$. For each set of ground terms B we denote by $\mathcal{R}^*(B)$ the set of ground terms related to an element of B modulo the reflexive-transitive closure of $\to_{\mathcal{R}}$.

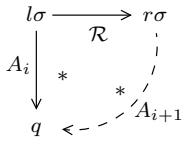
A tree automaton $\mathcal{A}$ is a tuple $(\mathcal{Q}, \Delta, F)$, where $\mathcal{Q}$ is the set of states, Δ the set of transitions, and F the set of final states. Transitions are rewriting rules of the form $f(q_1, \ldots, q_k) \to q$, where $f \in \mathcal{F}_k$ and the q_i's are in $\mathcal{Q}$. A term $t \in \mathcal{T}(\mathcal{F})$ is accepted or recognized by $\mathcal{A}$ if there exists $q \in F$ such that $t \to^*_\Delta q$ (we also write $t \to^*_{\mathcal{A}} q$). The set of terms accepted by $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A})$. For each state $q \in \mathcal{Q}$, we write $\mathcal{L}(\mathcal{A}, q)$ for the tree language $\mathcal{L}((\mathcal{Q}, \Delta, \{q\}))$. A tree automaton is finite if its set of transitions is finite.

2.2 Approximations to Handle Algebraic Properties

This section recalls the approximation-based framework we have been developing, and explains our objectives from a formal point of view.

Given a tree automaton $\mathcal{A}$ and a TRS $\mathcal{R}$ (for several classes of automata and TRSs), the tree automata completion [17,16] algorithm 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 [16]), and such that $\mathcal{L}(\mathcal{A}_k) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ otherwise.

The tree automata completion works as follows. From $\mathcal{A} = \mathcal{A}_0$ 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 $s \to_{\mathcal{R}} t$ then $t \in \mathcal{L}(\mathcal{A}_{i+1})$. If we find a fixpoint 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 one class of [16]). To build $\mathcal{A}_{i+1}$ from $\mathcal{A}_i$, we achieve a *completion step* which consists of finding *critical pairs* between $\to_{\mathcal{R}}$ and $\to_{\mathcal{A}_i}$. For a substitution $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ and a rule $l \to r \in \mathcal{R}$, a critical pair is an instance $l\sigma$ of l such that there exists $q \in \mathcal{Q}$ satisfying $l\sigma \to^*_{\mathcal{A}_i} q$ and $r\sigma \not\to^*_{\mathcal{A}_i} q$. For every critical pair $l\sigma \to^*_{\mathcal{A}_i} q$ and $r\sigma \not\to^*_{\mathcal{A}_i} q$ 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. $r\sigma \to_{\mathcal{A}_{i+1}} q$.



However, the transition $r\sigma \to q$ is not necessarily a normalized transition of the form $f(q_1, \ldots, q_n) \to q'$ and so has to be normalized first. For example, to normalize a transition of the form $f(g(a), h(q')) \to q$, we need to find some states q_1, q_2, q_3 and replace the previous transition by a set of normalized transitions: $\{a \to q_1, g(q_1) \to q_2, h(q') \to q_3, f(q_2, q_3) \to q\}$.

Assume that q_1, q_2, q_3 are new states, then adding the transition itself or its normalized form does not make any difference. Now, assume that $q_1 = q_2$, the normalized form becomes $\{a \to q_1, g(q_1) \to q_1, h(q') \to q_3, f(q_1, q_3) \to q\}$. This set of normalized transitions represents

the regular set of non normalized transitions of the form $f(g^\star(a), h(q')) \to q$; which contains the transition initially we wanted to add amongst many others. Hence, this is an over-approximation. We could have made an even more drastic approximation by identifying q_1, q_2, q_3 with q, for instance.

The above method does not work for all TRSs. For instance, consider the tree automaton $\mathcal{A} = (\{q_1, q_2, q_f\}, \{A \to q_1, A \to q_2, f(q_1, q_2) \to q_f\}, \{q_f\})$ and the TRS $\mathcal{R} = \{f(x, x) \to g(x)\}$. There is no substitution σ such that $l\sigma \to^*_{\mathcal{A}} q$, for a q in $\{q_1, q_2, q_f\}$. Thus, following the procedure, there is no transition to add. But $f(A, A) \in \mathcal{L}(\mathcal{A})$. Thus $g(A) \in \mathcal{R}(L(\mathcal{A}))$. Since $g(A) \notin \mathcal{L}(\mathcal{A})$, the procedure stops (in fact does not begin) before providing an over-approximation of $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$.

The TRSs used in the security protocol context are often non left-linear. Indeed, there are a lot of protocols that cannot be modeled by left-linear TRSs. Unfortunately, to be sound, the approximation-based analysis described in [17] requires the use of left-linear TRSs. Nevertheless, this method can still be applied to some non left-linear TRSs, which satisfy som weaker conditions. In [16] the authors propose new linearity conditions. However, these new conditions are not well-adapted to be automatically checked.

In our previous work [6] we explain how to define a criterion on $\mathcal{R}$ and $\mathcal{A}$ to make the procedure automatically work for industrial protocols analysis. This criterion ensures the soundness of the method described in [17,16].

However, to handle protocols the approach in [6] is based on a kind of constant typing. In this paper we go further and propose a procedure supporting a fully automatic analysis and handling – without typing – algebraic properties like `XOR` presented in Fig. 1.

```
xor(x,y) ⟶ xor(y,x)                    I.
xor(xor(x,y),z) ⟶xor(x,xor(y,z))       II.
xor(x,0) ⟶ x                          III.
xor(x,x) ⟶0                           IV.
```

Fig. 1. `XOR` properties

Let us remark first that the criterion defined in [16] does not allow managing the above rule IV. Second, in [6] we have to restrict `XOR` operations to typed terms to deal with the rule IV.

However, some protocols are known to be flawed by type confusing attacks [14,8,11]. In order to cope with these protocols, a new kind of substitution is defined in Section 2.3, and a new left-linear like criterion is introduced in Section 3.

Notice that following and extending [6], our approach can be applied for any kinds of TRSs. Moreover, it can cope with exponentiation algebraic properties and this way analyse Diffie-Hellman based protocols.

2.3 $(l \to r)$-substitutions

In this technical subsection, we define the notion of a $(l \to r)$-substitution suitable for the present work.

Definition 1. *Let $\mathcal{R}$ be a term rewriting system and $l \to r \in \mathcal{R}$. A $(l \to r)$-substitution is an application from $\mathcal{P}\mathrm{os}_{\mathcal{X}}(l)$ into $\mathcal{Q}$.*

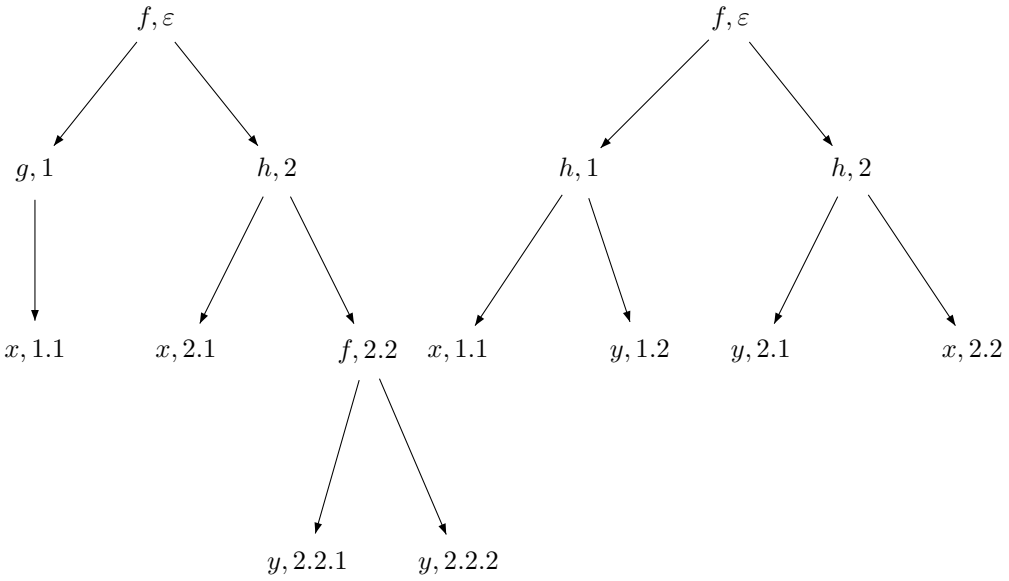
Let $l \to r \in \mathcal{R}$ and σ be a $(l \to r)$-substitution. We denote by $l\sigma$ the term of $\mathcal{T}(\mathcal{F}, \mathcal{Q})$ defined as follows:

- $\mathcal{P}\mathrm{os}(l\sigma) = \mathcal{P}\mathrm{os}(l)$,
- for each $p \in \mathcal{P}\mathrm{os}(l)$, if $p \in \mathcal{P}\mathrm{os}_{\mathcal{X}}(l)$ then $l\sigma(p) = \sigma(l(p))$, otherwise $l\sigma(p) = l(p)$.

Similarly, we denote by $r\sigma$ the term of $\mathcal{T}(\mathcal{F}, \mathcal{Q})$ defined by:

- $\mathcal{P}\mathrm{os}(r\sigma) = \mathcal{P}\mathrm{os}(r)$,
- for each $p \in \mathcal{P}\mathrm{os}(r)$, if $p \notin \mathcal{P}\mathrm{os}_{\mathcal{X}}(r)$ then $r\sigma(p) = r(p)$ and $r\sigma(p) = \sigma(l(p'))$ otherwise, where $p' = \min \mathcal{P}\mathrm{os}_{r(p)}(l)$ (positions are lexicographically ordered).

Example 1 *Let us consider $l = f(g(x), h(x, f(y, y)))$ and $r = f(h(x, y), h(y, x))$ represented by the following trees (elements after the comma are the positions in the term; l is represented on the left and r on the right):*



Functional positions of l *are* 1.1 *and* 2.1 *for* x*, and* $2.2.1$ *and* $2.2.2$ *for* y*. Let* $\sigma(1.1) = q_1$*,* $\sigma(2.1) = q_2$*,* $\sigma(2.2.1) = q_3$ *and* $\sigma(2.2.2) = q_4$*;* σ *is a* $(l \to r)$*-substitution and*

$$l\sigma = f(g(q_1), h(q_2, f(q_3, q_4)))$$

is the term obtained from l *by substituting the variable in position* p *by* $\sigma(p)$*. Now we explain how to compute* $r\sigma$*. The minimal position where* x *[resp.* y*] occurs in* l *is* 1.1 *[resp.* $2.2.1$*]. Thus* $r\sigma$ *is obtained from* r *by substituting all* x*'s in* r *by* $\sigma(1.1) = q_1$ *and all* y*'s by* $\sigma(2.2.1) = q_3$*. Thus*

$$r\sigma = f(h(q_1, q_3), h(q_3, q_1)).$$

As mentioned in Section 2.2, the completion procedure does not work for all tree automata and TRSs. That is why we introduce the notion of compatibility between finite tree-automata and $(l \to r)$-substitutions. Notice that the condition required below is weaker than the conditions in [16]. Moreover, it is more general and can be applied to a larger class of applications.

Definition 2. *Let* $\mathcal{A}$ *be a finite tree automaton. We say that a* $(l \to r)$*-substitution* σ *is* $\mathcal{A}$*-compatible if for each* $x \in \mathcal{V}\mathrm{ar}(l)$*,*

$$\bigcap_{p \in \mathcal{P}\mathrm{os}_{\{x\}}(l)} \mathcal{L}(\mathcal{A}, \sigma(p)) \neq \emptyset.$$

Example 2 *Let* $\mathcal{A}_{\text{exe}} = (\{q_0, q_f\}, \Delta_{\text{exe}}, \{q_f\})$ *with the set of transitions* $\Delta_{\text{exe}} = \{A \to q_0, A \to q_f, f(q_f, q_0) \to q_f, h(q_0, q_0) \to q_0\}$. *Let* $\mathcal{R}_{\text{exe}} = \{f(x, h(x, y)) \to h(A, x)\}$. *The automaton* $\mathcal{A}_{\text{exe}}$ *recognizes the set of trees such that every path from the root to a leaf is of the form* f^*h^*A. *Let us consider the substitution* σ_{exe} *defined by* $\sigma_{\text{exe}}(1) = q_f$, $\sigma_{\text{exe}}(2.1) = q_0$ *and* $\sigma_{\text{exe}}(2.2) = q_0$. *The tree* $t = A \to q_f$ *belongs to* $\mathcal{L}(\mathcal{A}, \sigma_{\text{exe}}(1))$. *Furthermore* $t = A \to q_0$, *so* $t \in \mathcal{L}(\mathcal{A}, \sigma_{\text{exe}}(2.2))$. *Therefore* σ_{exe} *is* $\mathcal{A}$*-compatible.*

3 Approximations for non-left Linear TRSs

In this section $\mathcal{R}$ denotes a fixed term rewriting system and $\mathcal{Q}$ an infinite set of states. We first introduce the notion of normalization associated with $(l \to r)$-substitutions. Secondly, we give the main result – consequence of the completion theorem – allowing us to over-approximate the descendants of regular sets.

3.1 Normalizations

The notion of normalization is common. The definitions below are simply adapted to our notion of $(l \to r)$-substitutions.

Definition 3. *Let* $\mathcal{A}$ *be a finite tree automaton. An approximation function (for* $\mathcal{A}$*) is a function which associates to each tuple* $(l \to r, \sigma, q)$, *where* $l \to r \in \mathcal{R}$, σ *is an* $\mathcal{A}$*-compatible* $(l \to r)$*-substitution and* q *a state of* $\mathcal{A}$, *a function from* $\mathcal{P}\text{os}(r)$ *to* $\mathcal{Q}$.

Example 3 *Consider the automaton* $\mathcal{A}_{\text{exe}}$, *the term rewriting system* $\mathcal{R}_{\text{exe}}$ *and the substitution* σ_{exe} *defined in Example 2. For* σ_{exe}, *an approximation function* γ_{exe} *may be defined by*

$$\gamma_{\text{exe}}(l \to r, \sigma_{\text{exe}}, q_f) : \begin{cases} \varepsilon \mapsto & q_1 \\ 1 \mapsto & q_0 \\ 2 \mapsto & q_1 \end{cases}.$$

To totally define γ_{exe}, *the others (finitely many)* $\mathcal{A}_{\text{exe}}$*-compatible substitutions should be considered too.*

The notion of normalization below is classical. The definition takes our notion of $(l \to r)$-substitutions into account only.

Definition 4. *Let* $\mathcal{A} = (\mathcal{Q}_0, \Delta, F_0)$ *be a finite tree automaton,* γ *an approximation function for* $\mathcal{A}$, $l \to r \in \mathcal{R}$, σ *an* $\mathcal{A}$*-compatible* $(l \to r)$*-substitution, and* q *a state of* $\mathcal{A}$. *We denote by* $\text{Norm}_\gamma(l \to r, \sigma, q)$ *the following set of transitions, called* normalization *of* $(l \to r, \sigma, q)$*:*

$$\{f(q_1,\ldots,q_k) \to q' \mid p \in \mathcal{P}\mathrm{os}_{\mathcal{F}}(r),\ t(p) = f,$$
$$q' = q \text{ if } p = \varepsilon \text{ otherwise } q' = \gamma(l \to r, \sigma, q)(p)$$
$$q_i = \gamma(l \to r, \sigma, q)(p.i) \text{ if } p.i \notin \mathcal{P}\mathrm{os}_{\mathcal{X}}(r),$$
$$q_i = \sigma(\min\{p' \in \mathcal{P}\mathrm{os}_{\mathcal{X}}(l) \mid l(p') = r(p.i)\}) \text{otherwise}$$
$$\}$$

The min *is computed for the lexical order.*

Notice that the set $\{p' \in \mathcal{P}\mathrm{os}_{\mathcal{X}}(l) \mid l(p') = r(p.i)\}$ used in the above definition is not empty. Indeed, in a term rewriting system variables occurring in the right-hand side must, by definition, occur in the left-hand side too.

Example 4 *Following Example 3, ε is the unique functional position of $r = h(x,y)$. Consequently, we set q' of the definition to be equal to q_f. Thus $\mathrm{Norm}_{\gamma_{\mathrm{exe}}}(l \to r, \sigma_{\mathrm{exe}}, q_f)$ is of the form $\{A \to q?, h(q?, q??) \to q_f\}$. Since for r, the position 1 is a functionnal position and 2 is in $\mathcal{P}\mathrm{os}_{\mathcal{X}}(r)$, we use the last line of the definition to compute $q??$ and $q?$ is defined by the approximation function γ_{exe}. Finally we obtain:*

$$\begin{aligned}\mathrm{Norm}_{\gamma_{\mathrm{exe}}}(l \to r, \sigma_{\mathrm{exe}}, q_f) &= \{r(1) \to \gamma_{\mathrm{exe}}(1), r(\varepsilon)(\gamma_{\mathrm{exe}}(1), \sigma_{\mathrm{exe}}(1)) \to q_0\} \\ &= \{A \to q_0, h(q_0, q_f) \to q_f\}.\end{aligned}$$

Lemma 1. *Let $\mathcal{A} = (\mathcal{Q}_0, \Delta, F_0)$ be a finite tree automaton, γ an approximation function, $l \to r \in \mathcal{R}$, σ an $\mathcal{A}$-compatible $(l \to r)$-substitution, and q a state of $\mathcal{A}$. If $l\sigma \to^*_{\mathcal{A}_0} q$ then*

$$r\sigma \to^*_{\mathrm{Norm}_\gamma(l \to r, \sigma, q)} q.$$

Proof is obvious. The transitions in Norm_γ are precisely added to reduce $r\sigma$ to q.

3.2 Completions

This section is dedicated to the proof of the main result: how to build a regular over-approximation of $\mathcal{R}^*(\mathcal{A})$? The above lemma shows how to over-approximate one rewriting step.

Lemma 2. *Let $\mathcal{A}_0 = (\mathcal{Q}_0, \Delta_0, F_0)$ be a finite tree automaton and γ an approximation function for $\mathcal{A}_0$. The automaton $\mathcal{C}_\gamma(\mathcal{A}_0) = (\mathcal{Q}_1, \Delta_1, F_1)$ is defined by:*

$$\Delta_1 = \bigcup \mathrm{Norm}_\gamma(l \to r, \sigma, q)$$

where the union involves all rules $l \to r \in \mathcal{R}$, all states $q \in \mathcal{Q}_0$, all $\mathcal{A}_0$-compatible $(l \to r)$-substitutions σ such that $l\sigma \to^*_{\mathcal{A}_0} q$ and $r\sigma \not\to^*_{\mathcal{A}_0} q$,

$$F_1 = F_0,$$

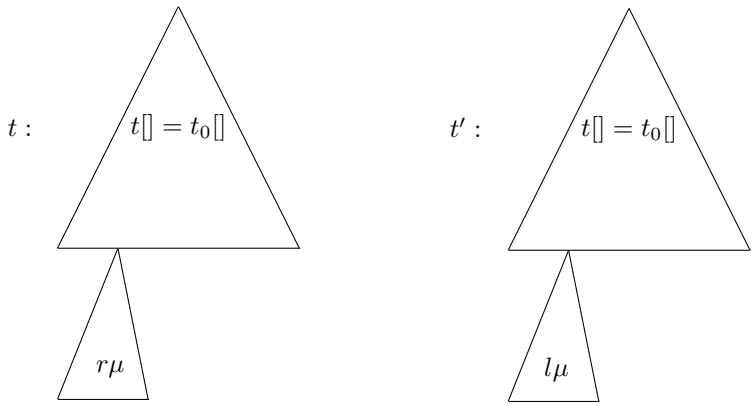
$$\mathcal{Q}_1 = \mathcal{Q}_0 \cup \mathcal{Q}_1,$$

where $\mathcal{Q}_1$ denotes the set of states occurring in left or right-hand sides of transitions of Δ_1. One has

$$\mathcal{L}(\mathcal{A}_0) \cup \mathcal{R}(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{C}_\gamma(\mathcal{A}_0)).$$

Proof. Let $t \in \mathcal{L}(\mathcal{A}_0) \cup \mathcal{R}(\mathcal{L}(\mathcal{A}_0))$. By definition of $\mathcal{C}_\gamma(\mathcal{A}_0)$ one has $\mathcal{L}(\mathcal{A}_0) \subseteq \mathcal{L}(\mathcal{C}_\gamma(\mathcal{A}_0))$. Consequently, if $t \in \mathcal{L}(\mathcal{A}_0)$, one has $t \in \mathcal{L}(\mathcal{C}_\gamma(\mathcal{A}_0))$. Thus we may now assume that $t \in \mathcal{R}(\mathcal{L}(\mathcal{A}_0))$. Thus there exists a rule $l \to r \in \mathcal{R}$ a term t_0 in $\mathcal{L}(\mathcal{A}_0)$, a position p of t_0 and a substitution μ in $\Sigma(\mathcal{X}, \mathcal{T}(\mathcal{F}))$ such that

$$t_{0|p} = l\mu \quad \text{and} \quad t = t_0[r\mu]_p. \tag{1}$$



Since $t_0 \in \mathcal{L}(\mathcal{A}_0)$, there exists a state $q \in \mathcal{Q}_0$ and a state $q_f \in F_0$ such that

$$l\mu \to^*_{\mathcal{A}_0} q \quad \text{and} \quad t_0[q]_p \to^*_{\mathcal{A}_0} q_f. \tag{2}$$

Since $l\mu \to^*_{\mathcal{A}_0} q$ there exists an $(l \to r)$-substitution σ such that $l\mu \to_{\mathcal{A}_0} l\sigma$. Furthermore, for each $x \in \mathcal{V}\mathrm{ar}(l)$,

$$\mu(x) \in \bigcap_{p \in \mathcal{P}\mathrm{os}_{\{x\}}(l)} \mathcal{L}(\mathcal{A}, \sigma(p)),$$

thus the $(l \to r)$-substitution σ is $\mathcal{A}_0$ compatible. Therefore, using Lemma 1, one has

$$r\sigma \to^*_{\mathcal{C}_\gamma(\mathcal{A}_0)} q. \tag{3}$$

For each variable x occurring in l and all positions p of x in l one has $\mu(x) \to^*_{\mathcal{A}_0} \sigma(p)$. In particular, for each variable x occurring in l, $\mu(x) \to^*_{\mathcal{A}_0} \sigma(p')$, where p' is the minimal position where x occurs in l. Consequently and by definition of $r\sigma$, one has

$$r\mu \to^*_{\mathcal{A}_0} r\sigma. \tag{4}$$

We are now able to conclude.

$$\begin{array}{lll} t = & t_0[r\mu] & \text{using (1)} \\ \rightarrow^*_{\mathcal{A}_0} & t_0[r\sigma] & \text{using (4)} \\ \rightarrow^*_{\mathcal{C}_\gamma(\mathcal{A}_0)} & t_0[q] & \text{using (3)} \\ \rightarrow^*_{\mathcal{A}_0} & q_f & \text{using (2)} \end{array}$$

Thus $t \in \mathcal{L}(\mathcal{C}_\gamma(\mathcal{A}_0))$, proving the theorem.

Let us remark that using well chosen approximation functions may iteratively lead to a fixpoint automaton which recognizes an over-approximation of $\mathcal{R}^*(\mathcal{A}_0)$. One can formally express this by the following main theorem.

Theorem 5 *Let $(\mathcal{A}_n)$ and (γ_n) be respectively a sequence a finite tree automata and a sequence of approximation functions defined by: for each integer n, γ_n is an approximation function for $\mathcal{A}_n$ and*

$$\mathcal{A}_{n+1} = \mathcal{C}_{\gamma_n}(\mathcal{A}_n).$$

If the sequence $(\mathcal{A}_n)$ is ultimately constant equal to $\mathcal{B}$, then

$$\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{B}).$$

The proof is immediate by a simple induction using Lemma 2. Notice that in [6], we have defined a family of approximation functions which can be automatically generated. Another advantage is that they can be easily adapted to the present construction as explained in the next section.

4 Application to the *View-Only* Protocol

In this section we illustrate the main concepts on the example of the *view-only* protocol, and we explain how to manipulate the tool `TA4SP` supporting an automated verification.

4.1 Verifying the *View-Only* Protocol

The *View-Only* protocol (Fig. 2) is a component of the *Smartright* system [1]. In the context of home digital network, this system prevents users from unlawfully copying movies broadcast on the network. The *view-only* participants are a decoder (`DC`) and a digital TV set (`TVS`). They share a secret key `Kab` securely sealed in both of them. The goal of this protocol is to periodically change a secret control word (`CW`) enabling to decode the current broadcast program. As seen in Fig. 2, the properties of the `XOR` operator allow to establish the sharing of `CW` between the participants.

The data `VoKey`, `VoR` and `VoRi` are randomly generated numbers. The functional symbol `h` represents a one-way function, meaning that no-one can guess `x` from `h(x)`, unless he already knows `x`.

Let us explain how this protocol works.

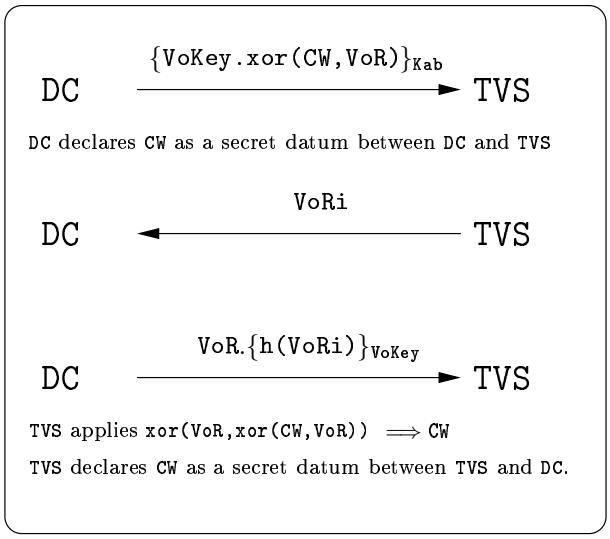


Fig. 2. The *"view-only"* Protocol

- **Step 1:** DC sends a message containing xor(CW,VoR) and VoKey to TVS. This message is encoded by the private shared key Kab. The data VoKey is a fresh symmetric key used along the session. At this stage, TVS can extract neither CW nor VoR from xor(CW,VoR) since TVS knows neither CW nor VoR.
- **Step 2:** TVS sends in return a random challenge VoRi whose goal is to identify DC.
- **Step 3:** TVS replies by sending $\texttt{VoR}.\{\texttt{h(VoRi)}\}_{\texttt{VoKey}}$. Receiving this message, TVS both checks whether the challenge's answer is correct (by comparing the hashed value h(VoRi) with its own value), and extracts CW from the xored datum xor(CW,VoR) received at step 1 and using VoR. This is done by computing xor(xor(CW,VoR),VoR), and by applying sequentially rules II., IV. and III. of Fig. 1 to it, TVS obtains: $\texttt{xor(xor(CW,VoR),VoR)} \overset{\text{II.}}{\rightarrow} \texttt{xor(CW,xor(VoR,VoR))} \overset{\text{IV.}}{\rightarrow} \texttt{xor(CW,0)} \overset{\text{III.}}{\rightarrow} \texttt{CW}$.

Notice that Rule IV. of Fig. 1 is crucial at Step 3 of this protocol.

In [18], the authors verified that no old value of CW can be reused. Indeed, if the freshness of CW was not satisfied then we can imagine that the copy-protection would become obsolete. By storing all control words and by reusing them, an unethical individual could for instance decrypt the broadcasted program without paying the amount. However, the model considered is strongly typed in the sense that the authors handle the XOR operator only for terms satisfying a particular given form.

In order to consider type confusing attacks, we propose to verify the secrecy of CW in an untyped model for the XOR algebaric properties. Using the method developed in this

paper, we have succeeded in verifying this. Furthermore, using the family of approximation functions defined in [6] we have succeeded in verifying it automatically.

We have implemented our approach within the TA4SP tool presented in the following subsection.

4.2 Using TA4SP

This tool, whose method is detailed in [6], is one of the four official tools of the AVISPA tool-set [2]. The particularity of this tool is verifying of secrecy properties for an unbounded number of sessions.

The structure of the TA4SP tool is detailed in Fig. 3.

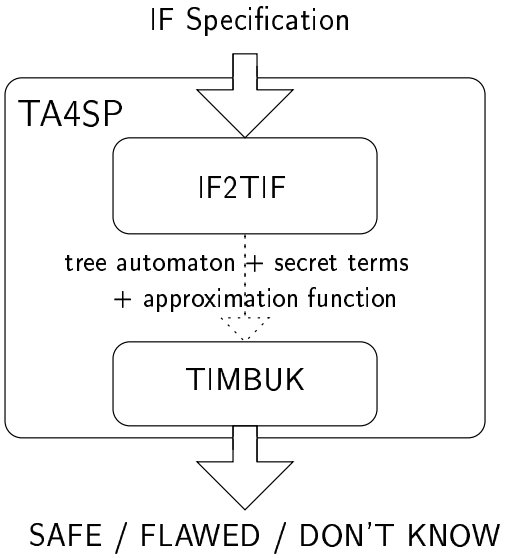


Fig. 3. TA4SP tool

The language IF is a low level specification language automatically generated from HLPSL (High Level Protocol Specification Language) [9] in the AVISPA toolset.

The TA4SP tool is made up of:

- IF2TIF, a translator from IF to a specification well-adapted to TIMBUK+, and
- TIMBUK+,[2] a collection of tools for achieving proofs of reachability over term rewriting systems and for manipulating tree automata. This tool has been initially developed by

[2] Timbuk is available at http://www.irisa.fr/lande/genet/timbuk/.

Th. Genet (IRISA/ INRIA-Rennes, FRANCE) and improved to handle our approximation functions.

Let us remark that the available version of `TA4SP` at `http://www.avispa-project.org` used in the framework of the AVISPA project is not yet updated with the `XOR` features. It is intended that this be updated in the near future.

5 Conclusion

This paper shows that the symbolic approximation-based approach we have been developing is well-adapted for analyzing security protocols using algebraic properties while considering an unbounded number of sessions. Indeed, the automatically generated symbolic approximation function enables us 1) an automated normalization of transitions, and 2) an automated completion procedure. Notice that the variables used to manipilate algebraic properties are not typed, like in [23]. Our symbolic approximation-based framework allowing us to handle algebraic properties does not deal with timestamps.

The tool `TA4SP` has been updated to take the exclusive or algebraic property of cryptographic primitives into account. This way the feasibility of the analysis has been confirmed by the experimentation on the *view-only* protocol. Future development concerns the implementation optimization.

We intend to investigate further algebraic properties that can be handled in practice. We anticipate that it could be carried out for algebraic properties expressed by quadratic rules. At this stage, experiments should be performed again.

To the best of our knowledge, this is the first attempt to automatically handle a large class of algebraic properties used in cryptographic protocols. Indeed, we wish to emphasize the fact that our theoretical framework is supported by a *push-button* tool `TA4SP` [3,5]. Moreover, `TA4SP` is used for protocols specified in the standard High Level Protocol Specification Language (HLPSL) [9,4]. This language is known to be suitable for industrial users.

These two significant advantages make it possible to use our framework and the fully automatic tool in the industrial context.

References

1. Smartright technical white paper v1.0. Technical report, Thomson, http://www.smartright.org, October 2001.
2. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santos Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the automated validation of internet security protocols and applications. In K. Etessami and S. Rajamani, editors, *17th International Conference on Computer Aided Verification, CAV'2005*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285, Edinburgh, Scotland, 2005. Springer.

3. A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Viganò, and L. Vigneron. The AVISS Security Protocol Analysis Tool. In *Proceedings of CAV'02*, LNCS 2404, pages 349–354. Springer, 2002.
4. AVISPA. Deliverable 2.1: The High-Level Protocol Specification Language. Available at *http://www.avispa-project.org*, 2003.
5. AVISPA. Deliverable 7.2: Assessment of the AVISPA tool v.1. Available at *http://www.avispa-project.org*, 2003.
6. Y. Boichut, P.-C. Héam, and O. Kouchnarenko. Automatic Verification of Security Protocols Using Approximations. Research Report RR-5727, INRIA-Lorraine - CASSIS Project, October 2005.
7. Y. Boichut, P.-C. Héam, O. Kouchnarenko, and F. Oehl. Improvements on the Genet and Klay technique to automatically verify security protocols. In *Proc. Int. Ws. on Automated Verification of Infinite-State Systems (AVIS'2004), joint to ETAPS'04*, pages 1–11, Barcelona, Spain, April 2004. The final version will be published in EN in Theoretical Computer Science, Elsevier.
8. L. Bozga, Y. Lakhnech, and M. Perin. HERMES: An automatic tool for verification of secrecy in security protocols. In *CAV: International Conference on Computer Aided Verification*, 2003.
9. Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, J. Mantovani, S. Mödersheim, and L. Vigneron. A high level protocol specification language for industrial security-sensitive protocols. In *Proceedings of Workshop on Specification and Automated Processing of Security Requirements (SAPS)*, volume 180, Linz, Austria, September 2004. Oesterreichische Computer Gesellschaft (Austrian Computer Society).
10. Y. Chevalier, R. Kusters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with XOR. *TCS: Theoretical Computer Science*, 338, 2005.
11. I. Cibrario, L. Durante, R. Sisto, and A. Valenzano. Automatic detection of attacks on cryptographic protocols: A case study. In Christopher Kruegel Klaus Julisch, editor, *Intrusion and Malware Detection and Vulnerability Assessment: Second International Conference*, volume 3548 of *Lecture Notes in Computer Science*, Vienna, 2005.
12. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 2002. http://www.grappa.univ-lille3.fr/tata/.
13. H. Comon-Lundh and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. *Theoretical Computer Science*, 331(1):143–214, February 2005.
14. V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 2005.
15. N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. Int. Ws. on Formal Methods and Security Protocols*, Italy, Trento, August 1999.
16. G. Feuillade, Th Genet, and V. Viet Triem Tong. Reachability analysis of term rewriting systems. *Journal of Automated Reasonning*, 33, 2004.
17. Th. Genet and F. Klay. Rewriting for cryptographic protocol verification. In *Proc. Int. Conf. CADE'00*, LNCS 1831, pages 271–290. Springer-Verlag, 2000.
18. Th. Genet, Y.-M. Tang-Talpin, and V. Viet Triem Tong. Verification of copy-protection cryptographic protocol using approximations of term rewriting systems. In *Proc. of WITS'03, Workshop on Issues in the Theory of Security*, 2003.
19. D. Monniaux. Abstracting cryptographic protocols with tree automata. In *Sixth International Static Analysis Symposium (SAS'99)*, number 1694 in Lecture Notes in Computer Science. Springer-Verlag, 1999.

20. H. Ohsaki and T. Takai. Actas: A system design for associative and commutative tree automata theory. In *Proc. of the 5th Int. Ws. on Rule-Based Programming: RULE'2004*, Aachen, Germany, June 2004. To appear in ENTCS.
21. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *14th IEEE Computer Security Foundations Workshop (CSFW '01)*, pages 174–190, Washington - Brussels - Tokyo, June 2001. IEEE.
22. T. Truderung. Regular protocols and attacks with regular knowledge. In *Proc. of 20th Int. Conf. on Automated Deduction (CADE'05)*, volume 3632 of *LNCS*, pages 377–391. Springer, 2005.
23. R. Zunino and P. Degano. Handling exp, x (and timestamps) in protocol analysis. To appear in Proc. of Int. Conf. FOSSACS'06, 2006.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399