Tree Automata for Detecting Attacks on Protocols with Algebraic Cryptographic Primitives

Boichut[1]

INRIA-LANDE
IRISA
Rennes, France

Héam[2] Kouchnarenko[3]

INRIA-CASSIS
LIFC
Besançon, France

Abstract

This paper extends a rewriting approximations-based theoretical framework in which the security problem – secrecy preservation against an active intruder – may be semi-decided through a reachability verification. In a recent paper, we have shown how to semi-decide whether a security protocol using algebraic properties of cryptographic primitives is safe. In this paper, we investigate the dual - insecurity - problem: we explain how to semi-decide whether a protocol using cryptographic primitive algebraic properties is unsafe. The main advantage of our work is that the approximation functions make it possible to automatically verify security protocols with an arbitrary number of sessions. Furthermore, our approach is supported by the tool `TA4SP` successfully applied for analysing the NSPK-xor protocol and the Diffie-Hellman protocol.

Keywords: Security protocol, algebraic properties, automatic approximation.

1 Introduction

Security protocols are part of systems for which the security problem is in general undecidable. Approximations and abstractions represent a well-suited alternative for verifying them in practice. A lot of investigations have been carried out on this topic [2,11,6,14,16,19,15,18].

An often encountered difficulty is about encoding with non-atomic keys. A non-atomic key is a key established in several steps from several data. This topic comes

[1] Email: `boichut@irisa.fr`
[2] Email: `heamp@lifc.univ-fcomte.fr`
[3] Email: `kouchna@lifc.univ-fcomte.fr`

close to the handling of operators with algebraic properties. On a strongly typed model (model in which the structure of a compound key is clearly specified), most of the developed methods are able to perform a protocol analysis. Unfortunately a secure strongly typed model is not a secure model because of type confusing attacks.

That is why our previous contribution [4] has extended the verification method in [3] in order to verify – without typing – security protocols bringing into play operators with algebraic properties. This improvement has made the computation of sound over-approximations of the intruder knowledge possible. Consequently, the safety, i.e., the secrecy preservation on protocols using algebraic properties of the *exclusive or* (`xor`) operator or the *exponential* (`exp`) operator can be established automatically. However, there is a lack of the attack detection, i.e. of showing that a protocol is unsafe.

The main contribution of this paper consists of showing the feasibility of the automatic unsafety verification for protocols when 1) the number of sessions is unbounded, and 2) the cryptographic primitives use algebraic operators properties. We propose sufficient conditions on term rewriting systems (TRSs for short), under which attack detection on such protocols becomes possible.

To illustrate the contributions, experiments on the detection of attacks against protocols with the primitives using `xor` or `exp` (xored and exped protocols, for short), are reported.

Structure of the paper The paper is organised as follows. After giving preliminary notions on tree automata and TRSs, we introduce in Section 2 a substitution depending on rules of a TRS, and a notion of compatibility between such substitutions and finite tree automata, both suitable for reachability analysis in rewriting with non left-linear TRSs. In Section 3, we present the extension of [4] dealing with under-approximations. Finally, before concluding, we give in Section 4 a brief overview of related works, and we explain how to apply the obtained new results to analyse xored or exped protocols.

2 Background and Notations

In this section basic notions on finite tree automata, term rewriting systems and approximations are recalled. The reader is referred to [8] for more detail.

2.1 Notations

Given the set $\mathbb{N}$ of natural integers, $\mathbb{N}^*$ 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 $t(p) \in \mathcal{F}_n$ (with $n \neq 0$), then $\{i \mid p.i \in \mathcal{P}\mathrm{os}(t)\} = \{1, \ldots, n\}$ and, for each $p \in \mathcal{P}\mathrm{os}(t)$, if $t(p) \in \mathcal{X}$ or $t(p) \in \mathcal{F}_0$, 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 $\mathcal{P}os_{\mathcal{K}}(t)$ 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 $\mathcal{P}os(t) = \mathcal{P}os_{\mathcal{F}}(t)$ (i.e. such that $\mathcal{P}os_{\mathcal{X}}(t) = \emptyset$). 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: $\mathcal{P}os(t_{|p}) = \{i \mid p.i \in \mathcal{P}os(t)\}$ and, For all $j \in \mathcal{P}os(t_{|p})$, $t_{|p}(j) = t(p.j)$. We denote by $t[s]_p$ the term obtained by replacing in t the subterm $t_{|p}$ by s. See Example 6.1.

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 $\mathcal{R}$ 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 TRS is left-linear if for each rule $l \to r$, every variable occur at most once in l. 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 classically denoted $\to_{\mathcal{R}}$. If $t \to_{\mathcal{R}} t'$ for $t, t' \in \mathcal{T}(\mathcal{F})$, then t is a *rewriting predecessor* of t' and t' is *rewriting successor* of t. 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 recognised 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. See Example 6.2.

In [4], a new kind of substitution has been introduced. We recall this definition below. Notice that the domain of these substitutions is not the set of variables anymore, but a set of positions. Thus, given a variable, this allows a symbolic representation of its values.

Definition 2.1 Let $\mathcal{R}$ be a term rewriting system, $\mathcal{Q}$ a set of states and $l \to r \in \mathcal{R}$. A $(l \to r)$-substitution is an application from $\mathcal{P}os_{\mathcal{X}}(l)$ into $\mathcal{Q}$.

We then adapt this kind of substitution to the rewriting framework in the following way. Let $l \to r \in \mathcal{R}$ and σ be a $(l \to r)$-substitution. We denote by $l\sigma$ the term of $\mathcal{T}(\mathcal{F}, \mathcal{Q})$ such that $\mathcal{P}os(l\sigma) = \mathcal{P}os(l)$, and for each $p \in \mathcal{P}os(l)$, if $p \in \mathcal{P}os_{\mathcal{X}}(l)$ 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: $\mathcal{P}os(r\sigma) = \mathcal{P}os(r)$ and, for each $p \in \mathcal{P}os(r)$, if $p \notin \mathcal{P}os_{\mathcal{X}}(r)$ then $r\sigma(p) = r(p)$ and $r\sigma(p) = \sigma(l(p^{'}))$ otherwise, where $p^{'} = \min \mathcal{P}os_{r(p)}(l)$ (positions are lexicographically ordered). For a given tree automaton, a particular class of $(l \to r)$-substitution can be drawn.

Definition 2.2 Let $\mathcal{A}$ be a finite tree automaton. We say that a $(l \to r)$-substitution σ is $\mathcal{A}$-compatible if for each $x \in \mathcal{V}ar(l)$,

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

See Example 6.3. Finally, the last notion we introduce is the definition of an approximation function.

Definition 2.3 Let $\mathcal{A}$ be a finite tree automaton. An approximation function (for $\mathcal{A}$) is a function associating with 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 mapping from $\mathcal{P}os(r)$ to $\mathcal{Q}$.

See Example 6.4. This notion is very useful for reachability analysis in rewriting with non left-linear TRSs as shown in the following section.

2.2 Reachability Analysis in Rewriting with non Left-linear TRSs

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 [14,13] 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 [13]), 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 there is a fix-point automaton $\mathcal{A}_k$ such that $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_k)) = \mathcal{L}(\mathcal{A}_k)$, then one has $\mathcal{L}(\mathcal{A}_k) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$ (or $\mathcal{L}(\mathcal{A}_k) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ if $\mathcal{R}$ is not in one class of [13]). In particular, for non left-linear TRSs, the completion is not sound. Indeed, if the completion converges towards a fix-point automaton $\mathcal{A}_k$, $\mathcal{L}(\mathcal{A}_k)$ is not necessarily either $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ or a super set of $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$.

In [4], the completion procedure has been improved so that the method is sound for non left-linear TRSs. This technique is introduced below. As mentioned previously, the completion builds a sequence $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_k$ of tree automata such that the set of terms reachable in one step of rewriting from $\mathcal{L}(\mathcal{A}_i)$ are in $\mathcal{L}(\mathcal{A}_{i+1})$. 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 $\to_{\mathcal{A}_i}$. Formally, for an approximation function γ, a rule $l \to r \in \mathcal{R}$ and an $\mathcal{A}_i$-compatible $(l \to r)$-substitution σ, 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, such that $l\sigma \to^*_{\mathcal{A}_i} q$ and $r\sigma \not\to^*_{\mathcal{A}_i} q$, detected between $\mathcal{R}$ and $\mathcal{A}_i$, $\mathcal{A}_{i+1}$ is built by adding new transitions to $\mathcal{A}_i$, so that it recognizes $r\sigma$ in q, i.e. $r\sigma \to_{\mathcal{A}_{i+1}} q$.

$$\begin{tikzcd}
l\sigma \arrow[r, "\mathcal{R}"'] \arrow[d, "\mathcal{A}_i"', "*"] & r\sigma \arrow[dl, dashed, bend left, "*"', "\mathcal{A}_{i+1}"] \\
q &
\end{tikzcd}$$

Before giving a definition of a completion step (Def. 2.5), we introduce a normalisation step described in Definition 2.4 .
Let's remark that the transition $r\sigma \to q$ is not necessarily a 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: $\{a \to q_1, g(q_1) \to q_2, h(q') \to q_3, f(q_2, q_3) \to q\}$. The states used in a normalization step

do not grow on trees and it is of the approximation function γ concern to deliver them at each completion step. Formally,

Definition 2.4 Let $\mathcal{A} = (\mathcal{Q}_0, \Delta, F_0)$ be a finite tree automaton, γ be an approximation function for $\mathcal{A}$, $l \rightarrow r$ be a rule of $\mathcal{R}$, σ be an $\mathcal{A}$-compatible $(l \rightarrow r)$-substitution, and q be a state of $\mathcal{A}$. We denote by $\mathrm{Norm}_\gamma(l \rightarrow r, \sigma, q)$ the following set of transitions, called *normalization* of $(l \rightarrow r, \sigma, q)$:

$$\begin{aligned}\{f(q_1, \ldots, q_k) \rightarrow q^{'} \ | p \in \mathcal{P}\mathrm{os}_\mathcal{F}(r),\ r(p) = f, &\\ q^{'} = q \text{ if } p = \varepsilon \text{ otherwise } q^{'} = \gamma(l \rightarrow r, \sigma, q)(p) &\\ q_i = \gamma(l \rightarrow r, \sigma, q)(p.i) \text{ if } p.i \notin \mathcal{P}\mathrm{os}_\mathcal{X}(r), &\\ q_i = \sigma(\min\{p^{'} \in \mathcal{P}\mathrm{os}_\mathcal{X}(l) \mid l(p^{'}) = r(p.i)\})\text{otherwise}\}&\end{aligned}$$

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 TRS, variables occurring in the right-hand side must, by definition, occur in the left-hand side too.

Definition 2.5 Let $\mathcal{R}$ be a TRS. 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:

$$\Delta_1 = \Delta_0 \cup \bigcup \mathrm{Norm}_\gamma(l \rightarrow r, \sigma, q)$$

where the union involves all rules $l \rightarrow r \in \mathcal{R}$, all states $q \in \mathcal{Q}_0$, all $\mathcal{A}_0$-compatible $(l \rightarrow r)$-substitutions σ such that $l\sigma \rightarrow^*_{\mathcal{A}_0} q$ and $r\sigma \not\rightarrow^*_{\mathcal{A}_0} q$, $F_1 = F_0$ and $\mathcal{Q}_1 = \mathcal{Q}_0 \cup \mathcal{Q}_2$, where $\mathcal{Q}_2$ denotes the set of states occurring in left/right-hand sides of transitions of Δ_1.

See Example 6.5 for an example of a completion step. Following theorem was proved in [4].

Theorem 2.6 *Let $(\mathcal{A}_n)$ and (γ_n) be respectively a sequence of finite tree automata and a sequence of approximation functions such that 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 there exists a positive integer N, such that for every $n \geq N$, $\mathcal{A}_n = \mathcal{A}_N$, then $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{A}_N)$.*

From a verification point of view, this technique is very helpful. Indeed, for a system Σ whose transition relation is Δ, one specifies the initial configuration of Σ by a tree language E, and Δ by a TRS $\mathcal{R}$. With a well-suited approximation function γ, an over-approximation of reachable configurations of Σ, denoted $E^\gamma_\mathcal{R}$, can be computed. Finally, a set of bad configurations, denoted E_{Bad}, can be encoded with a tree language and if $E^\gamma_\mathcal{R} \cap E_{Bad}$ is empty, then no bad configuration is reachable.

In particular, in [4], we have used this technique for verifying security protocols bringing into play the `xor` operator ($\oplus$). Note that the nilpotence property of $\oplus$ is specified with a non left-linear rule, i.e., $x \oplus x \rightarrow 0$. The tree languages specify the intruder knowledge and the configurations of the network. The TRS specifies the protocol and the intruder abilities for decoding, coding, depairing messages. Thus, if a secret term t does not belong to an over-approximation of the knowledge that the intruder might have, then t is actually secret.

3 Under-Approximations for non Left-linear TRSs

The over-approximation results in [4] do not provide a way to prove that a particular term is reachable: the method is not complete. This section adapts the means and extends the results in [4] to under-approximations computations. In the security protocol framework, computing under-approximations allows an under-estimation of the intruder knowledge, and thus secrecy flaws detection. Indeed, if a secret datum is in the intruder knowledge under-estimation, then the intruder actually knows this secret.

The main idea (and problem) behind the under-approximations is that one wants the languages of computed tree automata to be in the set of terms reachable by rewriting . Having some conditions on the TRS makes it possible to control the completion, and proving that a term is actually reachable is then possible.

We define here γ to be an injective approximation function from $\mathcal{R} \times (\mathbb{N}^* \mapsto \mathcal{Q}) \times \mathbb{N}^* \times \mathcal{Q}$ into $\mathcal{Q}$. Theorem 3.2 shows that with such an approximation function, an under-approximation of the set of reachable terms is possible. Before, Lemma 3.1 presents an intermediary result useful for proving Theorem 3.2: this result reveals some features of terms recognised by $\mathcal{C}_\gamma(\mathcal{A})$ for which there exists a rewriting predecessor recognised by $\mathcal{A}$.

In the following, we introduce the notation $NLV(t)$ which for a term t of $\mathcal{T}(\mathcal{F}, \mathcal{X})$, denotes the set of non-linear variables of t, i.e., the set of variables occurring at least twice within t.

Lemma 3.1 *Let $\mathcal{R}$ be a right-linear TRS for which $NLV(l) \cap \mathcal{V}\text{ar}(r) = \emptyset$ for all $l \rightarrow r \in \mathcal{R}$. Let $\mathcal{A}$ be the current tree automaton and $\mathcal{C}_\gamma(\mathcal{A})$ be the tree automaton obtained after one completion step with $\mathcal{R}$ and γ. If there exist a ground term t over $\mathcal{F}$, a state q of $\mathcal{A}$ and a function τ from $\mathcal{P}\text{os}(t)$ to Q such that $t \in \mathcal{L}(\mathcal{C}_\gamma(\mathcal{A}), q)$, $t \notin \mathcal{L}(\mathcal{A}, q)$ and τ satisfies the following conditions: (i) $\tau(\varepsilon) = q$; (ii) for all $p \in \mathcal{P}\text{os}(t)$, $t_{|p} \in \mathcal{L}(\mathcal{C}_\gamma(\mathcal{A}), \tau(p))$ and, (iii) for all $p \in \mathcal{P}\text{os}(t) \setminus \{\varepsilon\}$, if $\tau(p)$ is a state of $\mathcal{A}$, then $t_{|p} \in \mathcal{L}(\mathcal{A}, \tau(p))$. Then there exists $t_0 \in \mathcal{T}(\mathcal{F})$ such that $t_0 \in \mathcal{L}(\mathcal{A}, q)$ and $t_0 \rightarrow_{\mathcal{R}} t$.*

The proof of Lemma 3.1 is in Appendix 8.1.
The following result shows that each term of the language $\mathcal{C}_\gamma(\mathcal{A}_0)$ is reachable by rewriting from $\mathcal{A}_0$ and using $\mathcal{R}$.

Theorem 3.2 *Let $\mathcal{A}_0 = (\mathcal{Q}_0, \Delta_0, F_0)$ be a finite tree automaton. Let $\mathcal{R}$ be a right-linear TRS. Given the approximation function γ defined at the beginning of Section 3, if for all $l \rightarrow r \in \mathcal{R}$, $\mathcal{V}\text{ar}(r) \cap NLV(l) = \emptyset$ then $\mathcal{L}(\mathcal{C}_\gamma(\mathcal{A}_0)) \subseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$.*

The proof of Theorem 3.1 can be found in Appendix 8.2. Let $\mathcal{C}_\gamma^{(n)}(\mathcal{A}_0)$ be the tree automaton obtained after n completion steps performed from $\mathcal{A}_0$ by using the TRS $\mathcal{R}$ and the approximation function γ. Finally, Proposition 3.3 shows that the approximation function γ provides a sound under-approximation of reachable terms (see Appendix 8.3 for the proof).

Proposition 3.3 *If $\mathcal{R}$ is right-linear and for all $l \to r \in \mathcal{R}$, $NLV(l) \cap \mathcal{V}ar(r) = \emptyset$ then for all $n \leq 0$, $\mathcal{L}(\mathcal{C}_\gamma^{(n)}(\mathcal{A}_0)) \subseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$, $\mathcal{L}(\mathcal{C}_\gamma^{(n)}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{C}_\gamma^{(n+1)}(\mathcal{A}_0))$ and $\bigcup_{n\geq 0} \mathcal{L}(\mathcal{C}_\gamma^{(n)}(\mathcal{A}_0)) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$.*

At this point, we have developed theoretical frameworks which lead either to over-approximations of the set of reachable terms in general, or to its under-approximations under additional conditions on TRSs. The obtained results allow us to apply the approximation-based methods to system verification as presented in the next section.

4 Experiments and Related Works

With the extension brought for the under-approximations computation, we are now able to detect whether a protocol using algebraic properties of cryptographic primitives is flawed or not. In this section, we present some experimental results obtained on two protocols, well-known to be flawed, which are NSPK-xor and the key establishment à la Diffie-Helmann protocol. The technique presented in this paper has been implemented in the tool `TA4SP` (a description of the tool is given in Appendix 9).

4.1 TA4SP for Attack Detection

This section details two protocols, well-known to be flawed, which are NSPK-xor and the key establishment à la Diffie-Helmann protocol in its simplest form. The notations used are the following: `X -> Y: Z` specifies that `X` sends the message `Z` to `Y`, `X.Y` is the concatenation of data `X` and `Y`, and $\{\texttt{X}\}_{\texttt{Y}}$ (or `{X}_Y`) is the encoding of the message `X` by the message `Y`. Moreover, data `Na`, `Nb`, `ni(Na)` and `ni(Nb)` with `i` being an integer, are fresh random numbers, also called a *nonces*. Finally, the last concept to know concerns the keys, which can be public, private or symmetric. To a public key `Pka` is associated a private key `Prka`. A message encoded by one can be decoded by the other: $\{\{\texttt{M}\}_{\texttt{Pka}}\}_{\texttt{Prka}} = \{\{\texttt{M}\}_{\texttt{Prka}}\}_{\texttt{Pka}} = \texttt{M}$. A symmetric key `K` can decode a message encoded by itself: $\{\{\texttt{M}\}_{\texttt{K}}\}_{\texttt{K}} = \texttt{M}$.

The NSPK-xor Protocol is composed of three steps so that each participant can authenticate the other. First, the agent A sends the message $\{Na.A\}_{K_B}$ to the agent B. Second, B sends $\{Nb.Na \oplus B\}_{K_A}$ to A. Finally, A sends $\{Nb\}_{K_B}$ to B as a confirmation. Using `TA4SP`, we obtain in 71.03 seconds that the protocol does not preserve the secrecy of the data Nb against an intruder. Thanks to the `AVISPA` toolset, one can use one of three other tools (in this case CL-AtSe [20]) for exhibiting the following attack trace.

```
1. i -> (a,6):  start
2. (a,6) -> i:  {n9(Na).a}_ki

3. i -> (a,3):  start
4. (a,3) -> i:  {n1(Na).a}_kb

5. i -> (b,4):  {xor(i,xor(b,n9(Na))).a}_kb
6. (b,4) -> i:  {n5(Nb).xor(i,n9(Na))}_ka
              & Secret(n5(Nb),set_62);
              & Add a to set_62;  Add b to set_62;

7. i -> (a,6):  {n5(Nb).xor(i,n9(Na))}_ka
8. (a,6) -> i:  {n5(Nb)}_ki
```

At steps `1.` and `2.` of the attack, the agent `a` initiates a session with the intruder by sending the message `{n9(Na).a}_ki` to the intruder where `n9(Na)` is a nonce generated by `a` and `ki` is the public key of the intruder. At steps `3.` and `4.`, the agent `a` initiates a session with the agent `b`. The intruder composes at step `5.` the message `xor(i,xor(b,n9(Na))).a` and sends it to `b` after having encoded it with the public key of the agent `b`. The agent `b` deduces at step `6.` that this message comes from the agent `a` thanks to the identity occurring in the received message. Moreover, `b` considers the message xor(i,xor(b,n9(Na)))' as the nonce generated by `a`. Consequently, `b` performs the second step of the protocol. At step `6.` of the attack trace, `b` composes `n5(Nb).xor(b,xor(i,xor(b,n9(Na))))` which is equivalent to `n5(Nb).xor(i,n9(Na))` after considering the algebraic properties of $\oplus$ (xor operator). Then, he sends it to `a` after having encoded it with the public key of `a`. The agent `b` declares also the nonce `n5(Nb)` as a secret shared between himself and the agent `a`. But, according to the point of view of the agent `a`, the message `{n5(Nb).xor(i,n9(Na))}_ka` should come from `i` (the intruder) because `n5(Nb)` identifies the agent `i` for `a`. According to his deduction, the agent `a` sends `{n5(Nb)}_ki` to the intruder. Finally, the latter can deduce `n5(Nb)` which is a secret supposed to be shared between `b` and `a`.

The Diffie-Helmann Protocol is a key establishment protocol between two agents A and B. The simplest version of this protocol is composed of three steps. At step 1, A generates the nonce Na and computes $exp(G, Na)$ (standing for G^{Na}) where G is a number known by every agents. Thus A sends the message $exp(G, Na)$ to the agent B. At step 2, the agent B generates also a number Nb and computes on the one hand $exp(G, Nb)$ and on the other hand $K = exp(X, Nb)$ where X is the message received i.e. $exp(G, Na)$. The former is sent to A and the latter stands for the symmetric key shared between A and B. As soon as B receives the message $exp(G, Nb)$ from A, (s)he then computes $exp(exp(G, Nb), Na)$ and thus considers it as the symmetric key shared with A. Indeed, according to the algebraic properties of the exponentiation, $K = exp(exp(G, Na), Nb) = exp(exp(G, Nb), Na)$. Finally, the message $\{secret\}_K$ is sent by A to B in which $secret$ is a datum initially known uniquely by A and B. Using `TA4SP` this protocol has been shown as being flawed in 24.73 seconds. For this protocol, a MIM (*Man in the Middle*) attack is known and is detailed below with the attack trace outputted with the `AVISPA` tool-set.

```
1. i -> (a,3):  start
2. (a,3) -> i:  exp(g,n1(Na))

3. i -> (b,4):  g
4. (b,4) -> i:  exp(g,n5(Nb))

5. i -> (a,3):  g
6. (a,3) -> i:  {secab}_(exp(g,n1(Na)))

7. i -> (b,4):  {secab}_(exp(g,n5(Nb)))
8. (b,4) -> i:  ()
              & Secret(exp(g,n5(Nb)),set_65);  Add a to set_65;
              & Add b to set_65;
```

Roughly, the intruder establishes two keys: `exp(exp(g,n1(Na)),g)` with `a` at steps 2 and 5 and `exp(exp(g,n5(Nb)),g)` with `b` at steps 3 and 4. At step 6, the agent `a` sends the secret data to `b` with the key unfortunately shared with the intruder. The intruder then extracts the secret data and forwards it to `b` with the other key. Finally, `b` is persuaded that this message comes from `a`.

4.2 Related Work

In [17] 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 algorithms. However, the engine has still room to be modified and optimised to support an automated verification.

In [10], the authors study the IBM 4758 CCA (Common Cryptographic Architecture) API which has been shown as flawed in [5]. In response to this flaw, IBM then has proposed three recommendations designed to prevent it. The formalisation of these recommendations leads Cortier et al. to draw up a particular class of security protocols using the operator $\oplus$ for which the secrecy problem is decidable with an unbounded number of sessions. They have then shown that any one of the three recommendations is sufficient to secure the API against a Dolev-Yao intruder [12].

In the recent survey [9], the authors give an overview of the existing methods in formal approaches to analyse 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 emphasise the fact that the results remain theoretical, and very few implementations automatically verify protocols with algebraic properties.

5 Conclusion

The main purpose of this paper is to show that the symbolic approximation-based approach we have been developing is well-adapted for detecting attacks on protocols using algebraic properties while considering an unbounded number of sessions. Indeed, the automatically generated symbolic under-approximation function enables us 1) an automated normalisation of transitions, and 2) an automated completion procedure within the set of reachable terms.

With this extension our approximation-based framework proposes verification methods using either over-approximations of the set of reachable terms in general, or its under-approximations under additional conditions on TRSs. The contributions of the paper have been integrated into the push-button tool `TA4SP` [1] successfully applied for analysing the NSPK-xor protocol and the Diffie-Hellman protocol. Let us remark that `TA4SP` is used for protocols specified in the standard High Level Protocol Specification Language (HLPSL) [7]. This language is known to be suitable for industrial users.

Future development concerns implementation optimisation. We intend to investigate further algebraic properties that can be handled in practice. In this direction, we project to develop a theoretical framework in order to compute under-approximations without the right-linearity condition required Theorem 3.2. This may for example provide an approximation-based approach for detecting attacks on security protocols with cryptographic primitives using the homomorphism property [9].

References

[1] 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 *CAV 2005, Proceedings*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005.

[2] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *CSFW 2001*, pages 82–96. IEEE Computer Society Press, 2001.

[3] 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.

[4] Y. Boichut, P.-C. Héam, and O. Kouchnarenko. Handling algebraic properties in automatic analysis of security protocols. In *ICTAC-06*, volume 4281 of *Lecture Notes in Computer Science*, pages 153–167. Springer Berlin/Heidelberg, 2006.

[5] M. Bond. Attacks on cryptoprocessor transaction sets. In *CHES 2001, Proceedings*, Lecture Notes on Computer Science, pages 220–234. Springer Verlag, 2001.

[6] L. Bozga, Y. Lakhnech, and M. Perin. Pattern-based abstraction for verifying secrecy in protocols. In *TACAS 2003*, volume 2619 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[7] 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 *SAPS 2004*, 2004.

[8] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 2002.

[9] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 14:1–43, 2006.

[10] V. Cortier, G. Keighren, and G. Steel. Automatic analysis of the security of xor-based key management schemes. In *TACAS 2007*, 2007. To be published in Lecture Notes on Computer Science.

[11] V. Cortier, J. K. Millen, and H. Rueß. Proving secrecy is easy enough. In *14th IEEE Computer Security Foundations Workshop, CSFW 2001, Proceedings*, pages 97–110. IEEE, June 2001.

[12] D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[13] G. Feuillade, T. Genet, and V. VietTriemTong. Reachability analysis over term rewriting systems. *Journal of Automated Reasonning*, 33 (3-4), 2004.

[14] Th. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *CADE 2000, Proceedings*, volume 1831 of *Lecture Notes in Computer Science*, pages 271–290. Springer-Verlag, 2000.

[15] C. Meadows. The NRL protocol analyser: An overview. *Journal of Logic Programming*, 1994.

[16] D. Monniaux. Abstracting cryptographic protocols with tree automata. In *SAS 1999*, volume 1694 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[17] H. Ohsaki and T. Takai. Actas: A system design for associative and commutative tree automata theory. In *RULE'2004*, volume 124, Aachen, Germany, June 2004.

[18] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1):85–128, 1998.

[19] D. Song. Athena: A new efficient automatic checker for security protocol analysis. In *CSFW 1999, Proceedings*, pages 192–202. IEEE Computer Society Press, 1999.

[20] M. Turuani. The cl-atse protocol analyser. In *RTA 2006*, volume 4098 of *Lecture Notes in Computer Science*, pages 277–286. Springer-Verlag, 2006.

Appendix

6 Basic Examples

Example 6.1 Let $f, g, a \in \mathcal{F}$ be functional symbols such that $f \in \mathcal{F}_2$, $g \in \mathcal{F}_1$ and $a \in \mathcal{F}_0$. Let $x \in \mathcal{X}$ be a variable. Let t be a term of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $t = f(a, g(x))$, thus $\mathcal{P}\mathrm{os}(t) = \{\epsilon, 1, 2, 2.1\}$, $\mathcal{P}\mathrm{os}_{\mathcal{F}}(t) = \{\epsilon, 1, 2\}$, $t(1) = a$, $t(2) = g$, $t(\epsilon) = f$, $t_{|1} = a$, $t_{|2} = g(x)$, $t_{|2.1} = x$, $\mathcal{P}\mathrm{os}_{\{x\}}(t) = \{2.1\}$ and $t[a]_2 = f(a, a)$.

Example 6.2 Let $\mathcal{A} = (\mathcal{Q}, \Delta, F)$ be a tree automaton such that $\mathcal{F} = \{f, g, a\}$ with $f \in \mathcal{F}_2$, $g \in \mathcal{F}_1$ and $a \in \mathcal{F}_0$, $\mathcal{Q} = \{q_f, q_1\}$, $F = \{q_f\}$ and $\Delta = \{f(q_1, q_1) \to q_f, a \to q_1, g(q_1) \to q_1\}$. Then, $\mathcal{L}(\mathcal{A}, q_1) = \{g^*(a)\}$ and $\mathcal{L}(\mathcal{A}, q_f) = \mathcal{L}(\mathcal{A}) = \{f(g^*(a), g^*(a))\}$.

Example 6.3 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 $t = A$ can be reduced to q_f and belongs to $\mathcal{L}(\mathcal{A}, \sigma_{\text{exe}}(1))$. Furthermore $t \to q_0$, so $t \in \mathcal{L}(\mathcal{A}, \sigma_{\text{exe}}(2.1))$. Therefore σ_{exe} is $\mathcal{A}$-compatible.

Example 6.4 Consider the automaton $\mathcal{A}_{\text{exe}}$, the term rewriting system $\mathcal{R}_{\text{exe}}$ and the substitution σ_{exe} defined in Example 6.3. For σ_{exe}, an approximation function γ_{exe} may be defined by:

$$\gamma_{\text{exe}}(l \to r, \sigma_{\text{exe}}, q_f)(\varepsilon) = q_1, \quad \gamma_{\text{exe}}(l \to r, \sigma_{\text{exe}}, q_f)(1) = q_0, \quad \gamma_{\text{exe}}(l \to r, \sigma_{\text{exe}}, q_f)(2) = q_1.$$

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

Example 6.5 [A completion step] Following Example 6.4, ε and 1 are the functional positions of $r = h(A, y)$. We set $q^{'}$ of the definition to be equal to q_f. Thus $\mathrm{Norm}_{\gamma_{\text{exe}}}(l \to r, \sigma_{\text{exe}}, q_f)$ is of the form $\{A \to q?, h(q?, q??) \to q_f\}$. Since for r, the position 1 is a functional 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:

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

Consequently, the tree automaton resulting from a completion step on $\mathcal{A}_{\text{exe}}$ with γ_{exe} and $\mathcal{R}_{\text{exe}}$ is $\mathcal{C}_\gamma(\mathcal{A}_{\text{exe}}) = (\{q_0, q_f\}, \Delta_{\text{exe}} \cup \{A \to q_0, h(q_0, q_f) \to q_f\}, \{q_f\})$.

Notice that a new completion step could be performed on $\mathcal{C}_\gamma(\mathcal{A}_{\text{exe}})$. However, no transition would be added since no new critical pair would be detected. So, $\mathcal{C}_\gamma(\mathcal{A}_{\text{exe}})$ is the fix-point automaton. According to Theorem 2.6, every term reachable by rewriting from $\mathcal{L}(\mathcal{A}_{\text{exe}})$ are in the language of the fix-point automaton.

7 Example of a Completion Procedure

In this section we explain how our approach works on a toy example . We do not give the details of a protocol study since involving term rewriting systems are too huge to be readable.

We consider terms defined by

- $\mathcal{F}_0 = \{0\}$,
- $\mathcal{F}_1 = \{\mathrm{Inv}, s\}$,
- $\mathcal{F}_2 = \{+\}$ and
- $\mathcal{F}_{k\geq 3} = \emptyset$.

In this formalism, the symbol s denotes the successor function. For instance, $s(s(s(0)))$ is the successor of the successor of the successor of 0 and denotes the integer 3. The operator Inv denotes the inverse (for the addition). For example, $\mathrm{Inv}(s(0))$ is the inverse of the successor of 0 and denotes the integer -1.

We use the following term rewriting system to encode addition and subtraction over $\mathbb{Z}$. To simplify notations, we write $(x+y)$ or $x+y$ for $+(x,y)$.

$$\begin{aligned}
\mathcal{R} = \{&\mathrm{Inv}(\mathrm{Inv}(x))\rightarrow x && (1)\\
&x\rightarrow \mathrm{Inv}(\mathrm{Inv}(x)) && (2)\\
&x+\mathrm{Inv}(x)\rightarrow 0 && (3)\\
&x+y\rightarrow y+x && (4)\\
&x+(y+z)\rightarrow (x+y)+z && (5)\\
&x+0\rightarrow x && (6)\\
&x+s(0)\rightarrow s(x) && (7)\\
&s(x)\rightarrow x+s(0) && (8)\\
&\mathrm{Inv}(s(x))\rightarrow \mathrm{Inv}(s(s(x)))+s(0)\} && (9)
\end{aligned}$$

Notice that this term rewriting system is not left-linear (Rule (7)).

We are interested in the following problem: given three integers a, b and c, are there integers λ and μ such that

$$\lambda a + \mu b = c?$$

A basic number theory result states that the answer the previous question is yes if and only if c is a multiple of the greatest common divisor of a and b.

For instance, it is possible for $a = 7$, $b = 3$ and $c = 15$ (since $\gcd(a, b) = 1$). We may prove it using the above term rewriting system. Indeed, from $s^7(0)$ and $s^3(0)$ one can reach $s^{15}(0)$ using $+$, Inv and rewriting rules. For example:

$$s^3(0)\rightarrow_{12} s^2(0) + s(0)\rightarrow_{12} (s(0) + s(0)) + s(0)$$

Consequently

$$s^3(0) + s^3(0) \to^*_{12} ((s(0) + s(0)) + s(0)) + s^3(0) \to^*_{12,9} s^6(0) \tag{10}$$

Similarly one has

$$(((s^7(0) + s^7(0)) + s^7(0)) \to_{12,9} s^21(0) \tag{11}$$

Moreover, from (10) one has

$$\mathrm{Inv}(s^3(0) + s^3(0)) \to^*_{12,9} \mathrm{Inv}(s^6(0)) \to^*_{9,8} \mathrm{Inv}(s^{21}(0)) + s^{15}(0)$$

Therefore, by (11), one has

$$(((s^7(0)+s^7(0))+s^7(0))+\mathrm{Inv}(s^3(0)+s^3(0)) \to^*_{8,12,9} (s^21(0)+\mathrm{Inv}(s^{21}(0)))+s^{15}(0) \to_{7,10} s^{15}(0).$$

Now we prove that the problem has no solution for $a = 2$, $b = 4$ and $c = 5$ (this is mathematically trivial, the goal is just to illustrate that it can be proved automatically by our over-approximation approach).

We consider for initial terms the language accepted by the following tree automaton $\mathcal{A}$:

- States are q_0, q_1, q_2, q_3, q_4, q_{-2} q_{-4} and q_f,
- Final states are q_2,q_{-2}, q_{-4}, q_4, and q_f,
- Transitions are
  - $0 \to q_0$, $s(q_0) \to q_1$, $s(q_1) \to q_2$, $s(q_2) \to q_3$, $s(q_3) \to q_4$ (encodes that $s^2(0)$ and $s^4(0)$ are initially known),
  - $\mathrm{Inv}(q_4) \to q_{-4}$ (encodes that one can compute the inverse of 4),
  - $\mathrm{Inv}(q_2) \to q_{-2}$ (encodes that one can compute the inverse of 2),
  - $q_{f_1} + q_{f_2} \to q_f$ for all final states q_{f_1}, q_{f_2}, (encodes that one can do the addition of two computed integers terms).

We want to prove that $s^5(0) \notin \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$.

We give some details on the first completion step.

Rule (5) This rule doesn't provide new transition. Indeed, there is no state q in $\mathcal{A}$ such that $\mathrm{Inv}(\mathrm{Inv}(q))$ can be derived in $\mathcal{A}$ to a state.

Rule (6) For each state q one has to add the normalisation of the transition $\mathrm{Inv}(\mathrm{Inv}(q)) \to q$. Assume that

$$\gamma(Rule(6), \{\mathrm{e} \mapsto q_1\}, q_1)(1) = q_3$$

. Then during the completion step, the normalisation of $\mathrm{Inv}(\mathrm{Inv}(q_1)) \to q_1$ ensures that we add the transitions $\mathrm{Inv}(q_1) \to q_3$ and $\mathrm{Inv}(q_3) \to q_1$. With similar assumptions on γ one adds during the first completion step $\mathrm{Inv}(q_0) \to q_0$, $\mathrm{Inv}(q_{-4} \to q_4)$ and $\mathrm{Inv}(q_{-2}) \to q_2$.

Rule (7) Since $q_4 + \mathrm{Inv}(q_4) \to^*_{\mathcal{A}} q_f$, one has to add the transition $0 \to q_f$.

Rule (8) This rule doesn't provide new transition.

Rule (9) This rule doesn't provide new transition.

Rule (10) This rule doesn't provide new transition.

Rule (11) This rule doesn't provide new transition.

Rule (12) Since $s(q_0) \rightarrow_{\mathcal{A}} q_1$ and $q_0 + s(0) \not\rightarrow^*_{\mathcal{A}} q_1$, one has to add the following transitions (with correct assumptions on γ) $0 \rightarrow q_0$, $s(q_0) \rightarrow q_1$ (these two transitions are already in $\mathcal{A}$) and $q_0 + q_1 \rightarrow q_1$. Similarly, one has to add transitions $q_0 + q_2 \rightarrow q_2$, $q_0 + q_3 \rightarrow q_3$, $q_0 + q_4 \rightarrow q_4$.

Rule (12) Since $\mathrm{Inv}(s(q_1)) \rightarrow^*_{\mathcal{A}} q_{-2}$ and $\mathrm{Inv}(s(s(q_1)) + s(0)) \not\rightarrow^*_{\mathcal{A}} q_{-2}$, one has to add the transitions (with correct assumption on γ), $s(0) \rightarrow q_1$, $s(q_1) \rightarrow q_2$, $s(q_2) \rightarrow q_3$, $\mathrm{Inv}(q_3) \rightarrow q_1$, $q_1 + q_1 \rightarrow q_2$ and $\mathrm{Inv}(q_2) \rightarrow q_{-2}$.

Similar completion steps lead to the following tree automaton $\mathcal{B}$:

- States of $\mathcal{B}$ are $q_{-4}, q_{-2}, q_1, q_2, q_3, q_4$ and q_f.
- Final states are q_2, q_4, q_{-2}, q_{-4} and q_f.
- Transitions on constants are $0 \rightarrow q_0$ and $0 \rightarrow q_f$.
- Transitions with symbol s are given by the following table:

	q_0	q_1	q_2	q_3	q_4
s	q_1	q_2	q_3	q_4	q_1

  For instance, $s(q_{-2}) \rightarrow q_3$ is a transition.

- Transitions with symbol Inv are given by the following table:

	q_{-4}	q_{-2}	q_0	q_1	q_2	q_3	q_4	q_f
Inv	q_4	q_2	q_0	q_3	q_{-2}	q_1	q_{-4}	q_f

- Transitions with symbol $+$ are given by the following table:

$+$	q_{-4}	q_{-2}	q_0	q_1	q_2	q_3	q_4	q_f
q_{-4}	q_{-4}, q_f	q_{-2}, q_f	q_{-4}	q_1	q_2, q_f	q_3	q_0, q_4, q_f	q_f
q_{-2}	q_{-2}, q_f	q_0, q_f	q_{-2}, q_f	q_3	q_0, q_4, q_f	q_1	q_2, q_f	q_f
q_0	q_{-4}, q_f	q_{-2}, q_f	q_0	q_1	q_2, q_f	q_3	q_4, q_f	$\emptyset$
q_1	q_1	q_3	q_1	q_2, q_f	q_3	q_4, q_0, q_f	q_f	$\emptyset$
q_2	q_2, q_f	q_0, q_4, q_f	q_2, q_f	q_3	q_4, q_f, q_0	q_1	q_2, q_f	q_f
q_3	q_3	q_1	q_3	q_4, q_0, q_f	q_1	q_2, q_f	q_3	$\emptyset$
q_4	q_0, q_4, q_f	q_2, q_f	q_4, q_0, q_f	q_1	q_2, q_f	q_3	q_4, q_f, q_0	q_f
q_f	q_f	q_f	$\emptyset$	$\emptyset$	q_f	$\emptyset$	q_f	q_f

The automaton $\mathcal{B}$ is stable by the $\mathcal{C}_\gamma$ completion. Consequently, it accepts an over-approximation of reachable terms of $\mathcal{A}$ by $\mathcal{R}$. Since $s^5(0) \notin \mathcal{L}(\mathcal{B})$, its proved that we may not have $\lambda.2 + \mu.4 = 5$ with $\lambda, \mu \in \mathbb{Z}$.

8 Omitted Proof Details

8.1 Proof of Lemma 3.1

To simplify the notation we denote by Δ_1 the set of transitions of the automaton $\mathcal{C}_\gamma(\mathcal{A})$, Δ_0 the set of transitions of $\mathcal{A}$ and $\mathcal{Q}_0$ the set of states of $\mathcal{A}$.

The proof consists of 1) the construction of a term $s_1 \in \mathcal{T}(\mathcal{F}, \mathcal{Q})$ such that

$$t \rightarrow^*_{\Delta_1} s_1 \rightarrow_{\mathrm{Norm}_\gamma(l\rightarrow r,\sigma,q)} q, \tag{12}$$

2) the construction, by iterating a backward process, of a term $s \in \mathcal{T}(\mathcal{F}, \mathcal{Q})$ such that

$$t \rightarrow^*_{\Delta_1} s \rightarrow^*_{\mathrm{Norm}_\gamma(l\rightarrow r,\sigma,q)} q, \text{ and} \tag{13}$$

3) the proof that

$$t \rightarrow^*_{\Delta_0} r\sigma \rightarrow^*_{\mathrm{Norm}_\gamma(l\rightarrow r,\sigma,q)} q. \tag{14}$$

First, using (ii) at the position ε gives

$$t_{|\varepsilon} \rightarrow^*_{\Delta_1} \tau(\varepsilon).$$

Since $t = t_{|\varepsilon}$ and since $\tau(\varepsilon) = q$ (by (i)), one has

$$t \rightarrow^*_{\Delta_1} q.$$

Since $t \in \mathcal{T}(\mathcal{F})$ one has $t \neq q$, and every derivation $t \rightarrow^*_{\Delta_1} q$ has the length one, at least. Consequently, there exists $s_1 \in \mathcal{T}(\mathcal{F}, \mathcal{Q})$ such that

$$t \rightarrow^*_{\Delta_1} s_1 \rightarrow_{\Delta_1} q.$$

We now show by contradiction that the transition $s_1 \rightarrow q \notin \Delta_0$. Suppose that $s_1 \rightarrow q$ is a transition of Δ_0. Then $s_1 \in \mathcal{T}(\mathcal{F}, \mathcal{Q}_0)$. Thus, using (iii), $t \rightarrow^*_{\Delta_0} s_1 \rightarrow_{\Delta_0} q$, a contradiction ($t \not\rightarrow^*_{\Delta_0} q$).

Therefore, the transition $s_1 \rightarrow q$ is in $\Delta_1 \setminus \Delta_0$. By definition of Δ_1 (see Definition 2.5), there exist $q^{'}$, $\sigma : \mathcal{P}\mathrm{os}_X(l)^* \mapsto \mathcal{Q}$ and $l \rightarrow r \in \mathcal{R}$ such that $s_1 \rightarrow_{\mathcal{C}_\gamma(\mathcal{A})} q \in \mathrm{Norm}_\gamma(r\sigma \rightarrow q^{'}, l \rightarrow r)$ and

$$l\sigma \rightarrow^*_{\Delta_0} q^{'}. \tag{15}$$

Now by definitions of $\mathrm{Norm}_\gamma(r\sigma \rightarrow q^{'}, l \rightarrow r)$ and γ, each source state or target state of a transition in $\mathrm{Norm}_\gamma(r\sigma \rightarrow q^{'}, l \rightarrow r)$ is either $\mathcal{Q} \setminus \mathcal{Q}_0$, or is equal to $q^{'}$. Since $s_1 \rightarrow_{\mathcal{C}_\gamma(\mathcal{A})} q \in \mathrm{Norm}_\gamma(r\sigma \rightarrow q^{'}, l \rightarrow r)$, either $q \in \mathcal{Q} \setminus \mathcal{Q}_0$, or $q = q^{'}$. Because $q \in \mathcal{Q}_0$, one has $q = q^{'}$ and

$$t \rightarrow^*_{\Delta_1} s_1 \rightarrow_{\mathrm{Norm}_\gamma(l\rightarrow r,\sigma,q)} q.$$

We are done for (12). We now perform an iterative construction. If $s_1 \notin \mathcal{T}(\mathcal{F}, \mathcal{Q}_0)$, then there exists a position p of s_1 such that $s_1(p) \in \mathcal{Q} \setminus \mathcal{Q}_0$. Thus $s_1(p)$ is of the form $s_1(p) = \gamma(l \rightarrow r, \sigma, q)(p)$. Since γ is injective, the only transition of Δ_1 leading to $s_1(p)$ is

$$r(p)(\gamma(l \rightarrow r, \sigma, q)(p.1), \ldots, \gamma(l \rightarrow r, \sigma, q)(p.\ell)) \rightarrow s_1(p).$$

Consequently, the derivation $t \rightarrow^*_{\Delta_1} s_1$ has to conclude by

$$t \rightarrow^*_{\Delta_1} s_2 \rightarrow s_1$$

where

$$s_2 = s_1[r(p)(\gamma(l \rightarrow r, \sigma, q)(p.1), \ldots, \gamma(l \rightarrow r, \sigma, q)(p.\ell))]_p.$$

So, one has

$$t \rightarrow^*_{\Delta_1} s_2 \rightarrow_{\text{Norm}_\gamma(l \rightarrow r, \sigma, q)} s_1 \rightarrow_{\text{Norm}_\gamma(l \rightarrow r, \sigma, q)} q.$$

Now, if $s_2 \notin \mathcal{T}(\mathcal{F}, \mathcal{Q}_0)$, the same construction can be iteratively applied to s_2 and so on. Consequently, one can build a term $s \in \mathcal{T}(F, \mathcal{Q}_0)$ such that $\mathcal{P}\text{os}(s) = \mathcal{P}\text{os}(r)$ and

$$t \rightarrow^*_{\Delta_1} s \rightarrow^*_{\text{Norm}_\gamma(l \rightarrow r, \sigma, q)} q, \tag{16}$$

and for each position p of s such that $s(p) \notin \mathcal{Q}$,

$$s(p) = r(p). \tag{17}$$

We are done for (13) .

We can begin the last part of the proof. Let $q_1, \ldots, q_n$ be the states occurring in s by reading s from the left to the right. Let $p_1, \ldots, p_n$ be respectively the positions in s of states $q_1, \ldots, q_n$. Notice that the backward construction of s is deterministic. Indeed every derivation from t to q can be split up to

$$t \rightarrow^*_{\Delta_1} s \rightarrow^*_{\text{Norm}_\gamma(l \rightarrow r, \sigma, q)} q.$$

It implies that for each q_i, with $i = 1, \ldots, n$, one has

$$q_i = \tau(p_i). \tag{18}$$

At this stage, s is of the form $r\sigma$ since γ is defined for every position of r.

Now using (18) and the hypothesis iii), one has

$$t \rightarrow_{\Delta_0} r\sigma \rightarrow^*_{\text{Norm}_\gamma(l \rightarrow r, \sigma, q)} q.$$

The TRS $\mathcal{R}$ being right-linear with $NLV(l) \cap \mathcal{V}\text{ar}(r) = \emptyset$ for each rule $l \rightarrow r$ of $\mathcal{R}$, one can built a substitution $\mu : \mathcal{P}\text{os}_{\mathcal{X}}(l) \mapsto \mathcal{T}(\mathcal{F})$ such that:

- For $p \in \mathcal{P}\text{os}_{\mathcal{V}\text{ar}(r)}(l)$, one can set $\mu(p) = t'$ and $t' = t|_{p'}$ with $p' \in \mathcal{P}\text{os}_{\{l|_p\}}(r)$. Moreover, since $l|_p \notin NLV(l)$, one obtains $\mu(p) = t' \rightarrow^*_{\Delta_0} \sigma(p)$.
- For $p \in \mathcal{P}\text{os}_{\mathcal{V}\text{ar}(l) \setminus \mathcal{V}\text{ar}(r)}(l)$, one can proceed in the following way:

- if $l(p) \in NLV(l)$ then one can set $\mu(p'_1), \ldots, \mu(p'_1)$ to t' where $t' \in \mathcal{L}(\mathcal{A}_0, \sigma(p'_1)) \cap \ldots \cap \mathcal{L}(\mathcal{A}_0, \sigma(p'_n))$ with $\{p'_1, \ldots, p'_n\} = \mathcal{P}\mathrm{os}_{\{l(p)\}}(l)$.
- Otherwise, one can set $\mu(p)$ to a term $t' \in \mathcal{L}(\mathcal{A}_0, \sigma(p))$.

By this way, there exists $t_0 = l\mu \in \mathcal{T}(\mathcal{F})$ such that $t_0 \rightarrow^*_{\mathcal{A}_0} q$ and $t_0 \rightarrow_{\mathcal{R}} t$, proving the lemma.

8.2 Proof of Theorem 3.2

Let $\mathcal{P}_n$ be the following proposition:
For all $t \in \mathcal{L}(\mathcal{C}_\gamma(\mathcal{A}_0))$, if there exists a function τ from $\mathcal{P}\mathrm{os}(t)$ to $\mathcal{Q}$ such that $\tau(\varepsilon) = q_f$ and for all $p \in \mathcal{P}\mathrm{os}(t)$,

$$t|_p \rightarrow^*_{\mathcal{C}_\gamma(\mathcal{A}_0)} \tau(p) \quad \text{and} \quad t[\tau(p)]_p \rightarrow^*_{\mathcal{C}_\gamma(\mathcal{A}_0)} q_f$$

and such that

$$|\{p \in \mathcal{P}\mathrm{os}(t) \mid \tau(p) \in \mathcal{Q}_0 \ \wedge \ t|_p \not\rightarrow^*_{\mathcal{A}_0} \tau(p)\}| = n,$$

then $t \in \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$.

We prove that $\mathcal{P}_n$ is true for all $n \geq 0$ by induction on n. To simplify notations, let

$$NR(t,\tau) = \{p \in \mathcal{P}\mathrm{os}(t) \mid \tau(p) \in \mathcal{Q}_0 \text{ and } t|_p \not\rightarrow^*_{\mathcal{A}_0} \tau(p)\}.$$

$\mathcal{P}_0$: Assume that t and τ satisfy the hypothesis on $\mathcal{P}_0$. We have $|NR(t,\tau)| = 0$. In particular, $\varepsilon \notin NR(t,\tau)$. So, $t = t|_\varepsilon \rightarrow_{\mathcal{A}_0} \tau(\varepsilon) = q_f$. Since $\mathcal{A}_0$ and $\mathcal{C}_\gamma(\mathcal{A}_0)$ have the same set of final states, $t \in \mathcal{L}(\mathcal{A}_0)$.

$\mathcal{P}_n \implies \mathcal{P}_{n+1}$: Assume that $\mathcal{P}_n$ is true for $n \geq 0$ and that t and τ satisfy the hypothesis on $\mathcal{P}_{n+1}$. Since $NR(t,\tau)$ is non-empty, let p be a maximal element of $NR(t,\tau)$ (for the lexicographical order). Then, by maximality of p, one can apply Lemma 3.1 to $t_{|p}$. Thus, there exists $t_0 \in \mathcal{T}(\mathcal{F})$ such that $t_0 \rightarrow^*_{\mathcal{A}_0} \tau(p)$ and $t_0 \rightarrow_{\mathcal{R}} t_p$. Therefore, there exists a function τ_1 from $\mathcal{P}\mathrm{os}(t_0)$ into $\mathcal{Q}_0$ such that for all $p^{'}$, $t_0 \rightarrow^*_{\mathcal{A}_0} \tau_1(p^{'})$, $t[\tau_1(p^{'})]_{p^{'}} \rightarrow^*_{\mathcal{C}_\gamma(\mathcal{A}_0)} \tau(p)$. We define the function τ_2 from $\mathcal{P}\mathrm{os}(t[t_0]_p)$ to $\mathcal{Q}$ as follows.
- If p is not a prefix of $p^{'}$, then $\tau_2(p^{'}) = \tau(p^{'})$,
- Otherwise, if $p^{'}$ is of the form $p.u$, then $\tau_2(p^{'}) = \tau_1(u)$.

By construction, $t[t_0]_p \rightarrow_{\mathcal{R}} t$ and $|NR(t[t_0]_p, \tau_2)| = n - 1$. Thus, by induction, $t \in \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$.

8.3 Proof of Proposition 3.3

By definition $\mathcal{C}^{(n+1)}_\gamma(\mathcal{A}_0) = g_\gamma(\mathcal{C}^{(n)}_\gamma(\mathcal{A}_0)))$. Consequently, the set of transitions of $\mathcal{C}^{(n)}_\gamma(\mathcal{A}_0)$ is included in the transitions set of $\mathcal{C}^{(n+1)}_\gamma(\mathcal{A}_0)$. Thus $\mathcal{L}(\mathcal{C}^{(n)}_\gamma(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{C}^{(n+1)}_\gamma(\mathcal{A}_0))$.

Now, using a Lemma 2 of [4] leading to Theorem 2.6, one has for all $n \geq 1$,

$$\mathcal{R}(\mathcal{L}(\mathcal{C}^{(n)}_\gamma(\mathcal{A}_0))) \subseteq \mathcal{L}(\mathcal{C}^{(n+1)}_\gamma(\mathcal{A}_0)).$$

Consequently, by a direct induction,

$$\mathcal{R}^{\leq n}(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{C}_\gamma^{(n+1)}(\mathcal{A}_0)).$$

It implies that

$$\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \subseteq \bigcup_{n\geq 0} \mathcal{L}(\mathcal{C}_\gamma^{(n)}(\mathcal{A}_0)).$$

One can prove that for all $n \in \mathbb{N}$, $\mathcal{L}(\mathcal{C}_\gamma^{(n)}(\mathcal{A}_0)) \subseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$ by a direct induction on n using Theorem 3.2, and we are done.

9 TA4SP Description

The `TA4SP` [4] tool, whose method is detailed in [3], is one of the four official tools of the `AVISPA` tool-set [1]. A version of `TA4SP` without xored extensions is freely available within the AVISPA toolset at `http://www.avispa-project.org`. The main particularity of this tool is the ability for verifying secrecy properties for an unbounded number of sessions.

The structure of the `TA4SP` tool is detailed in Fig. 1.

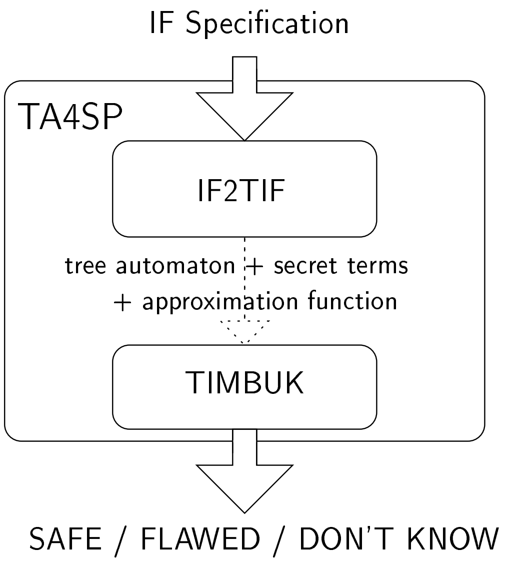

Figure 1. `TA4SP` tool

The language `IF` is a low level specification language automatically generated from the `HLPSL` (High Level Protocol Specification Language) [7] 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+`, [5] a collection of tools for achieving proofs of reachability over term rewriting systems and for manipulating tree automata. This tool has been initially de-

[4] A distribution of the `TA4SP` tool will be soon available at `http://www.irisa.fr/lande/boichut/ta4sp.html`.

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

veloped by Th. Genet (IRISA/ INRIA-Rennes, FRANCE) and enhanced in order to handle our approximation functions.

Note that the tool `TA4SP` may also answer “FLAWED” while performing under-approximations.