Rewriting for Cryptographic Protocol Verification

Thomas Genet[1] Francis Klay[2]

[1] IRISA / Université de Rennes 1
Campus de Beaulieu
F-35042 Rennes Cedex
genet@irisa.fr

[2] CNET / France Telecom
2, avenue Pierre Marzin
F-22307 Lannion Cedex
Francis.Klay@cnet.francetelecom.fr

Abstract. On a case study, we present a new approach for verifying cryptographic protocols, based on rewriting and on tree automata techniques. Protocols are operationally described using Term Rewriting Systems and the initial set of communication requests is described by a tree automaton. Starting from these two representations, we automatically compute an over-approximation of the set of exchanged messages (also recognized by a tree automaton). Then, proving classical properties like confidentiality or authentication can be done by automatically showing that the intersection between the approximation and a set of prohibited behaviors is the empty set. Furthermore, this method enjoys a simple and powerful way to describe intruder work, the ability to consider an unbounded number of parties, an unbounded number of interleaved sessions, and a theoretical property ensuring safeness of the approximation.

Introduction

In this paper, we present a new way of verifying cryptographic protocols. We do not aim here at discovering attacks on the protocol but our goal is to prove that there is not any, which is a more difficult problem. In practice, positive proofs of security properties on cryptographic protocols are highly desirable results since they give a better guarantee on the reliability of the protocol than any amount of passed tests. In [9], a decidable approximation of the set of descendants (reachable terms) was presented. In this paper, we propose to apply those theoretical results to the verification of cryptographic protocols. Our case study is the Needham-Schroeder Public Key protocol [19] (NSPK for short). We chose this particular example for two reasons. First of all, this protocol is real but can be easily understood. The second reason is that, in spite of its apparent simplicity and robustness, and in spite of several verification attempts, this protocol designed in 1978 was proved insecure only in 1995 by G. Lowe [13] and in 1996 by C. Meadows [17]. In particular, G. Lowe found a smart attack invalidating the main security properties of the protocol. In this paper, we will use the corrected version of the NSPK protocol also proposed by G. Lowe in [14].

Starting from a TRS representing the protocol and a tree automaton recognizing the initial set of communication requests, we automatically compute a superset of the set of exchanged messages by over-approximating the set of reachable terms. This model – also a tree automaton – takes into account an unbounded number of parties, an unbounded number of interleaved sessions as well as a powerful intruder activity description. For building this model, we needed to extend the approximation technique of [9], initially designed to approximate functional programs encoded by left-linear TRSs, to the more general class of TRSs (possibly non left-linear) with associative and commutative symbols.

In section 1, we recall basic definitions of terms, term rewriting systems, and tree automata. In section 2, we recall the technique for approximating the set of descendants for left-linear term rewriting systems and regular set of terms [9]. In section 3, we shortly present the Needham-Schroeder Public Key protocol, comment on its expected properties and propose an encoding into a term rewriting system in section 4. However, the term rewriting system describing the NSPK is not left-linear, has Associative and Commutative (AC for short) symbols and, consequently, is out of the scope of the basic approximation technique of [9]. Thus, in section 5, we show how to extend our technique to the case of non left-linear and AC TRSs. We also describe the application of approximation to NSPK and show how to prove confidentiality and authentication properties. Finally, in section 6, we conclude, compare with other approaches and present ongoing developments.

1 Preliminaries

We now introduce some notations and basic definitions. Comprehensive surveys can be found in [7] for term rewriting systems, in [3] for tree automata and tree language theory, and in [11] for connections between regular tree languages and term rewriting systems.

Terms, Substitutions, Rewriting systems

Let $\mathcal{F}$ be a finite set of symbols associated with an arity function, $\mathcal{X}$ be a countable set of variables, $\mathcal{T}(\mathcal{F}, \mathcal{X})$ the set of terms, and $\mathcal{T}(\mathcal{F})$ the set of ground terms (terms without variables). Positions in a term are represented as sequences of integers. The set of positions in a term t, denoted by $\mathcal{P}os(t)$, is ordered by lexicographic ordering $\prec$. The empty sequence ϵ denotes the top-most position. If $p \in \mathcal{P}os(t)$, then $t|_p$ denotes the subterm of t at position p and $t[s]_p$ denotes the term obtained by replacement of the subterm $t|_p$ at position p by the term s. For any term $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, we denote by $\mathcal{P}os_{\mathcal{F}}(s)$ the set of functional positions in s, i.e. $\{p \in \mathcal{P}os(s) \mid p \neq \epsilon \text{ and } \mathcal{R}oot(s|_p) \in \mathcal{F}\}$ where $\mathcal{R}oot(t)$ denotes the symbol at position ϵ in t. A *ground context* is a term of $\mathcal{T}(\mathcal{F} \cup \{\square\})$ with exactly one occurrence of $\square$, where $\square$ is a special constant not occurring in $\mathcal{F}$. For any term $t \in \mathcal{T}(\mathcal{F})$, $C[t]$ denotes the term obtained after replacement of $\square$ by t in the ground context $C[\,]$. The set of variables of a term t is denoted by $\mathcal{V}ar(t)$. A term is linear if any variable of $\mathcal{V}ar(t)$ has exactly one occurrence in t. A substitution is a mapping σ from $\mathcal{X}$ into $\mathcal{T}(\mathcal{F},\mathcal{X})$, which can uniquely be extended to an endomorphism of $\mathcal{T}(\mathcal{F},\mathcal{X})$. Its domain $\mathcal{D}om(\sigma)$ is $\{x \in X \mid x\sigma \neq x\}$.

A term rewriting system $\mathcal{R}$ is a set of *rewrite rules* $l \to r$, where $l, r \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $l \notin \mathcal{X}$, and $\mathcal{V}ar(l) \supseteq \mathcal{V}ar(r)$. A rewrite rule $l \to r$ is *left-linear* (resp. *right-linear*) if the left-hand side (resp. right-hand side) of the rule is linear. A rule is linear if it is both left and right-linear. A TRS $\mathcal{R}$ is linear (resp. left-linear, right-linear) if every rewrite rule $l \to r$ of $\mathcal{R}$ is linear (resp. left-linear, right-linear).

The relation $\to_{\mathcal{R}}$ induced by $\mathcal{R}$ is defined as follows: for any $s, t \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $s \to_{\mathcal{R}} t$ if there exist a rule $l \to r$ in $\mathcal{R}$, a position $p \in \mathcal{P}os(s)$ and a substitution σ such that $l\sigma = s|_p$ and $t = s[r\sigma]_p$. The reflexive transitive closure of $\to_{\mathcal{R}}$ is denoted by $\to^*_{\mathcal{R}}$. The set of $\mathcal{R}$-descendants of a set of ground terms E is denoted by $\mathcal{R}^*(E)$ and $\mathcal{R}^*(E) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in E \text{ s.t. } s \to^*_{\mathcal{R}} t\}$.

Automata, Regular Tree Languages

Let $\mathcal{Q}$ be a finite set of symbols, with arity 0, called *states*. $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ is called the set of *configurations*. A *transition* is a rewrite rule $c \to q$, where $c \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ and $q \in \mathcal{Q}$. A *normalized transition* is a transition $c \to q$ where $c = q' \in \mathcal{Q}$ or $c = f(q_1, \ldots, q_n)$, $f \in \mathcal{F}$, $ar(f) = n$, and $q_1, \ldots, q_n \in \mathcal{Q}$. A bottom-up non-deterministic finite tree automaton (tree automaton for short) is a quadruple $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, where $\mathcal{Q}_f \subseteq \mathcal{Q}$ and Δ is a set of normalized transitions. A tree automaton is *deterministic* if there are no two rules with the same right hand side. The rewriting relation induced by Δ is denoted either by $\to_\Delta$ or by $\to_{\mathcal{A}}$. The tree language recognized by $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists q \in \mathcal{Q}_f \text{ s.t. } t \to^*_{\mathcal{A}} q\}$. For a given $q \in \mathcal{Q}$, the tree language recognized by $\mathcal{A}$ and q is $\mathcal{L}(\mathcal{A}, q) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \to^*_{\mathcal{A}} q\}$. A tree language (or a set of terms) E is *regular* if there exists a bottom-up tree automaton $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = E$. The class of regular tree languages is closed under boolean operations $\cup, \cap, \setminus$, and inclusion is decidable. A $\mathcal{Q}$-substitution is a substitution $\sigma : \mathcal{X} \mapsto \mathcal{Q}$. Let $\Sigma(\mathcal{Q},\mathcal{X})$ be the set of $\mathcal{Q}$-substitutions. For every transition, there exists an equivalent set of normalized transitions. Normalization consists in decomposing a transition $s \to q$, into a set $Norm(s \to q)$ of normalized transitions. The method consists in abstracting subterms s' of s s.t. $s' \notin \mathcal{Q}$ by states of $\mathcal{Q}$. We first define the abstraction function as follows:

Definition 1. *Let $\mathcal{F}$ be a set of symbols, and $\mathcal{Q}$ a set of states. For a given configuration $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \setminus \mathcal{Q}$, an* abstraction *of s is a mapping α:*

$$\alpha : \{s|_p \mid p \in \mathcal{P}os_{\mathcal{F}}(s)\} \mapsto \mathcal{Q}$$

The mapping α is extended on $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ by defining α as identity on $\mathcal{Q}$, i.e. $\forall q \in \mathcal{Q} : \alpha(q) = q$.

Definition 2. *Let $\mathcal{F}$ be a set of symbols, $\mathcal{Q}$ a set of states, $s \to q$ a transition s.t. $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ and $q \in \mathcal{Q}$, and α an abstraction of s. The set $Norm_\alpha(s \to q)$ of* normalized transitions *is inductively defined by:*

1. *if* $s = q$*, then* $Norm_\alpha(s \to q) = \emptyset$*, and*
2. *if* $s \in \mathcal{Q}$ *and* $s \neq q$*, then* $Norm_\alpha(s \to q) = \{s \to q\}$*, and*
3. *if* $s = f(t_1, \ldots, t_n)$*, then* $Norm_\alpha(s \to q) =$
   $\{f(\alpha(t_1), \ldots, \alpha(t_n)) \to q\} \cup \bigcup_{i=1}^{n} Norm_\alpha(t_i \to \alpha(t_i))$.

Example 1. Let $\mathcal{F} = \{f, g, a\}$ and $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, where $\mathcal{Q} = \{q_0, q_1, q_2, q_3, q_4\}$, $\mathcal{Q}_f = \{q_0\}$, and $\Delta = \{f(q_1) \to q_0, g(q_1, q_1) \to q_1, a \to q_1\}$.

• The languages recognized by q_1 and q_0 are the following: $\mathcal{L}(\mathcal{A}, q_1)$ is the set of terms built on $\{g, a\}$, i.e. $\mathcal{L}(\mathcal{A}, q_1) = \mathcal{T}(\{g, a\})$, and $\mathcal{L}(\mathcal{A}, q_0) = \mathcal{L}(\mathcal{A}) = \{f(x) \mid x \in \mathcal{L}(\mathcal{A}, q_1)\}$.

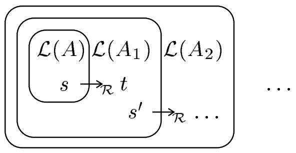
• Let $s = f(g(q_1, f(a)))$, and α_1 be an abstraction of s, mapping $g(q_1, f(a))$ to q_2, $f(a)$ to q_3 and a to q_4. The normalization of transition $f(g(q_1, f(a))) \to q_0$ with abstraction α_1 is the following: $Norm_{\alpha_1}(f(g(q_1, f(a))) \to q_0) = \{f(q_2) \to q_0, g(q_1, q_3) \to q_2, f(q_4) \to q_3, a \to q_4\}$.

2 Approximation Technique

For a regular set of terms $E \subseteq \mathcal{T}(\mathcal{F})$, although there exists some restricted classes of TRSs $\mathcal{R}$ such that $\mathcal{R}^*(E)$ is regular (see [5, 21, 4, 12]), this is not the case in general [11, 12]. In [9], for any tree automaton $\mathcal{A}$ (s.t. $\mathcal{L}(\mathcal{A}) \supseteq E$) and for any left-linear TRS $\mathcal{R}$, it is proposed to build an approximation automaton $\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})$ such that $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})) \supseteq \mathcal{R}^*(E)$. The quality of the approximation highly depends on an approximation function called γ which define some *folding positions*: subterms who can be approximated. We now briefly recall the construction of $\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})$ [9]:

Let $\mathcal{R}$ be a left-linear term rewriting system and $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ a tree automaton such that $E = \mathcal{L}(\mathcal{A})$ (or even $E \subseteq \mathcal{L}(\mathcal{A})$). First, we infinitely extend the set of states $\mathcal{Q}$ of $\mathcal{A}$ with an infinite number of new states, initially not occurring in $\mathcal{Q}$. Note that since we do not modify Δ nor $\mathcal{Q}_f$ (in particular, they remain finite), the language recognized by $\mathcal{A}$ is the same. On the other hand, it is always possible to come back to a finite set of states for $\mathcal{A}$ by restricting $\mathcal{Q}$ to the set of *accessible states*, i.e. states q such that $\mathcal{L}(\mathcal{A}, q) \neq \emptyset$.

Starting from $\mathcal{A}_0 = \mathcal{A}$, we incrementally build a finite number of tree automata $\mathcal{A}_i = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta_i \rangle$ with $i \geq 0$ such that $\forall i \geq 0 : \mathcal{L}(\mathcal{A}_i) \subset \mathcal{L}(\mathcal{A}_{i+1})$ until we get an automaton $\mathcal{A}_k$ with $k \in \mathbb{N}$ such that $\mathcal{L}(\mathcal{A}_k) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$, i.e. $\mathcal{L}(\mathcal{A}_k) \supseteq \mathcal{R}^*(E)$. We denote by $\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})$ this automaton $\mathcal{A}_k$. To construct $\mathcal{A}_{i+1}$ from $\mathcal{A}_i$, the technique consists in finding a term s in $\mathcal{L}(\mathcal{A}_i)$ such that $s \to_{\mathcal{R}} t$ and $t \notin \mathcal{L}(\mathcal{A}_i)$, and then in building Δ_{i+1} such that $\mathcal{L}(\mathcal{A}_i) \subset \mathcal{L}(\mathcal{A}_{i+1})$ and $t \in \mathcal{L}(\mathcal{A}_{i+1})$.



Since $\mathcal{A}_i$ and $\mathcal{A}_{i+1}$ only differs by their respective transitions sets, to ensure $\mathcal{L}(\mathcal{A}_i) \subset \mathcal{L}(\mathcal{A}_{i+1})$ it is enough to construct Δ_{i+1} such that it strictly contains

Δ_i. In order to have also $t \in \mathcal{L}(\mathcal{A}_{i+1})$ it is necessary to add some transitions to Δ_i to obtain Δ_{i+1}. This can be viewed as a *completion step* between the two term rewriting systems: the set of transitions Δ_i of $\mathcal{A}_i$ and $\mathcal{R}$. If there exists a term s in $\mathcal{L}(\mathcal{A}_i)$ such that $s \rightarrow_{\mathcal{R}} t$, by definition of $\rightarrow_{\mathcal{R}}$, there exists a rule $l \rightarrow r$, a ground context $C[\,]$ and a substitution (a match) σ such that $s = C[l\sigma] \rightarrow_{\mathcal{R}} C[r\sigma] = t$. On the other hand, by construction of tree automata, $s = C[l\sigma] \in \mathcal{L}(\mathcal{A}_i)$ means that (1) there exists a state $q \in \mathcal{Q}$ such that $l\sigma \rightarrow^*_{\mathcal{A}_i} q$ and (2) $C[q] \rightarrow^*_{\mathcal{A}_i} q'$ such that $q' \in \mathcal{Q}_f$. Hence, from (1) we know that we have following *critical pair* between transitions of $\mathcal{A}_i$ and rules of $\mathcal{R}$:

$$\begin{tikzcd} l\sigma \arrow[r, "\mathcal{R}"'] \arrow[d, "\mathcal{A}_i"', "*"] & r\sigma \\ q & \end{tikzcd}$$

Since every transition of $\mathcal{A}_i$ is in $\mathcal{A}_{i+1}$ (i.e. $\Delta_i \subseteq \Delta_{i+1}$), for the term t to be recognized by $\mathcal{A}_{i+1}$, it is enough to ensure that (3) $r\sigma \rightarrow^*_{\mathcal{A}_{i+1}} q$. This is sufficient since we can then rewrite $t = C[r\sigma]$ into $C[q]$ and from (2) we get that $C[q] \rightarrow^*_{\mathcal{A}_{i+1}} q'$, since $\Delta_i \subseteq \Delta_{i+1}$. Finally, since $q' \in \mathcal{Q}_f$, $t \in \mathcal{L}(\mathcal{A}_{i+1})$.

To ensure (3), we need to add some transitions to Δ_{i+1}, i.e. join the critical pair:

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

A direct solution to have $r\sigma \rightarrow^*_{\mathcal{A}_{i+1}} q$ is to have a transition of the form $r\sigma \rightarrow q$ in $\mathcal{A}_{i+1}$. However, this is not compatible with the standard normalized form of the tree automata we use here[1]. Thus, before adding $r\sigma \rightarrow q$ to transitions of $\mathcal{A}_i$, we normalize it first thanks to the $Norm_\alpha$ function (see definition 2). Hence, $\Delta_{i+1} = \Delta_i \cup Norm_\alpha(r\sigma \rightarrow q)$. We give here an example of completion process on a simple TRS

Example 2. Let $\mathcal{F} = \{f, g, a\}$ and $\mathcal{R}$ the one rule TRS $\mathcal{R} = \{f(g(x)) \rightarrow g(f(x))\}$. Let $\mathcal{A}_0 = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta_0 \rangle$ such that $\mathcal{Q}_f = \{q_f\}$ and $\Delta_0 = \{f(q_f) \rightarrow q_f, g(q_a) \rightarrow q_f, a \rightarrow q_a\}$. We have $\mathcal{L}(\mathcal{A}_0) = f^*(g(a))$. Between $\mathcal{R}$ and transitions of $\mathcal{A}_0$ there exists a critical pair:

$$\begin{tikzcd} f(g(q_a)) \arrow[r, "\mathcal{R}"'] \arrow[d, "\mathcal{A}_0"', "*"] & g(f(q_a)) \\ q_f & \end{tikzcd}$$

[1] keeping tree automata in standard normalized form allows, in particular, to apply usual algorithms: intersection, union, etc.

The $\mathcal{Q}$-substitution used here is $\sigma = \{x \mapsto q_a\}$. As defined before, we have $\Delta_1 = \Delta_0 \cup Norm_\alpha(g(f(q_a))) \to q_f)$. Let α be the abstraction function such that $\alpha(f(q_a)) = q_{new}$ where q_{new} is a state not occurring in transitions of $\mathcal{A}_0$. Then, we have $\Delta_1 = \Delta_0 \cup \{g(q_{new}) \to q_f,\ f(q_a) \to q_{new}\}$.

Except in some simple decidable case, this completion procedure is not guaranteed to converge but, instead, may infinitely add new transitions and thus generate an infinite number of tree automata $\mathcal{A}_1, \mathcal{A}_2$, etc. However, choosing particular values for α may force the completion process to converge by approximating infinitely many transitions by finite sets of more general transitions. Those particular abstraction functions are associated with *approximation functions* denoted by γ, defining some folding positions: positions in the right hand side of rules where subterms are approximated by regular languages: for each completion step from $\mathcal{A}_i$ to $\mathcal{A}_{i+1}$ involving a rewrite step $l\sigma \to_{\mathcal{R}} r\sigma$, a folding position p is a position in r which is assigned a state q' such that we only ensure $\mathcal{L}(\mathcal{A}_{i+1}, q') \supseteq \{r\sigma|_p\}$ instead of strict equality: $\mathcal{L}(\mathcal{A}_{i+1}, q') = \{r\sigma|_p\}$. This comes from the fact that the same state q' can be used for recognizing different terms obtained by different positions, rules or substitutions. The role of the approximation function is to relate $r\sigma|_p$ and the state q'. Folding positions depend on the applied rule $l \to r$ and on the substitution σ. Furthermore, since in our setting a rewriting step $s = C[l\sigma] \to_{\mathcal{R}} C[r\sigma] = t$ is modeled by a completion step on the critical pair $l\sigma \to_{\mathcal{R}} r\sigma$ and $l\sigma \to_{\mathcal{A}_i} q$, q is also a parameter of the approximation function. Finally, the approximation function γ maps every triple $(l \to r, q, \sigma)$ to a sequence of states (one for each position in $\mathcal{P}os_{\mathcal{F}}(r)$) used for the normalization of the transition $r\sigma \to q$.

Definition 3. *Let $\mathcal{Q}$ be a set of states and $\mathcal{Q}^*$ the set of sequences $q_1 \cdots q_k$ of states in $\mathcal{Q}$. An approximation function is a mapping $\gamma : \mathcal{R} \times \mathcal{Q} \times \Sigma(\mathcal{Q}, \mathcal{X}) \mapsto \mathcal{Q}^*$, such that $\gamma(l \to r, q, \sigma) = q_1 \cdots q_k$, where $k = \mathcal{C}ard(\mathcal{P}os_{\mathcal{F}}(r))$.*

From every $\gamma(l \to r, q, \sigma) = q_1 \cdots q_k$, we can associate $q_1, \ldots, q_k$ to positions $p_1, \ldots, p_k$ in $\mathcal{P}os_{\mathcal{F}}(r)$. This can be done by defining the corresponding abstraction function α on the restricted domain $\{r\sigma|_p \mid \forall l \to r \in \mathcal{R}, \forall p \in \mathcal{P}os_{\mathcal{F}}(r), \forall \sigma \in \Sigma(\mathcal{Q}, \mathcal{X})\}$:

$$\alpha(r\sigma|_{p_i}) = q_i$$

for all $p_i \in \mathcal{P}os_{\mathcal{F}}(r) = \{p_1, \ldots, p_k\}$, s.t. $p_i \prec p_{i+1}$ for $i = 1 \ldots k-1$ (where $\prec$ is the lexicographic ordering). In the following, we will note $Norm_\gamma$ the normalization function whose α value is defined according to γ as above.

Starting from a left-linear TRS $\mathcal{R}$, a tree automaton $\mathcal{A}$ and an approximation function γ, the *algorithm* for building the approximation automaton $\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A})$ is the following. First, set $\mathcal{A}_0$ to $\mathcal{A}$. Then, to construct $\mathcal{A}_{i+1}$ from $\mathcal{A}_i$:

1. search for a critical pair, i.e. a state $q \in \mathcal{Q}$, a rewrite rule $l \to r$ and a substitution $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ such that $l\sigma \to^*_{\mathcal{A}_i} q$ and $r\sigma \not\to^*_{\mathcal{A}_i} q$.
2. $\mathcal{A}_{i+1} = \mathcal{A}_i \cup Norm_\gamma(r\sigma \to q)$.

This process is iterated until it stops on a tree automaton $\mathcal{A}_k$ such that $\forall q \in \mathcal{Q}$, $\forall l \to r \in \mathcal{R}$ and $\forall \sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ if $l\sigma \to^*_{\mathcal{A}_k} q$ then $r\sigma \to^*_{\mathcal{A}_k} q$. Then, $\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A}) = \mathcal{A}_k$. The fact that $\mathcal{Q}$ and $\Sigma(\mathcal{Q}, \mathcal{X})$ may be infinite is not a problem in practice since, for finding a critical pair, we can restrict $\mathcal{Q}$ to the finite set of accessible states in $\mathcal{A}_i$, without changing $\mathcal{L}(\mathcal{A}_i)$ nor $\mathcal{L}(\mathcal{A}_{i+1})$. We now recall a theorem of [9].

Theorem 1. *(Completeness) Given a tree automaton $\mathcal{A}$ and a left-linear TRS $\mathcal{R}$, for any approximation function γ,*

$$\mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A})) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$$

The γ function fix the quality of the approximation. For example, one of the roughest approximation is obtained with a constant γ function mapping every triple $(l \to r, \sigma, q)$ to sequences of q' a unique state of $\mathcal{Q}$: $\forall l \to r \in \mathcal{R}, \forall \sigma \in \Sigma(\mathcal{Q}, \mathcal{X}), \forall q \in \mathcal{Q} : \gamma(l \to r, \sigma, q) = q' \cdots q'$. On the opposite, the best approximation consists in mapping every triple $(l \to r, \sigma, q)$ to sequences of distinct states. However, although any rough approximation built with the first γ is guaranteed to terminate, this is not necessarily the case for the second one.

On a practical point of view, the fact that completeness of the approximation construction does not depend on the chosen γ (Theorem 1) is a very interesting property. Indeed, it guarantees that for any approximation function, $\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A})$ is a safe model of $\mathcal{R}^*(E)$, in the sense of abstract interpretation.

Example 3. Back to the example 2, adding to $\mathcal{A}_0$ transitions $\{g(q_{new}) \to q_f, f(q_a) \to q_{new}\}$ to obtain $\mathcal{A}_1$ brings another critical pair:

$$\begin{tikzcd} f(g(q_{new})) \arrow[r, "\mathcal{R}"'] \arrow[d, "\mathcal{A}_1"', "*"] & g(f(q_{new})) \\ q_f & \end{tikzcd}$$

Like in the previous example, we build Δ_2 by adding $Norm_\alpha(f(g(q_{new})) \to q_f)$ to Δ_1. However, if α maps $g(q_{new})$ to another state q'_{new} not occurring in Δ_1, we add some new transitions and get another critical pair, and the process may go on for ever. Instead, we can here define an approximation function γ in a simple and static way, for example: $\forall \sigma \in \Sigma(\mathcal{Q}, \mathcal{X}), \forall q \in \mathcal{Q} :\ \gamma(f(g(x)) \to g(f(x)), q, \sigma) = q_{new}$. Since $\mathcal{P}os_{\mathcal{F}}(g(f(x))) = \{1\}$ is a singleton, note that the γ function maps triple of the form $(f(g(x)) \to g(f(x)), q, \sigma)$ to sequences of states of length one. This γ function defines a very rough approximation since the same state q_{new} is used for every normalization, whatever values q and σ may be. Thanks to this approximation function γ, the completion terminates. The value of Δ_1 remain the same but, for the next completion step, we have $Norm_\gamma(f(g(q_{new})) \to q_f) = \{g(q_{new}) \to q_f,\ f(q_{new}) \to q_{new}\}$. Thus, $\Delta_2 = \Delta_1 \cup \{f(q_{new}) \to q_{new}\}$, there is no new critical pair between Δ_2 and rule $f(g(x)) \to g(f(x))$, and we have $\mathcal{L}(\mathcal{A}_2) = f^*(g(f^*(a)))$.

Once $\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A})$ is obtained, it is easy to verify some reachability properties on $\mathcal{R}$ and E. It can be shown for example that a regular set of terms F cannot be reached from terms of E by $\rightarrow_{\mathcal{R}}{}^*$. This can be done by showing that $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})) \cap F = \emptyset$. We will apply this to the verification of the Needham-Schroeder Public Key Protocol in section 5.

3 Needham-Schroeder Public Key Protocol

In this section, we present our case study on the Needham-Schroeder Public Key protocol (NSPK). More precisely, we here use the fixed version of the protocol [14] without key server. Key servers have been discarded here for the sake of simplicity. Note that attacks from [14] have been found on the NSPK without key servers. Moreover, the approximation technique have also been successfully applied to the protocol with key servers.

The NSPK protocol aim at mutual authentication of two agents, an initiator A and a responder B, separated by an insecure network. Mutual authentication means that, when a protocol session is completed between two agents, they should be assured of each other's identity. In general, the main property expected for this kind of protocol is to prevent an intruder from impersonating one of the two agents. This protocol is based on an exchange of *nonces* (usually fresh random numbers or time stamps) and on *asymmetric* encryption of messages: every agent has a *public key* (for encryption) and a *private key* (for decryption). Every public key is supposed to be known by any agent whereas, the private key of agent X is supposed to be only known by X. Thus, in this setting, we suppose that messages encrypted with the public key of X can only be decrypted and read by X. Here is a description of the three steps of the fixed version of protocol, borrowed from [14]:

1. $A \hookrightarrow B : \{N_A, A\}_{K_B}$
2. $B \hookrightarrow A : \{N_A, N_B, B\}_{K_A}$
3. $A \hookrightarrow B : \{N_B\}_{K_B}$

In the first step, A tries to initiate a communication with B: A creates a nonce N_A and sends to B a message, containing N_A as well as his identity, encrypted with the public key of B: K_B. Then, in the second step, B sends back to A a message encrypted with the public key of A, containing the nonce N_A that B received, a new nonce N_B, and B's identity. Finally, in the last step, A returns the nonce N_B he received from B. If the protocol is completed, mutual authentication of the two agents is ensured:

- as soon as A receives the message containing the nonce N_A, sent back by B at step 2., A *believes* that this message was really built and sent by B. Indeed, N_A was encrypted with the public key of B and, thus, B is the only agent that is able to send back N_A,
- similarly, when B receives the message containing the nonce N_B, sent back by A at step 3., B *believes* that this message was really built and sent by A.

Another property that may be expected for this kind of protocol is *confidentiality* of nonces. In particular, if nonces remain confidential, they can be used later as keys for symmetric encryption of communications between A and B. Thus, confidentiality of nonces may also be of interest.

A cryptographic protocol is supposed to resist to any attack of an intruder. In particular for NSPK, we intend to show that, for agents respecting the protocol, and whatever the intruder may do,

– nonces and private keys remain confidential (confidentiality),
– if an agent X believes that a message was built by another agent Y, then the message was effectively built by Y (authentication).

4 Encoding the protocol and the intruder

In this section, we show how to model NSPK by a TRS. First, we present the signature $\mathcal{F}$ and the terms of $\mathcal{T}(\mathcal{F})$ used for representing agents, messages, keys, etc. Each agent is labeled by a unique identifier, let L_{agt} be the set of agent labels (terms representing agent labels will be given later). For any agent label $l \in L_{agt}$, the term $agt(l)$ will denote the agent whose label is l. The term $mesg(x, y, c)$ will represent a message whose header refers agent x as emitter, agent y as receiver and whose contents is c. The term $pubkey(a)$ denotes the public key of agent a and $encr(k, a, c)$ denotes the result of encryption of content c by key k. In this last term, a is a flag recording who has performed the encryption. This field is not used by the protocol rules but will be used for verification. The term $N(x, y)$ represents a nonce generated by agent x for identifying a communication with y. We also use an AC binary symbol $\sqcup$ in order to represent sets. For example the term $x \sqcup (y \sqcup z)$ (equivalent modulo AC to $(x \sqcup y) \sqcup z$) will represent the set $\{x, y, z\}$.

Starting from a set of initial requests, our aim is to compute a tree automaton recognizing an over-approximation of all sent messages. The approximation also contains some terms signaling either communication requests or established communications. For example, a term of the form $goal(x, y)$ means that x expect to open a communication with y. A term of the form $c\_init(x, y, z)$ means that x believes to have initiated a communication with y, but, in reality x communicates with z. Conversely, a term $c\_resp(y, x, z)$ means that y believes to have responded to a communication request coming from x but z is the real author of the request.

Then, encoding of the protocol into AC rewrite rules[2] is straightforward: each step of the protocol is described thanks to a rewrite rule whose left-hand

[2] We describe here our encoding in a general way. However, for the particular case of NSPK, encoding could have be done without the AC-symbol $\sqcup$, since $\sqcup$ is only needed when the sending of a message depends on the reception of two (or more) distinct messages, i.e. rules of the form: $m_1 \sqcup m_2 \rightarrow m_3$. In general, those rules are necessary to modelize protocols, but it is not the case for this simple version of NSPK.

side is a precondition on the current state (set of received messages and communication requests), and the right-hand side represents the message to be sent (and sometimes established communication) if the precondition is met. The sent message is added to the current state. As a result, every rewrite rule we use is a 'cumulative rule', i.e. of the form $l \to l \sqcup X$. Thus, for commodity, we choose to use the short-hand LHS for the term l occurring in the right-hand side. For instance, the rule $mesg(x,y,c) \to LHS \sqcup c\_init(x,y,y)$ will represent the rule: $mesg(x,y,c) \to mesg(x,y,c) \sqcup c\_init(x,y,y)$. Now for each step of the protocol, we give the corresponding rewrite rule. The encoding into TRS is longer than the initial protocol specification of section 3 because it is more complete. For instance, whereas the initial specification only informally define how to check the content of messages and how to deal with communication requests, these points are formally defined in our specification with rewrite rules. Furthermore, the initial specification can be viewed as a trace of a correct execution of the NSPK protocol for two specific agents A and B. Thus, this specification cannot be directly used in a more general context where some other agents also use the protocol. Hence, another difference between our specification and the initial specification of section 3 is that agents' identities of initial specification (A and B) have been abstracted by term with variables of the form $agt(x)$, $agt(y)$. In the following, x, y, z, u, v, $x2$, $x3$ and $z2$ are supposed to be variables since we consider an unbounded number of agents and transactions.

1. $A \hookrightarrow B : \{N_A, A\}_{K_B}$. The emission of the first message is encoded by the rule:

   $goal(x,y) \to LHS \sqcup mesg(x,y,encr(pubkey(y),x,[\![N(x,y),x]\!]))$

   The meaning of this rule is the following: if an agent x wants to establish a communication with y then x sends a message to y whose contents is encrypted with public key of y. The contents is here represented by a list (build with classical operators *cons* and *null*) containing a nonce $N(x,y)$ produced by x for y as well as x's identity. For commodity, lists will be represented in the usual way, for example a list of the form $cons(u,cons(v,null))$ will be denoted by $[\![u,v]\!]$.

2. $B \hookrightarrow A : \{N_A, N_B, B\}_{K_A}$.

   $mesg(x,agt(u),encr(pubkey(agt(u)),z,[\![v,agt(x2)]\!])) \to$
   $LHS \sqcup mesg(agt(u),agt(x2),encr(pubkey(agt(x2)),agt(u),$
   $[\![v,N(agt(u),agt(x2)),agt(u)]\!]))$

   The second message is sent by an agent $agt(u)$ when he receives the first message from an agent $agt(x2)$ whose identity is enclosed in the message[3]. Note that in those rules, we achieve some kind of type checking on the content of the message. For instance, in the left-hand side of this rule, by expecting

[3] In this protocol, agent's identity contained in the header of the message (x in our example) is never used, since it may have been corrupted by an intruder. However, this information is sometimes used, for example in the extended version of NSPK where a key server is also involved.

the message content pattern $[\![v, agt(x2)]\!]$ instead of a more general pattern like $[\![v, x3]\!]$, we check that this element of the message is an agent's identity. The role of this kind of type checking is important since it permits to avoid some attacks based on type confusion like those described in [17].

3. $A \hookrightarrow B : \{N_B\}_{K_B}$. This step is encoded by the rule:
$mesg(x, agt(y), encr(pubkey(agt(y)), z2, [\![N(agt(y), agt(z)), u, agt(z)]\!])) \rightarrow$
$LHS \sqcup mesg(agt(y), agt(z), encr(pubkey(agt(z)), [\![u]\!]))$
$\quad \sqcup\, c\_init(agt(y), agt(z), z2)$
When agent $agt(y)$ receives from $agt(z)$ the nonce $N(agt(y), agt(z))$ he has built for $agt(z)$ then he performs two actions. The first action is to send the last protocol message to $agt(z)$. The second action consists in reporting the communication $agt(y)$ thinks to have established with $agt(z)$. However, the reality may be different and the identity of the real author of the message, $z2$, is used for filling the third field of the c_init term.

4. In the last step of the protocol, no message is sent but when an agent receives the last message of the protocol sent at step 3., he reports a communication where he has the responder role.
$mesg(x, agt(y), encr(pubkey(agt(y)), z2, [\![N(agt(y), z)]\!])) \rightarrow$
$LHS \ \sqcup \ c\_resp(agt(y), z, z2)$

To prove the authentication property on the protocol, we need to prove that any couple of agents can securely establish a communication through the network, whatever the behavior of other agents and the behavior of an intruder may be. Thus, we assume that there is an unbounded number of agent labels in L_{agt} but we will observe more precisely two agents, namely agents labeled by A and B. For the unbounded number of other agent labels we will use integers built on usual operators 0 and s (successor). Hence, $L_{agt} = \{A, B\} \cup \mathbb{N}$ and the initial set of terms E is the set of terms of the form $goal(agt(x), agt(y))$ where $x, y \in L_{agt}$. In other words, E is the set of all communication requests

- from A or B towards any other agent $agt(i)$ with $i \in \mathbb{N}$, and
- from $agt(i)$ with $i \in \mathbb{N}$ towards A or B, and
- from any agent $agt(i)$ to any agent $agt(j)$, $i, j \in \mathbb{N}$, and
- from A to B, B to A, A to A and B to B.

Note that we work in a very general setting where we also take into account the case where an agent use the protocol to authenticate himself. It is clear that self-authentication of an agent may be not of practical interest, but, if it happens we want to verify that the intruder cannot take advantage of it to build an attack. The set E is recognized by the following tree automaton $\mathcal{A}_0$. The final state of $\mathcal{A}_0$ is q_{net} and here is the set of transitions:

$$
\begin{array}{rrr}
0 \rightarrow q_{int} & agt(q_B) \rightarrow q_{agtB} & goal(q_{agtA}, q_{agtI}) \rightarrow q_{net} \\
s(q_{int}) \rightarrow q_{int} & q_{net} \sqcup q_{net} \rightarrow q_{net} & goal(q_{agtI}, q_{agtA}) \rightarrow q_{net} \\
A \rightarrow q_A & goal(q_{agtA}, q_{agtB}) \rightarrow q_{net} & goal(q_{agtB}, q_{agtI}) \rightarrow q_{net} \\
B \rightarrow q_B & goal(q_{agtB}, q_{agtA}) \rightarrow q_{net} & goal(q_{agtI}, q_{agtB}) \rightarrow q_{net} \\
agt(q_{int}) \rightarrow q_{agtI} & goal(q_{agtA}, q_{agtA}) \rightarrow q_{net} & goal(q_{agtI}, q_{agtI}) \rightarrow q_{net} \\
agt(q_A) \rightarrow q_{agtA} & goal(q_{agtB}, q_{agtB}) \rightarrow q_{net} &
\end{array}
$$

Description of the intruder

In this last automaton, the state q_{net} is a special state representing both the network and the fact base containing communication requests and communication reports. As in many other verification approach of cryptographic protocols, the intruder is supposed to have a total control on the network. In particular, the intruder is assumed to know every message sent on the network. In our approach this assumption is a bit stronger: the intruder *is* the network. A direct consequence of this choice is that the knowledge of the intruder and every message that the intruder can build is supposed to always remain on the network. Furthermore, we suppose that agents $agt(i)$ with $i \in \mathbb{N}$ (i.e. every agent that is not A or B) may be dishonest and deliberately give to the intruder their private key as well as the content of any message they send or receive. The intruder can also disassemble messages or build new ones from his knowledge. Rewrite rules are the simplest way to describe how an intruder can decrypt or disassemble components of a message. Since the agents $agt(i)$ with $i \in \mathbb{N}$ are fool enough to give their private keys to the intruder, he can decrypt the messages encrypted with their public keys. On the opposite, we assume that the intruder has no means of guessing the private key of A or B. Here are the corresponding rules which can be applied on the AC-term representing the network, i.e. the intruder knowledge:

$$
\begin{array}{ll}
cons(x,y) \sqcup z \to LHS \sqcup x & /*\ \text{Disassembling}\ */ \\
cons(x,y) \sqcup z \to LHS \sqcup y & \\
mesg(x,y,z) \sqcup u \to LHS \sqcup z & \\
encr(pubkey(agt(0)),y,z) \sqcup u \to LHS \sqcup z & /*\ \text{Decrypting}\ */ \\
encr(pubkey(agt(s(x))),y,z) \sqcup u \to LHS \sqcup z &
\end{array}
$$

On the other hand, intruder's ability to build new messages from its knowledge is shortly defined thanks to some tree automaton transitions. Since q_{net} is the state of $\mathcal{A}_0$ recognizing all the messages on the network, and since in our setting the knowledge of the intruder *is* the network, q_{net} is also the state recognizing the knowledge of the intruder. First, we assume that the intruder knows the identity of every agent of the network, as well as their public keys.

$$
\begin{array}{rrr}
agt(q_{int}) \to q_{net} & agt(q_A) \to q_{net} & agt(q_B) \to q_{net} \\
pubkey(q_{agtI}) \to q_{net} & pubkey(q_{agtA}) \to q_{net} & pubkey(q_{agtB}) \to q_{net}
\end{array}
$$

Agents $agt(i)$ with $i \in \mathbb{N}$ give the intruder the nonces they produce for other agents:

$$
N(q_{agtI}, q_{agtA}) \to q_{net} \qquad N(q_{agtI}, q_{agtB}) \to q_{net} \qquad N(q_{agtI}, q_{agtI}) \to q_{net}
$$

Finally, starting from components he already knows or will obtain later (i.e. terms in q_{net}), the intruder can combine them into lists with the *cons* operator, encrypt them with anything (including keys) he knows with operator *encr*, build messages with operator *mesg*, etc. in order to enrich his knowledge (the language recognized by q_{net}). Note, however, that the second field of the operator *encr* (which is a flag) cannot be corrupted by the intruder and always refer to q_{agtI} the real author of the encryption, i.e. the intruder.

$$cons(q_{net}, q_{net}) \to q_{net} \qquad null \to q_{net} \qquad encr(q_{net}, q_{agtI}, q_{net}) \to q_{net}$$
$$mesg(q_{net}, q_{net}, q_{net}) \to q_{net}$$

There are several things to notice here. First, the initial description of $\mathcal{L}(\mathcal{A}_0, q_{net})$ is as wide and loose as possible: roughly, it authorizes the intruder to build nearly every term of $\mathcal{T}(\mathcal{F})$ except terms containing nonces built by A or B, i.e. terms containing subterms of the form $N(agt(A), agt(x))$ or $N(agt(B), agt(y))$. This can be automatically obtained by a complement operation. This kind of specification is quite natural with regards to intruder description since it is much more simpler and more convincing to specify what cannot be built by the intruder than to precisely and totally define what he can do. Consequently, the language recognized by state q_{net} is loose and it may also contain strangely formed messages whose effect on the protocol can hardly be predicted, for example:

$$mesg(agt(A), agt(B), encr(pubkey(agt(B)), agt(0), [\![encr(pubkey(agt(A)), agt(0), [\![N(agt(0), agt(A))]\!]), N(agt(0), agt(B))]\!]))$$

i.e. a message of the form $agt(A) \hookrightarrow agt(B) : \{\{N_{agt(0)}\}_{K_{agt(A)}}, N_{agt(0)}\}_{K_{agt(B)}}$. The language recognized by q_{net} contains also, for instance, terms representing repeated encryption (an unbound number) which are important to consider for cryptographic protocols verification:

$$encr(pubkey(agt(A)), agt(s(0)), encr(pubkey(agt(B)), agt(0), encr(\ldots$$

The last thing to remark here is that during approximation construction, new messages or messages components m obtained by rewriting are added to the language recognized by automaton $\mathcal{A}_i$ as new transitions into $\mathcal{A}_{i+1}$ s.t. $m \to^*_{\mathcal{A}_{i+1}} q_{net}$ and thus can be used 'dynamically' as new base components for intruder's message constructions.

To sum up, we have here described a model where we consider an unbounded number of agents executing an unbounded number of protocol sessions in parallel. In particular, note that if there exists an attack based on parallel protocol sessions between, say four agents A, B, C and D, this attack will appear in the model: C and D can be represented by two 'dishonest' agents, say $agt(i)$ and $agt(j)$ with $i, j \in \mathbb{N}$ and $i \neq j$ since all 'dishonest' agents are able to respect the protocol.

5 Approximation and verification

Extensions of approximations to AC non left-linear TRSs

In this section, we show how to extend the approximation construction to this larger class of TRSs. Roughly, the problem with non left-linear rules is the following: let $f(x, x) \to g(x)$ be a rule of $\mathcal{R}$ and let $\mathcal{A}$ be a tree automaton whose set of transitions contains $f(q_1, q_1) \to q_0$ and $f(q_2, q_3) \to q_0$. Although we can construct a valid substitution $\sigma = \{x \mapsto q_1\}$ for matching the rewrite rule on the first transition, it is not the case for the second one. The semantics of a completion between rule $f(x, x) \to g(x)$ and transition $f(q_2, q_3) \to q_0$ would be to find the common language of terms recognized both by q_2 and q_3. This can be obtained by computing a new tree automaton $\mathcal{A}'$ with a set of states $\mathcal{Q}'$ such that $\mathcal{Q}'$ is disjoint from states of $\mathcal{A}$ and $\exists q \in \mathcal{Q}' : \mathcal{L}(\mathcal{A}', q) = \mathcal{L}(\mathcal{A}, q_2) \cap \mathcal{L}(\mathcal{A}, q_3)$. Then, to end the completion step it would be enough to add transitions of $\mathcal{A}'$ to $\mathcal{A}$ with the new transition $g(q) \to q_0$. However, adding transitions of $\mathcal{A}'$ to $\mathcal{A}$ also adds $\mathcal{Q}'$ to states of $\mathcal{A}$. Thus, we add new states to $\mathcal{A}$ and, in some cases, this may lead to non-termination of the approximation construction.

On the other hand, one can remark that the non-linearity problem would disappear with deterministic automata since for any deterministic automaton $\mathcal{A}_{det}$ and for all states q, q' of $\mathcal{A}_{det}$ we trivially have $\mathcal{L}(\mathcal{A}, q) \cap \mathcal{L}(\mathcal{A}, q') = \emptyset$. However, determinization of a tree automaton may result in an exponential blow-up of the number of states [3]. Thus, we chose here to use *locally deterministic* tree automata: non-deterministic tree automata with some *deterministic states*, i.e. states q such that there is no two rules $t \to q$ and $t \to q'$ with $q \neq q'$. Hence, for all deterministic state q, we have $\forall q' \neq q : \mathcal{L}(\mathcal{A}, q) \cap \mathcal{L}(\mathcal{A}, q') = \emptyset$. During the approximation construction, if all states, matched by a non-linear variable of the left-hand side of a rule, are deterministic then it is enough to build critical pairs where non linear variables of the left-hand side are mapped to the same state. For instance, in the last example, it is enough to build the first critical pair, add the transition $g(q_1) \to q_0$, and keep q_2, q_3 deterministic, i.e. such that $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A}), q_2) \cap \mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A}), q_3) = \emptyset$. We now show the completeness of this algorithm on locally deterministic tree automata.

For all term t non linear, let us denote by t_{lin} the term t linearized, i.e. where all occurrences of non linear variables are replaced by disjoint variables. For example, if $t = f(x, y, g(x, x))$, then $t_{lin} = f(x', y, g(x'', x'''))$.

Definition 4. *(States matching) Let $\mathcal{A}$ be a tree automaton, $\mathcal{Q}$ its set of states, $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ a non linear term, and $\{p_1, \ldots, p_n\} \subseteq \mathcal{P}os(t)$ the set of positions of a non linear variable x in t. We say that states $q_1, \ldots, q_n \in \mathcal{Q}$ are* matched *by x iff $\exists \sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ s.t. $t_{lin}\sigma \to_{\mathcal{A}}^* q \in \mathcal{Q}$, and $t_{lin}\sigma|_{p_1} = q_1, \ldots, t_{lin}\sigma|_{p_n} = q_n$*

Theorem 2. *(Completeness extended to non left-linear TRS) Let $\mathcal{A}$ be a tree automaton, $\mathcal{R}$ a TRS, $\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A})$ the corresponding approximation automaton and $\mathcal{Q}$ its set of states. For all non left-linear rule $l \to r \in \mathcal{R}$, for all non linear variable x of l, for all states $q_1, \ldots, q_n \in \mathcal{Q}$ matched by x, if either $q_1 = \ldots = q_n$ or $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A}), q_1) \cap \ldots \cap \mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A}), q_n) = \emptyset$ then*

$$\mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A})) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$$

Proof. (Sketch)(See [10] for a detailed proof) Assume that there exists a term t such that $t \in \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ and $t \notin \mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A}))$. Let $s \in \mathcal{L}(\mathcal{A})$ such that $s \to_{\mathcal{R}}^* t$. On the rewrite chain from s to t, let t_1, t_2 be the first two terms such that $t_1 \in \mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(A))$, $t_1 \to_{\mathcal{R}} t_2$ and $t_2 \notin \mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(A))$. We then show that the rule

$l \to r \in \mathcal{R}$ applied for rewriting t_1 into t_2 is necessarily a non left-linear rule (otherwise t_2 would be in $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A}))$). Then, we obtain that there exists a subterm u of t_1 matched by all occurrences of a non linear variable x in l and there exists at least two distinct states q, q' of $\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A})$ such that $u \to^*_{\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})} q$ and $u \to^*_{\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})} q'$. This contradicts the hypothesis of the theorem since q and q' are matched by x in l and $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A}), q) \cap \mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A}), q') \supseteq \{u\} \neq \emptyset$. □

In our framework, states matched by non-linear variables are easily kept deterministic. For example, in the NSPK specification, non-linear variables always match terms A, B, $i \in \mathbb{N}$ (representing agent labels) which are initially recognized by q_A, q_B, q_{int}, respectively. Those states are initially deterministic and this property is trivially preserved during completion since agent labels do not occur in right-hand side of rules and thus agent labels do not occur in new transitions to be added. However, when necessary, we can also automatically check this property on $\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A})$ by proving that $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A}), q_1) \cap \ldots \cap \mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A}), q_n) = \emptyset$, for each non linear variable x of a rule matching distinct states $q_1, \ldots, q_n$.

For dealing with the AC symbols, the extension is straightforward. Since approximation can deal with non terminating TRS, we can explicitly define the AC-behavior of a symbol. Thus, we replace in $\mathcal{F}$ the (implicit) AC-symbol $\sqcup$ by a non-AC symbol U and add to $\mathcal{R}$ the following left-linear rules defining explicitly the AC behavior of U:

$$x \mathrel{\mathrm{U}} y \to y \mathrel{\mathrm{U}} x \qquad (x \mathrel{\mathrm{U}} y) \mathrel{\mathrm{U}} z \to x \mathrel{\mathrm{U}} (y \mathrel{\mathrm{U}} z) \qquad x \mathrel{\mathrm{U}} (y \mathrel{\mathrm{U}} z) \to (x \mathrel{\mathrm{U}} y) \mathrel{\mathrm{U}} z$$

Approximation function

Let $\mathcal{R}$ and $\mathcal{A}_0$ be respectively the set of all rewrite rules and the tree automaton given above. Our aim is now to compute a tree automaton $\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A}_0)$ recognizing a superset of $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$ and thus, to over-approximate the network, i.e. the set of all possible sent messages (as well as the set of communication reports). We now give the approximation function γ, defining the folding positions for $\mathcal{R}$ and $\mathcal{A}_0$. For approximation, the first choice we have made is to confuse dishonest agents ($agt(i)$ with $i \in \mathbb{N}$) together. In other words, in our approximation, no difference is made between agents $agt(i)$ and $agt(j)$ for any $i, j \in \mathbb{N}$. However, we still distinguish between $agt(A)$, $agt(B)$ and any agent $agt(i)$ with $i \in \mathbb{N}$. In a similar manner, we collapse together all the messages sent and received by dishonest agents but we still do not confuse messages involving $agt(A)$ or $agt(B)$. For example, the approximation function used for the rule ①, i.e.

$$goal(x, y) \to LHS \mathrel{\mathrm{U}} mesg(x, y, encr(pubkey(y), x, [\![N(x, y), x]\!]))$$

is such that there are only seven distinct values for γ (The detail of sequences of new states used for each value can be found in [10] with the complete specification.):

i	$\gamma(①, q_{net}, \{x \mapsto q_{agtA}, y \mapsto q_{agtB}\})$	ii	$\gamma(①, q_{net}, \{x \mapsto q_{agtB}, y \mapsto q_{agtA}\})$
iii	$\gamma(①, q_{net}, \{x \mapsto q_{agtA}, y \mapsto q_{agtA}\})$	iv	$\gamma(①, q_{net}, \{x \mapsto q_{agtB}, y \mapsto q_{agtB}\})$
v	$\gamma(①, q_{net}, \{x \mapsto q_{agtI}, y \mapsto q_{agtA}\})$	vi	$\gamma(①, q_{net}, \{x \mapsto q_{agtI}, y \mapsto q_{agtB}\})$
vii	$\gamma(①, q_{net}, \{y \mapsto q_{agtI}\})$		

According to case (i) all messages generated thanks to rule 1, where x is the agent labeled by A and y is the agent labeled by B, are decomposed using the same states defined by the sequence $\gamma(①, q_{net}, \{x \mapsto q_{agtA}, y \mapsto q_{agtB}\})$. Similarly, the case (vii) means that all messages generated thanks to rule 1, where y is an agent labeled by $i \in \mathbb{N}$ and x is any agent, are decomposed using states of the same sequence $\gamma(①, q_{net}, \{y \mapsto q_{agtI}\})$. Thus, no difference is made, for example, between messages sent by $agt(A)$ to $agt(i)$, messages sent by $agt(B)$ to $agt(j)$, and messages sent for by $agt(i)$ to $agt(j)$ for any $i, j \in \mathbb{N}$. This is in fact natural since all messages sent to a dishonest agent are captured and factorized by the same intruder.

Verification

We use a prototype, based on a tree automata library [9, 8] developed in E-LAN [2], which permits to automatically compute approximations for a given $\mathcal{R}$, $\mathcal{A}_0$ and an approximation function γ. Thanks to the approximation function given above, we obtain a finite tree automaton $\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A}_0)$, with about 130 states and 340 transitions, recognizing a regular superset of $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$. See [10] for the complete specification and for a complete listing of the automaton $\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A}_0)$.

Thanks to this automaton, we can directly verify that NSPK has the confidentiality and authentication property. For confidentiality, it is enough to verify that the intruder cannot capture a nonce of the form $N(agt(x), agt(y))$ where $x, y \in \{A, B\}$. Since in our model the intruder emits all his knowledge on the network (as explained in section 4), this can be done by checking that the intruder cannot emit a nonce of the form $N(agt(A), agt(B))$, $N(agt(B), agt(A))$, ... i.e. that the intersection between $\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A}_0)$ and the automaton $\mathcal{A}_{conf}$ is empty. The final state of $\mathcal{A}_{conf}$ is q_{net} and its transitions are:

$$
\begin{array}{rcc}
A \to q_A & agt(q_B) \to q_{agtB} & N(q_{agtA}, q_{agtA}) \to q_{net} \\
B \to q_B & N(q_{agtA}, q_{agtB}) \to q_{net} & N(q_{agtB}, q_{agtB}) \to q_{net} \\
agt(q_A) \to q_{agtA} & N(q_{agtB}, q_{agtA}) \to q_{net} & q_{net} \ \mathrm{U}\ q_{net} \to q_{net}
\end{array}
$$

The intersection can be automatically computed and we obtain a tree automaton whose set of states is empty, i.e. the recognized language is empty. Hence, there is no term of $\mathcal{L}(\mathcal{A}_{conf})$ in $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(\mathcal{A}_0))$ nor in $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$. Similarly, the cases where authentication is corrupted can be described by the following automaton $\mathcal{A}_{aut}$ whose final state is q_{net} and transitions are:

$$
\begin{array}{rcc}
0 \to q_{int} & c\_init(q_{agtA}, q_{agtB}, q_{agtI}) \to q_{net} & c\_init(q_{agtA}, q_{agtA}, q_{agtI}) \to q_{net} \\
s(q_{int}) \to q_{int} & c\_init(q_{agtA}, q_{agtB}, q_{agtA}) \to q_{net} & c\_resp(q_{agtA}, q_{agtA}, q_{agtI}) \to q_{net} \\
A \to q_A & c\_resp(q_{agtB}, q_{agtA}, q_{agtI}) \to q_{net} & c\_init(q_{agtA}, q_{agtA}, q_{agtB}) \to q_{net} \\
B \to q_B & c\_resp(q_{agtB}, q_{agtA}, q_{agtB}) \to q_{net} & c\_resp(q_{agtA}, q_{agtA}, q_{agtB}) \to q_{net} \\
agt(q_{int}) \to q_{agtI} & c\_init(q_{agtB}, q_{agtA}, q_{agtI}) \to q_{net} & c\_init(q_{agtB}, q_{agtB}, q_{agtI}) \to q_{net} \\
agt(q_A) \to q_{agtA} & c\_init(q_{agtB}, q_{agtA}, q_{agtB}) \to q_{net} & c\_resp(q_{agtB}, q_{agtB}, q_{agtI}) \to q_{net} \\
agt(q_B) \to q_{agtB} & c\_resp(q_{agtA}, q_{agtB}, q_{agtI}) \to q_{net} & c\_init(q_{agtB}, q_{agtB}, q_{agtA}) \to q_{net} \\
q_{net} \ \mathrm{U}\ q_{net} \to q_{net} & c\_resp(q_{agtA}, q_{agtB}, q_{agtA}) \to q_{net} & c\_resp(q_{agtB}, q_{agtB}, q_{agtA}) \to q_{net}
\end{array}
$$

encoding all the cases where there is a distortion in communication reports between the belief of the parties and the reality, for example terms of the form $c\_init(agt(A),\ agt(B), agt(k))$ for $k \in \mathbb{N} \cup \{A\}$ meaning that $agt(A)$ think to have established a communication with B but, in reality, he has been fooled and he communicates with some $agt(i)$ with $i \in \mathbb{N}$ or with himself. The intersection between $\mathcal{T_R}\!\uparrow(\mathcal{A}_0)$ and the automaton $\mathcal{A}_{aut}$ is also empty (see [10] for traces of execution).

6 Conclusion

In this paper, we have shown an application of descendant approximation to cryptographic protocols verification. We have obtained a positive proof of authentication and confidentiality of NSPK. Moreover, applying the same approximation mechanism on the flawed NSPK specification of [19] has led to some non-empty intersections with $\mathcal{A}_{conf}$ and $\mathcal{A}_{aut}$, signaling violation of confidentiality and authentication properties.

An interesting aspect of this method is that it takes advantage of theorem proving and a form of abstract interpretation called approximation. The basic deduction mechanism, coming from the domain of theorem proving, provide some simple and efficient tools – tree automata – to manipulate infinite objects. On the other hand, approximation simplifies the proof in such a way that it can be automatically computed afterwards.

Compared to other rewriting based verification techniques like proofs by consistency or proofs by induction, properties that can be proved with the approximation technique are clearly more restricted: they could be qualified as 'regular properties'. However, by restricting attention to 'regular properties', we obtain a verification technique that enjoys many interesting practical properties: termination of the TRS is not needed, TRS may include AC symbols, proofs are obtained by intersections with $\mathcal{T_R}\!\uparrow(\mathcal{A}_0)$ (automatically and quickly computed), construction of $\mathcal{T_R}\!\uparrow(\mathcal{A}_0)$ is automatic, incremental and can be guaranteed to terminate by a good choice of the γ approximation function (like in the NSPK case above or in a fully automatic way like in [9]). Constructing an approximation function does not require any particular skill in formal proof since it only consists in pointing out some sets of objects (represented here by states recognizing regular sets of terms) to be merged together in order to build an approximated model. In the NSPK case, the γ approximation has been entirely given by hand but it is systematic: for each distinct value of the co-domain of γ the user has to give a sequence of fresh states used for normalizing new transitions. For historical reasons, this step is manual in our prototype but will be automated in the new implementation of this tool which is in progress.

We can also compare this technique with other verification techniques used for verifying cryptographic protocols. The first main difference to be pointed out is that our technique is not designed for discovering attacks. From approximation $\mathcal{T_R}\!\uparrow(\mathcal{A}_0)$, we can derive some information on the context of those attacks but it is approximate and should be studied with a theorem prover or a model-checker to re-construct an exact trace of the attack. Model-checking is, in fact, particularly well suited for attack discovery as showed by the many flaws discovered by G. Lowe [15]. Furthermore, when attacks are no longer found, model-checking can also be used to verify cryptographic protocols by lifting the properties proved on a finite domain to an unbounded. However, the lifting has to be done by hand like G. Lowe did in [14] or, in a more automatic way, by abstract interpretation like it is done by D. Bolignano in [1]. Although we started with a different formalism and used a different technique, our approach is very close to D. Bolignano's one. In particular, approximation functions can be seen as particular abstract interpretations. Nevertheless, approximations enjoys a property that abstract interpretations have not in general: safety of our abstract model (approximations) is implicit and guaranteed by Theorem 1 for every approximation function γ.

Automated theorem proving has also been widely used for cryptographic protocols verification. The NRL Protocol Analyzer, developed by C. Meadows [17], uses narrowing. L. Paulson applied induction proof and the theorem prover Isabelle/HOL to the verification of cryptographic protocols [20]. Those two theorem proving approaches achieve a very detailed verification of protocols. In particular, they provide one of the most convincing answer to the problem of freshness. In counterpart, the proofs may diverge and the main difficulty remain to inject the right lemma at the right moment in order to make the proof converge. Thus, automation of this kind of method remains partial. Furthermore, proofs are long, complex and they require a user with a strong practical experience of the prover. A more recent work is due to C. Weidenbach [22] who gave a positive proof for the Neuman-Stubblebine protocol thanks to the theorem prover SPASS. His technique is based on saturation of sets of horn clauses, which is related to the descendant computation we here use. For a restricted class of clauses called semi-linear, saturation can be computed exactly. However, when the protocol specification cannot be encoded into semi-linear clauses the saturation process may diverge. Thus, specifications must be modified in order to ensure termination of the process. In our framework, no restriction is set on the TRSs we use but, instead, we defined an over-approximation technique in order to tackle the divergence problem.

In [6], G. Denker, J. Meseguer and C. Talcott proposed to encode the NSPK into object-oriented TRSs. This encoding is executable and is used for detecting attacks in the initial version of the protocol by testing. Using objects is clearly a great advantage for a better clarity and readability of the encoding. Nevertheless, since rewriting remains the operational model of object oriented rewriting, it should be possible to extend approximations to objects and thus benefit of the clarity of object oriented specifications.

In [18], D. Monniaux also use tree automata and a completion mechanism for verifying cryptographic protocols. With regards to our work, an important difference is that his method can only deal with a bounded number of agents and a bounded number of protocol sessions. On a more technical point of view, unlike our approach, rewriting is only used for estimating intruder knowledge and not for encoding the protocol itself. Moreover, his completion mechanism is limited to the decidable and well known case of collapsing rules[4] covered by the decidable and more general case of right-linear and monadic rules [21]. However, this approach is interesting since it shows a possible way for combining tree automata and state-transition models for abstract interpretation of protocols: tree automata and completion for abstracting structures and state-transition models for representing the notion of time in the abstract model.

In the approximation model we consider for NSPK, time is totally collapsed, i.e. every message is considered to be permanently sent and received at every moment. Collapsing time let us easily consider an infinite number of protocol sessions in a finite model. Although this does not raise problems for proving confidentiality or authentication properties on NSPK, this is not the case in general. For instance, in electronic commerce protocols like SET [16], there is little hope to prove any security property on an abstract model with no time since freshness plays a central role. A direct solution is to consider several states for the network (i.e. of intruder knowledge) for different steps of the protocol instead of collapsing all states in one.

Our main goal is to be able to handle protocol as complex as SET. To achieve this goal, the first thing to consider is to formally define the concepts used in cryptographic protocols (keys, nonces, agents, ...) in order to get a natural protocol language description and an automatic translator to the encoding presented in this paper. The second point would be, on the one hand, to extend the present work with conditional rules in order to get a more powerful behavior description language and, on the other hand, to handle other tree grammars to get finer approximations.

Finally, we think that approximations could be used for the verification of systems different from cryptographic protocols. Rewriting based approximations seems to be a way to combine, in the same formalism, automated theorem proving techniques and abstract interpretation: theorem proving for proving properties needing high level proof techniques – like induction – and approximations for proving the remaining parts of the proof where abstract interpretation and model-checking are enough.

Acknowledgments

We would like to thank Pascal Brisset for discussion about cryptographic protocols and Pierre-Etienne Moreau for technical help with ELAN.

References

1. D. Bolignano. Towards a Mechanization of Cryptographic Protocol Verification. In *Proc. 9th CAV Conf., Haifa (Israel)*, volume 1254 of *LNCS*. Springer-Verlag, 1997.

[4] right-hand side of a collapsing rule is reducted to a variable occurring in its left-hand side.

2. P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and M. Vittek. ELAN: A logical framework based on computational systems. In *Proc. 1st WRLA*, volume 4 of *ENTCS*, Asilomar (California), 1996.
3. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. `http://l3ux02.univ-lille3.fr/tata/`, 1997.
4. J. Coquidé, M. Dauchet, R. Gilleron, and S. Vágvölgyi. Bottom-up tree pushdown automata and rewrite systems. In R. V. Book, editor, *Proc. 4th RTA Conf., Como (Italy)*, volume 488 of *LNCS*, pages 287–298. Springer-Verlag, 1991.
5. M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proc. 5th LICS Symp., Philadelphia (Pa., USA)*, pages 242–248, June 1990.
6. G. Denker, J. Meseguer, and C. Talcott. Protocol Specification and Analysis in Maude. In *Proc. 2nd WRLA Workshop, Pont à Mousson (France)*, 1998.
7. N. Dershowitz and J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter 6: Rewrite Systems, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990. Also as: Research report 478, LRI.
8. T. Genet. Tree Automata Library. `http://www.loria.fr/ELAN/`.
9. T. Genet. Decidable approximations of sets of descendants and sets of normal forms. In *Proc. 9th RTA Conf., Tsukuba (Japan)*, volume 1379 of *LNCS*, pages 151–165. Springer-Verlag, 1998.
10. T. Genet and F. Klay. Rewriting for cryptographic protocols verification (extended version). Technical report, INRIA, 2000. `http://www.irisa.fr/lande/genet/publications.html`.
11. R. Gilleron and S. Tison. Regular tree languages and rewrite systems. *Fundamenta Informaticae*, 24:157–175, 1995.
12. F. Jacquemard. Decidable approximations of term rewriting systems. In H. Ganzinger, editor, *Proc. 7th RTA Conf., New Brunswick (New Jersey, USA)*, pages 362–376. Springer-Verlag, 1996.
13. G. Lowe. An Attack on the Needham-Schroder Public-Key Protocol. *IPL*, 56:131–133, 1995.
14. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In *Proc. 2nd TACAS Conf., Passau (Germany)*, volume 1055 of *LNCS*, pages 147–166. Springer-Verlag, 1996.
15. G. Lowe. Some New Attacks upon Security Protocols. In *9th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1996.
16. Mastercard & Visa. Secure Electronic Transactions. `http://www.visa.com/set/`, 1996.
17. C. A. Meadows. Analyzing the Needham-Schroeder Public Key Protocol: A comparison of two approaches. In *Proc. 4th ESORICS Symp., Rome (Italy)*, volume 1146 of *LNCS*, pages 351–364. Springer-Verlag, 1996.
18. D. Monniaux. Abstracting Cryptographic Protocols with Tree Automata. In *Proc. 6th SAS, Venezia (Italy)*, 1999.
19. R. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *CACM*, 21(12):993–999, 1978.
20. L. Paulson. Proving Properties of Security Protocols by Induction. In *10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.
21. K. Salomaa. Deterministic Tree Pushdown Automata and Monadic Tree Rewriting Systems. *J. of Computer and System Sciences*, 37:367–394, 1988.
22. C. Weidenbach. Towards an Automatic Analysis of Security Protocols. In *Proc. 16th CADE Conf., Trento, (Italy)*, volume 1632 of *LNAI*, pages 378–382. Springer-Verlag, 1999.