Adel Bouhoula and
Florent Jacquemard

Tree Automata, Implicit Induction and Explicit Destructors for Security Protocol Verification

Research Report LSV-07-10

February 2007

Tree Automata, Implicit Induction and Explicit Destructors for Security Protocol Verification [1]

Adel Bouhoula & Florent Jacquemard [2,3]

École Supérieure des Communications de Tunis, Tunisia.

INRIA Futurs & LSV UMR CNRS–ENSC, France.

Abstract

We present a new method for automatic implicit induction theorem proving, and its application for the verification of cryptographic protocols. The method is based on constrained tree grammars and handles non-confluent rewrite systems which are required in the context of the verification of security protocols because of the non-deterministic behavior of attackers. It also handles axioms between constructor terms which allows us to specify explicit destructors representing cryptographic operators. Constrained tree grammars are used in our procedure both as induction schemes and as oracles for checking validity and redundancy by reduction to an emptiness problem. They also permit to characterize security failure of cryptographic protocols as sets of execution traces corresponding to an attack. This way, we obtain a generic framework for the verification of protocols, in which we can verify reachability properties like confidentiality, but also more complex properties like authentication. We present three case studies which gave very promising results.

Key words: Security Protocol Verification, Inductive Theorem Proving, Conditional and Constrained Term Rewriting, Tree Automata.

1 Introduction

Inductive theorem proving techniques and tools have been successfully applied in last years to the verification of security protocols, both for proving security properties and for identifying attacks on faulty protocols.

Paulson's inductive approach [12] for the verification of cryptographic protocols has been applied to several case studies during the past years. In

[1] This work has been partially supported by the grant INRIA–DGRSRT 06/I09, the RNTL project Prouvé, and the ACI "Sécurité Informatique" Rossignol.
[2] Email: `bouhoula@planet.tn`
[3] Email: `florent.jacquemard@inria.fr`

this method, protocols are formalized in typed higher-order logic and the Isabelle/HOL interactive theorem prover is used to prove security properties. Paulson's technique handles infinite state protocols and does not assume any restriction on the number of protocol participants. However, it is not automatic and requires interaction with the user and a good expertise (if a proof fails with Isabelle, it is difficult to conclude whether the proof attempt fails or the conjecture to be proved is not valid).

Bundy and Steel [5] derive attacks on faulty protocols specified in first-order logic using a *proof by consistency* technique. Such a technique is sometimes also called *inductionless induction* [8] since it does not construct an induction proof following an induction schema but rather tries to automatically derive an inconsistency using first-order theorem proving techniques. This technique is hence fully automatic but its outcome may be difficult to analyze.

In this paper we present a new method for the formal verification security protocols with an extension of an *implicit induction* procedure [3]. The advantage of this procedure is that it is automatic and returns readable proofs or counter-examples. There are two main novelties with our procedure, compared to other implicit induction techniques. First, it handles specification which are not confluent. The property of ground confluence (any two divergent reduction sequences starting from the same ground term converge ultimately) is usually required for induction procedures. For the application to protocol verification, we consider a model with an active attacker which interferes non-deterministically with the communications of honest agents. Such a model is interesting in the context of security because it corresponds to minimal security assumptions concerning the environment (the smallest the security assumptions are, the best the commitment of a security certification is), but it can not be expressed with a ground-confluent specification. Second, our procedure also handles axioms between constructors. Such equational axioms are used in our settings in order to specify cryptographic operators like decryption. This approach with *explicit destructors* is the base of a uniform framework for the verification of security protocols in an insecure communication environment [1]. Models with explicit destructors are strictly more expressive than models based on free algebra, in the sense that they captures more attacks [11].

In contrast to the technique of [5], implicit induction is a goal directed proof technique, and we believe that it is therefore quite efficient for automatically finding attacks on faulty protocols. The use of tree automata techniques in [3] permits in particular to focus on traces of events in normal form, and consequently to minimize the set of traces to be checked. Moreover, since our procedure is refutationally complete (under some conditions for the specification) its application on any flawed protocol will return an attack in finite (and typically very small) time and in a completely automatic way.

We present in this paper three examples of protocols analyzed with our method. The first two examples (Sections 4 and 5) are flawed protocol and our procedure permits to recover some attacks from the readable counter-example generated. The last example (Section 6) is a an amended version of the protocol of Section 4 and we show how our method can help in its validation.

2 Preliminaries

The reader is assumed familiar with the basic notions of term rewriting [10].

Signatures and terms

Let $\mathcal{F}$ be many-sorted signature partitioned into $\mathcal{F} = \mathcal{C} \uplus \mathcal{D}$, where $\mathcal{C}$ is a set of *constructor symbols*, and $\mathcal{D}$ is a set of *defined symbols*. The set of well-sorted terms over $\mathcal{F}$ (resp. constructor well-sorted terms) with variables in $\mathcal{X}$ will be denoted by $\mathcal{T}(\mathcal{F}, \mathcal{X})$ (resp. $\mathcal{T}(\mathcal{C}, \mathcal{X})$) and the subsets of variable-free terms, or *ground* terms, $\mathcal{T}(\mathcal{F})$ (resp. $\mathcal{T}(\mathcal{C})$). A term t is identified as usual to a function from its set of *positions* (strings of positive integers) $\mathcal{P}os(t)$ to symbols of $\mathcal{F}$ and $\mathcal{X}$. The *subterm* of t at position p is denoted by $t|_p$. The result of replacing $t|_p$ with s at position p in t is denoted by $t[s]_p$. This notation is also used to indicate that s is a subterm of t, in which case p may be omitted. A term t is *linear* if every variable occurs at most once in t. A *substitution* is a finite mapping from variables to terms, extended as usual as a morphism from terms to terms, written in postfix notation.

Constraints for terms and clauses

We assume given a constraint language $\mathcal{L}$, which is a finite set of predicate symbols with a recursive Boolean interpretation in $\mathcal{T}(\mathcal{C})$. Typically, $\mathcal{L}$ contains the syntactic equality and disequality $\approx$ and $\not\approx$ and membership $x{:}L$ to a fixed regular tree language $L \subseteq \mathcal{T}(\mathcal{C})$. *Constraints* on the language $\mathcal{L}$ are Boolean combinations of atoms of the form $P(t_1, \ldots, t_n)$ where $P \in \mathcal{L}$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C}, \mathcal{X})$. An empty combination is interpreted as true. A *solution* of a constraint c is a (constructor) substitution σ grounding for all terms in c and such that $c\sigma$ is interpreted to true[4]. The set of solutions of the constraint c is denoted by $sol(c)$.

A *constrained term* $t\,[\![c]\!]$ is a linear term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ together with a constraint c, which may share some variables with t. Note that the assumption that t is linear is not restrictive, since any non-linearity may be expressed in the constraint, for instance $f(x, x)\,[\![c]\!]$ is semantically equivalent to $f(x, x')\,[\![c \wedge x \approx x']\!]$. A *literal* is an equation $s = t$ or an oriented equation $s \to t$ between two terms. We consider *clauses* of the form $\Gamma \Rightarrow L$ where Γ is a conjunction of literals and L is a literal. We find convenient to see clauses

[4] The extension of application of substitutions from terms to constraints is straightforward.

themselves as terms on a signature extended by the predicate symbols $=$ and $\rightarrow$, and the connective $\wedge$ and $\Rightarrow$. This way, we can define a *constrained clause* as a constrained term. A constrained clause $C\,[\![c]\!]$ is said to *subsume* a constrained clause $C'\,[\![c']\!]$ if there is a substitution σ such that $C\sigma$ is a sub-clause of C' and $c\sigma \wedge \neg c'$ is unsatisfiable.

Conditional constrained rewriting

A *conditional constrained rewrite rule* is a constrained clause of the form $u_1 = v_1, \ldots, u_n = v_n \Rightarrow \ell \rightarrow r\,[\![c]\!]$ where u_1, v_1, $\ldots$, u_n, v_n, ℓ and r are terms of $\mathcal{T}(\mathcal{F}, \mathcal{X})$, the terms ℓ and r (called resp. left- and right-hand side of the rule, $\ell \rightarrow r$ is an oriented equation) are linear and have the same sort, and c is a constraint. Note that the assumption that l and r are linear is no restrictive non linearities may be expressed as equalities in c. The conjunction of equations $u_1 = v_1, \ldots, u_n = v_n$ is called the *condition* of the rule. When $n = 0$, the above rule is called a *constrained rewrite rule.* A set $\mathcal{R}$ of conditional constrained, resp. constrained, rules is called a *conditional constrained* (resp. *constrained*) *rewrite system* or CCTRS (resp. CTRS).

We shall consider in this paper CCTRS of the form $\mathcal{R}_\mathcal{D} \uplus \mathcal{R}_\mathcal{C}$ where $\mathcal{R}_\mathcal{D}$ contains conditional constrained rules of the form $u_1 = v_1, \ldots, u_n = v_n \Rightarrow f(\ell_1, \ldots, \ell_n) \rightarrow r\,[\![c]\!]$ with $f \in \mathcal{D}$, $\ell_1, \ldots, \ell_n \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ and $\mathcal{R}_\mathcal{C}$ contains constrained rewrite rules with constructor symbols from $\mathcal{C}$ only.

A term $t\,[\![d]\!]$ rewrites to $s\,[\![d]\!]$ by the above rule of a CCTRS $\mathcal{R}$, denoted by $t\,[\![d]\!] \xrightarrow[\mathcal{R}]{} s\,[\![d]\!]$ if $t|_p = \ell\sigma$ for some position p and substitution σ, $s = t[r\sigma]_p$, the substitution σ is such that $d \wedge \neg c\sigma$ is unsatisfiable and $u_i\sigma \downarrow_\mathcal{R} v_i\sigma$ for all $i \in [1..n]$; $u\sigma \downarrow_\mathcal{R} v$ denotes that there exists w such that $u \xrightarrow[\mathcal{R}]{*} w \xleftarrow[\mathcal{R}]{*} v$, where $\xrightarrow[\mathcal{R}]{*}$ (resp. $\xrightarrow[\mathcal{R}]{+}$) is the reflexive transitive (resp. transitive) closure of $\xrightarrow[\mathcal{R}]{}$.

Note the semantic difference between conditions and constraints in rewrite rules. The validity of the condition is defined wrt the system $\mathcal{R}$ whereas the interpretation of the constraint is fixed and independent from $\mathcal{R}$.

The CCTRS $\mathcal{R}$ is *terminating* if there is no infinite sequence $t_1 \xrightarrow[\mathcal{R}]{} t_2 \xrightarrow[\mathcal{R}]{} \ldots$, and $\mathcal{R}$ is *ground-confluent* if for any ground terms $u, v, w \in \mathcal{T}(\mathcal{F})$, $v \xleftarrow[\mathcal{R}]{*} u \xrightarrow[\mathcal{R}]{*} w$, implies that $v \downarrow_\mathcal{R} w$, and $\mathcal{R}$ is *ground convergent* if $\mathcal{R}$ is both ground-confluent and terminating.

A constrained term $s\,[\![c]\!]$ is *reducible* by $\mathcal{R}$ if there exists $t\,[\![c]\!]$ such that $s\,[\![c]\!] \xrightarrow[\mathcal{R}]{} t\,[\![c]\!]$. Otherwise $s\,[\![c]\!]$ is called $\mathcal{R}$*-irreducible*, or an $\mathcal{R}$*-normal form.* The set of $\mathcal{R}$-normal forms of $s\,[\![c]\!]$ is denoted $\mathrm{NF}_\mathcal{R}(s\,[\![c]\!])$. A constrained term $s\,[\![c]\!]$ is *ground reducible* (resp. *ground irreducible*) by $\mathcal{R}$ if for every irreducible substitution $\sigma \in sol(c)$ grounding for t, $t\sigma$ is reducible (resp. irreducible) by $\mathcal{R}$.

Inductive theorems, tautologies

Let $\mathcal{R}$ be a terminating CCTRS. A constrained equation $a = b\,[\![c]\!]$ is called an *inductive theorem* of $\mathcal{R}$ (denoted by $\mathcal{R} \models_{ind} a = b\,[\![c]\!]$) if for all substitution $\sigma \in sol(c)$ grounding for a and b, $a\sigma \xleftrightarrow[\mathcal{R}]{*} b\sigma$, and it is called a *joinable inductive* *theorem* of $\mathcal{R}$ (denoted by $\mathcal{R} \models_{jind} a = b \llbracket c \rrbracket$) if for all substitution $\sigma \in sol(c)$ grounding for $a = b$, and all $\mathcal{R}$-normal forms n_a, n_b respectively of $a\sigma$, $b\sigma$, we have $n_a = n_b$. These notions are extended to clauses as expected. The definition of joinable inductive theorems is motivated by the applications presented in Sections 4-6.

Note that the two notions of inductive and joinable inductive theorem coincide when $\mathcal{R}$ is ground-confluent. However, they can differ otherwise. Consider for instance $\mathcal{R} = \{c \to a, c \to b\}$. The conjecture $a = b \Rightarrow f(x) = c$ is a joinable inductive theorem but is not an inductive theorem. On the other hand, $a = b$ is an inductive theorem but it is not a joinable inductive theorem.

We call *tautology* of $\mathcal{R}$ a constrained clause of the form $a = a \vee L \llbracket c \rrbracket$ such that a is ground irreducible by $\mathcal{R}$ or of the form $a = b \vee a \neq b \vee L \llbracket c \rrbracket$ such that a and b are ground irreducible by $\mathcal{R}$. Note that every tautology is both a inductive and a joinable inductive theorem of $\mathcal{R}$.

Completeness

A function symbol $f \in \mathcal{D}$ is *sufficiently complete* wrt $\mathcal{R}$ iff for all $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C})$, there exists t in $\mathcal{T}(\mathcal{C})$ such that $f(t_1, \ldots, t_n) \xrightarrow[\mathcal{R}]{+} t$. We say that the system $\mathcal{R}$ is sufficiently complete iff every defined operator $f \in \mathcal{D}$ is sufficiently complete wrt $\mathcal{R}$. Let $f \in \mathcal{D}$ be a function symbol and let:

$$\Big\{\Gamma_1 \Rightarrow f(t_1^1, \ldots, t_k^1) \to r_1 \llbracket c_1 \rrbracket, \ldots, \Gamma_n \Rightarrow f(t_1^n, \ldots, t_k^n) \to r_n \llbracket c_n \rrbracket\Big\}$$

be a maximal subset of rules of $\mathcal{R}_{\mathcal{D}}$ whose left-hand sides are identical up to a variable renaming $\{\mu_1, \ldots, \mu_n\}$ *i.e.* $f(t_1^1, \ldots, t_k^1)\mu_1 = f(t_1^2, \ldots, t_k^2)\mu_2 = \ldots f(t_1^n, \ldots, t_k^n)\mu_n$. We say that f is *strongly complete* wrt $\mathcal{R}$ (see [2]) if f is sufficiently complete wrt $\mathcal{R}$ and $\mathcal{R} \models_{jind} \Gamma_1\mu_1 \llbracket c_1\mu_1 \rrbracket \vee \ldots \vee \Gamma_n\mu_n \llbracket c_n\mu_n \rrbracket$ for every subset of $\mathcal{R}$ as above. The system $\mathcal{R}$ is said strongly complete if every function symbol $f \in \mathcal{D}$ is strongly complete wrt $\mathcal{R}$.

Constrained tree grammars

A *constrained tree grammar* $\mathcal{G} = (Q, \Delta)$ is given by a finite set Q of *non-terminals* of the form $\llcorner u \lrcorner$, where u is a linear term of $\mathcal{T}(\mathcal{F}, \mathcal{X})$, and a finite set Δ of *production* rules of the form $\llcorner t \lrcorner := f(\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner) \llbracket c \rrbracket$ where $f \in \mathcal{F}$, $\llcorner t \lrcorner$, $\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner \in Q$ and c is a constraint.

The non-terminals are always considered modulo variable renaming. In particular, we assume that the term $f(u_1, \ldots, u_n)$ is linear. The production relation $\vdash_{\mathcal{G}}^{x}$ on constrained terms is defined by:

$$t[x] \llbracket x{:}\llcorner u \lrcorner \wedge d \rrbracket \vdash_{\mathcal{G}}^{x} t[f(x_1, \ldots, x_n)] \llbracket x_1{:}\llcorner u_1 \lrcorner \wedge \ldots \wedge x_n{:}\llcorner u_n \lrcorner \wedge c \wedge d\sigma \rrbracket$$

if there exists $\llcorner u \lrcorner := f(\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner) \llbracket c \rrbracket \in \Delta$ such that $f(u_1, \ldots, u_n) = u\sigma$ (we assume that the variables of $u_1, \ldots, u_n$ and c do not occur in the constrained term $t[x]\,[\![x{:}\,\llcorner u \lrcorner \wedge d]\!])$ and $x_1,\ldots,x_n$ are fresh variables. The union of the relations $\vdash^x_{\mathcal{G}}$ for all x is denoted $\vdash_{\mathcal{G}}$ and the reflexive transitive and transitive closures of the relation $\vdash_{\mathcal{G}}$ are respectively denoted by $\vdash^*_{\mathcal{G}}$ and $\vdash^+_{\mathcal{G}}$.

The language $L\big(\mathcal{G},\, \llcorner u \lrcorner\big)$ is the set of ground terms t *generated* by a constrained tree grammar $\mathcal{G}$ starting with the non-terminal $\llcorner u \lrcorner$, *i.e.* such that $x\,[\![x{:}\,\llcorner u \lrcorner]\!] \vdash^*_{\mathcal{G}} t\,[\![c]\!]$ where c is satisfiable. The above membership constraints $t{:}\,\llcorner u \lrcorner$, with $\llcorner u \lrcorner \in Q$, are interpreted by: $sol(t{:}\,\llcorner u \lrcorner) = \{\sigma \mid t\sigma \in L(\mathcal{G},\, \llcorner u \lrcorner)\}$. Note that we can use such constraints in order to restrict a term to a given sort or any given regular tree language.

3 Implicit Inductive Theorem Proving procedure

We present in this section a goal-directed inductive theorem proving procedure for conditional and constrained specifications. It will be applied to the verification of cryptographic protocols in Sections 4-6. This procedure has two features which are generally not supported by former inductive theorem proving approaches and are crucial for the application to security protocols. First, it handles non ground-confluent rewrite systems. It is required in the context of verification of concurrent asynchronous communicating systems like protocols, as we shall see in Sections 4-6. Second, following the principle of [3] we use context tree grammars, which permits to deal with constrained rewrite rules between constructors. This allows us to consider explicit definition of cryptographic operators, hence a more accurate model of protocols.

This procedure belongs to the family of *implicit induction* (in the lines of [4]) and combines the power of two classical methods for automatic induction: *explicit induction* and *proof by consistency* [8]. As described below, the whole procedure is based on a constrained tree grammar, which is computed automatically from the given specification. It is used for several purposes :

(i) as an induction scheme. Using a constrained tree grammar instead of a test-set like in the former procedures [4,2] permits precisely to handle constrained rewrite rules between constructors, hence to model protocols with explicit cryptographic operators as explained above.

(ii) as an oracle for checking validity and redundancy at each induction steps, by reduction to an emptiness problem.

(iii) in order to characterize (regular) sets of special traces for the verification of trace properties, see Section 4.

3.1 Constrained Tree Grammar for Induction

Constrained tree grammars permit an exact representation of the set of ground constructor terms irreducible by a given CTRS. For this reason, such formalisms have been studied in many works related to inductive theorem proving, see *e.g.* [8]. Indeed, under some assumptions like sufficient completeness and termination for constructor axioms, they provide a finite description of the minimal Herbrand model (a set of representatives of the minimal Herbrand model is the language of ground constructor $\mathcal{R}_\mathcal{C}$-normal forms in this case).

For every constructor CTRS $\mathcal{R}_\mathcal{C}$, we can construct a constrained tree grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ which generates the language of ground $\mathcal{R}_\mathcal{C}$-normal forms. This construction, presented in [3] and exemplified in the Section 4, generalizes the one of [7] and intuitively corresponds to the complementation and completion of a tree grammar for $\mathcal{R}_\mathcal{C}$-reducible terms, where every subset of non-terminals (for the complementation) is represented by the most general unifier of its elements. The proof of the following lemma can be found in [3].

Lemma 3.1 *The language* $\bigcup_{\llcorner u \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})} L(\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}), \llcorner u \lrcorner)$ *is the set of* $\mathcal{R}_\mathcal{C}$*-irreducible terms of* $\mathcal{T}(\mathcal{C})$.

In a first step of our induction procedure, we construct a constrained tree grammar $\mathcal{G} = (Q, \Delta)$. This grammar is assumed fixed in the rest of the section. For the construction of $\mathcal{G}$, we start with $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ and possibly make an intersection with one or several regular tree grammars, following the principle of remark (iii) above (see Section 4 for an example). The intersection between a constrained tree grammar and a regular tree grammar is a constrained tree grammar. It is built with a standard product construction. We call a constrained term $t\,[\![c]\!]$ *decorated* if $c = x_1 \colon \llcorner u_1 \lrcorner \wedge \ldots \wedge x_n \colon \llcorner u_n \lrcorner \wedge d$, $\{x_1, \ldots, x_n\} = \mathit{var}(t)$, $\llcorner u_i \lrcorner \in Q$ and $\mathit{sort}(u_i) = \mathit{sort}(x_i)$ for all $i \in [1..n]$.

Some of the following inference rules invoke tests for (a) satisfiability of constraints in clauses, (b) ground irreducibility of constructor clauses and (c) joinable inductive validity of ground irreducible of constructor clauses. It is shown in [3] that the properties (a) and (b) are reducible to emptiness decision for constrained tree grammars which slightly extend $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$. A similar reduction is also possible for (c), following the idea that when a clause is ground irreducible, testing joinable inductive validity amounts, by definition, at testing syntactic equality. In other terms, when a and b are ground irreducible $\mathcal{R} \models_{\mathit{jind}} a = b\,[\![c]\!]$ iff the constraint $c \wedge a \not\approx b$ is unsatisfiable, and (c) is reducible to (a), hence to emptiness decision.

Based on former decision results for tree automata with equality and disequality constraints [6], some restrictions on $\mathcal{R}_\mathcal{C}$ are given [3] which ensure the decidability of the three above problem. The rewrite systems described in Sections 4 and 6 fulfill these restrictions.

3.2 Simplification Rules

We present in Figure 1 the system $\mathcal{S}$ of simplification rules for constructor clauses. Rewriting simplifies goals with axioms. Since $\mathcal{R}$ may not be ground-confluent, we consider all the (one step) reductions with $\mathcal{R}$. Rewrite Splitting simplifies a constrained clause which contains a subterm matching some left member of rule of $\mathcal{R}_\mathcal{D}$. The inference checks moreover that all cases are covered for the application of such rules of $\mathcal{R}_\mathcal{D}$, *i.e.* that for each ground substitution

τ, the conditions and the constraints of at least one rule is true wrt τ. Partial Splitting eliminates ground reducible terms in a constrained clause $C\,[\![c]\!]$ by adding to $C\,[\![c]\!]$ the negation of constraint of some rules of $\mathcal{R}_\mathcal{C}$. Therefore, the saturated application of Partial splitting and Rewriting will always lead to Deletion or to ground irreducible constructor clauses. Finally, Deletion and Validity remove respectively tautologies and clauses with unsatisfiable constraints, and ground irreducible constructor joinable inductive theorems of $\mathcal{R}$. As explained in Section 3.1, the tests in the rules Deletion and Validity are discharged to a decision procedure for the emptiness of constrained tree grammars.

Rewriting $\quad C\,[\![c]\!] \vdash_\mathcal{S} \{D_1\,[\![c]\!], \ldots, D_k\,[\![c]\!]\}$

if $\quad$ for all $i \leq k$, $D_i\,[\![c]\!] \ll C\,[\![c]\!]$

where $\quad \{D_1\,[\![c]\!], \ldots, D_k\,[\![c]\!]\}$ are all the clauses obtained by one-step rewriting with $\mathcal{R}$ from $C\,[\![c]\!]$.

Rewrite Splitting $\quad C\,[\![c]\!] \vdash_\mathcal{S} \big\{\Gamma_i\sigma_i \Rightarrow C[r_i\sigma_i]_{p_i}\,[\![c \wedge c_i\sigma_i]\!] \;\big|\; p_i \text{ pos. of } C\big\}_{i\in[1..n]}$

if $\quad \mathcal{R} \models_{jind} \Gamma_1\sigma_1\,[\![c \wedge c_1\sigma_1]\!] \vee \ldots \vee \Gamma_n\sigma_n\,[\![c \wedge c_n\sigma_n]\!]$, $C|_{p_i} > r_i\sigma_i$ and $\{C|_{p_i}\} >^{mul} \Gamma_i\sigma_i$

where $\quad$ the $\Gamma_i\sigma_i \Rightarrow l_i\sigma_i \rightarrow r_i\sigma_i\,[\![c_i\sigma_i]\!]$, $i \in [1..n]$, are all the instances of rules $\Gamma_i \Rightarrow l_i \rightarrow r_i\,[\![c_i]\!] \in \mathcal{R}_\mathcal{D}$ such that $l_i\sigma_i = C|_{p_i}$

Partial Splitting $\quad C[l\sigma]_p\,[\![c]\!] \vdash_\mathcal{S} \big\{C[r\sigma]_p\,[\![c \wedge c'\sigma]\!], C[l\sigma]_p\,[\![c \wedge \neg c'\sigma]\!]\big\}$

if $\quad l \rightarrow r\,[\![c']\!] \in \mathcal{R}_\mathcal{C}$, $l\sigma > r\sigma$, and neither $c'\sigma$ nor $\neg c'\sigma$ is a subformula of c

where $\quad C\,[\![c]\!]$ is a constructor clause.

Deletion $\quad C\,[\![c]\!] \vdash_\mathcal{S} \emptyset$

if $\quad C\,[\![c]\!]$ is a tautology or c is unsatisfiable

Validity $\quad C\,[\![c]\!] \vdash_\mathcal{S} \emptyset$

if $\quad C\,[\![c]\!]$ is a ground irreducible constructor clause and $\mathcal{R} \models_{jind} C\,[\![c]\!]$

Figure 1: System $\mathcal{S}$: simplification rules

3.3 Main inference system

The main inference system $\mathcal{I}$ is displayed in Figure 2. Its rules apply to pairs $(\mathcal{E}, \mathcal{H})$, where $\mathcal{E}$ is the set of current conjectures and $\mathcal{H}$ is the *set* of inductive

Simplification $$\frac{(\mathcal{E} \cup \{C\,[\![c]\!]\}, \mathcal{H})}{(\mathcal{E} \cup \mathcal{E}', \mathcal{H})}$$

if $\mathcal{I}ndvar(C\,[\![c]\!]) = \emptyset$ and $C\,[\![c]\!] \vdash_{\mathcal{S}} \mathcal{E}'$

Inductive Narrowing $$\frac{(\mathcal{E} \cup \{C\,[\![c]\!]\}, \mathcal{H})}{(\mathcal{E} \cup \mathcal{E}_1 \cup \ldots \cup \mathcal{E}_n, \mathcal{H} \cup \{C\,[\![c]\!]\})}$$

if for all i in $[1..n]$, $d(C_i) - d(C) \leq d(\mathcal{R}) - 1$ and $C_i\,[\![c_i]\!] \vdash_{\mathcal{S}} \mathcal{E}_i$

where $\{C_1\,[\![c_1]\!], \ldots, C_n\,[\![c_n]\!]\}$ is the set of clauses s. t. $C\,[\![c]\!] \vdash^{+}_{\mathcal{G}} C_i\,[\![c_i]\!]$

Subsumption $$\frac{(\mathcal{E} \cup \{C\,[\![c]\!]\}, \mathcal{H})}{(\mathcal{E}, \mathcal{H})}$$

if $C\,[\![c]\!]$ is subsumed by another clause of $\mathcal{R} \cup \mathcal{E} \cup \mathcal{H}$

Disproof $$\frac{(\mathcal{E} \cup \{C\,[\![c]\!]\}, \mathcal{H})}{(\mathsf{Disproof}, \mathcal{H})}$$

if $C\,[\![c]\!]$ is a constructor clause and no other rule applies to $C\,[\![c]\!]$

Failure $$\frac{(\mathcal{E} \cup \{C\,[\![c]\!]\}, \mathcal{H})}{(\mathsf{Failure}, \mathcal{H})}$$

if $C\,[\![c]\!]$ is not a constructor clause

and no other rule applies to the clause $C\,[\![c]\!]$

Figure 2: System $\mathcal{I}$: inference rules for joinable-induction

hypotheses (constrained clauses). The inference rules of $\mathcal{I}$ use the constrained tree grammar $\mathcal{G}$ in order to instantiate variables. The replacements are limited to variables, called *induction variables*, whose instantiation is needed in order to trigger a rewrite step.

Definition 3.2 The set $\mathcal{I}ndpos(f, \mathcal{R})$ of *induction positions* of $f \in \mathcal{D}$ is the set of non-root and non-variable positions of left-hand sides of rules of $\mathcal{R}_{\mathcal{D}}$ with the symbol f at the root position. The set $\mathcal{I}ndvar(t)$ of *induction variables* of $t = f(t_1, \ldots, t_n)$, with $f \in \mathcal{D}$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, is the subset of variables of $var(t)$ occurring in t at positions of $\mathcal{I}ndpos(f, \mathcal{R})$.

Let us now describe the inference rules of $\mathcal{I}$. Simplification reduces a conjecture which does not contain any induction variable using the rules of System $\mathcal{S}$ (Figure 1). Inductive Narrowing generates new subgoals by application of the production rules of the constrained grammar $\mathcal{G}$ until the obtained clause is deep enough to cover left hand side of rules of $\mathcal{R}_{\mathcal{D}}$. Each obtained clause must be simplified by one of the rules of $\mathcal{S}$ (if one instance cannot be simplified, then the rule Inductive Narrowing cannot be applied). Subsumption deletes clauses redundant with axioms of $\mathcal{R}$, induction hypotheses of $\mathcal{H}$ and other conjectures not yet proved (in $\mathcal{E}$).

3.4 Soundness and Completeness

Our inference system is sound, and refutationally complete.

Definition 3.3 We call *derivation* a sequence of inference steps generated by a pair of the form $(\mathcal{E}_0, \emptyset)$, using the inference rules in $\mathcal{I}$, written $(\mathcal{E}_0, \emptyset) \vdash_{\mathcal{I}} (\mathcal{E}_1, \mathcal{H}_1) \vdash_{\mathcal{I}} \cdots (\mathcal{E}_n, \mathcal{H}_n) \vdash_{\mathcal{I}} \cdots$. We say that a derivation is *fair* if the set of persistent constrained clauses $(\cup_i \cap_{j \geq i} \mathcal{E}_j)$ is empty or equal to Disproof or Failure. The derivation is said to be a *disproof* or *failure*, respectively, in the two last cases, and a *success* in the first case.

Finite success is obtained when the set of conjectures to be proved is exhausted. Infinite success is obtained when the procedure diverges, assuming fairness. When it happens, the clue is to guess some lemmas which are used to subsume or simplify the generated infinite family of subgoals, therefore stopping the divergence. This is possible in our approach, since lemmas can be used in the same way as axioms are.

Lemma 3.4 *Let $s_1, \ldots, s_n$ be all the terms obtained by one-step rewriting with $\mathcal{R}$ from a term $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ which contains no induction variables. For all substitution θ grounding for s, $\mathrm{NF}_{\mathcal{R}}(s\theta) = \bigcup_{i=1}^{k} \mathrm{NF}_{\mathcal{R}}(s_i\theta)$.*

Proof. The direction $\supseteq$ is obvious. We prove the direction $\subseteq$ by contradiction. Assume that there is a normal form of $s\theta$ which is not an element of $\bigcup_{i=1}^{k} \mathrm{NF}_{\mathcal{R}}(s_i\theta)$. Then the first rewrite step from $s\theta$ is necessarily done at a non-root position, this implies that s contains an induction variable, a contradiction with hypotheses. □

Theorem 3.5 (Soundness of successful derivations) *Let $\mathcal{E}_0$ be a set of decorated constrained clauses. If there exists a successful derivation $(\mathcal{E}_0, \emptyset) \vdash_{\mathcal{I}} (\mathcal{E}_1, \mathcal{H}_1) \vdash_{\mathcal{I}} \cdots$ then $\mathcal{R} \models_{jind} \mathcal{E}_0$.*

Proof. Assume that $\mathcal{R} \not\models_{jind} \mathcal{E}_0$, and let $(\mathcal{E}_0, \emptyset) \vdash_{\mathcal{I}} (\mathcal{E}_1, \mathcal{H}_1) \vdash_{\mathcal{I}} \cdots$ be an arbitrary successful derivation. Let D_0 be a clause minimal wrt $\gg$ in the set:

$$\big\{ D\sigma \;\big|\; D\,[\![d]\!] \in \cup_i \mathcal{E}_i, \sigma \in sol(d) \text{ is constructor and irreducible and } \mathcal{R} \not\models_{jind} D\sigma \big\}$$

Note that such a clause exists since we have assumed that $\mathcal{R} \not\models_{jind} \mathcal{E}_0$. Let $C\,[\![c]\!]$ be a clause of $\cup_i \mathcal{E}_i$ minimal by subsumption ordering and let $\theta \in sol(c)$ an irreducible and constructor ground substitution such that $C\theta = D_0$.

We show that we obtain a contradiction when any inference, other than Disproof or Failure, is applied to $C\,[\![c]\!]$, hence the above derivation is not successful.

Simplification. We consider all the subcases of Figure 1:

Rewriting: let $\{D_1\,[\![c]\!], \ldots, D_k\,[\![c]\!]\}$ be the result of the application of this rule to $C\,[\![c]\!]$. We recall that $\mathcal{I}ndvar\,[\![c]\!] = \emptyset$ and $D_1\,[\![c]\!], \ldots, D_k\,[\![c]\!]$ are all the clauses obtained by one-step rewriting from $C \llbracket c \rrbracket$. Hence by lemma 3.4, there exists $i \leq k$ such that $\mathcal{R} \not\models_{jind} D_i \llbracket c \rrbracket$. Moreover, $C\theta \gg D_i \llbracket c \rrbracket$ and $D_i \llbracket c \rrbracket \in \cup_i \mathcal{E}_i$, and it is a contradiction.

Rewrite Splitting. Assume that this rule is applied to $C \llbracket c \rrbracket$, and let $\{\Gamma_1 \Rightarrow \ell_1 \to r_1 \llbracket c_1 \rrbracket, \ \ldots \ , \Gamma_n \Rightarrow \ell_n \to r_n \llbracket c_n \rrbracket\}$ be the non-empty subset of $\mathcal{R}_\mathcal{D}$ such that for all i in $[1..n]$, $C|_{p_i} = \ell_i \sigma_i$ and

$$\mathcal{R} \models_{jind} \Gamma_1\sigma_1 \llbracket c \wedge c_1\sigma_1 \rrbracket \vee \ldots \vee \Gamma_n\sigma_n \llbracket c \wedge c_n\sigma_n \rrbracket \tag{1}$$

The result of the application of Rewrite Splitting is:

$$\{\Gamma_1\sigma_1 \Rightarrow C[r_1\sigma_1]_{p_1} \llbracket c \wedge c_1\sigma_1 \rrbracket, \ldots, \Gamma_n\sigma_n \Rightarrow C[r_n\sigma_n]_{p_n} \llbracket c \wedge c_n\sigma_n \rrbracket\}$$

where $p_1, \ldots, p_n$ are positions in C. From (1) it follows that there exists $k \leq n$ such that $\mathcal{R} \models_{jind} \Gamma_k\sigma_k\delta$ for all $\delta \in \mathit{Sol}(c \wedge c_k\sigma_k)$. Let C_k be $\Gamma_k\sigma_k \Rightarrow C[r_k\sigma_k]_{p_k} \llbracket c \wedge c_k\sigma_k \rrbracket$, we have $\mathcal{R} \not\models_{jind} C_k\delta$, since $\mathcal{R} \models_{jind} \Gamma_k\sigma_k\delta$, $\mathcal{R} \models_{jind} \ell_k\sigma_k\delta = r_k\sigma_k\delta$, and $\mathcal{R} \not\models_{jind} C \llbracket c \rrbracket$. On the other hand, $C \gg C_k$ since $\{\ell_k\sigma_k\} >^{mul} \Gamma_k\sigma_k$, and $\ell_k\sigma_k > r_k\sigma_k$. This contradicts the minimality of $C \llbracket c \rrbracket$.

Partial Splitting: this case is similar to the previous one.

Deletion, Validity: since $C \llbracket c \rrbracket$ is a counter-example, these rules cannot be applied.

Inductive Narrowing. Suppose that this inference rule is applied to $C \llbracket c \rrbracket$. By hypothesis, C has been decorated, *i.e.* $c = d \wedge x_1 : \llcorner u_1 \lrcorner \wedge \ldots \wedge x_n : \llcorner u_n \lrcorner$ with $\{x_1, \ldots, x_n\} = \mathit{var}(C)$ and for all $i \in [1..n]$, $\llcorner u_i \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$. Hence, since $\theta \in \mathit{sol}(c)$, there exists σ and τ such that $\theta = \sigma\tau$ and $C \llbracket c \rrbracket \vdash^+ C\sigma \llbracket c' \rrbracket$. We show, with a similar case analysis as above, that whatever rule of $\mathcal{S}$ (Figure 1) is applied to $C\sigma \llbracket c' \rrbracket$, we obtain a contradiction.

Subsumption: Since $\mathcal{R} \not\models_{jind} C\theta$, $C \llbracket c \rrbracket$ cannot be subsumed by an axiom of $\mathcal{R}$. If there exists $H \llbracket h \rrbracket \in \mathcal{H} \cup (\mathcal{E} \setminus \{C \llbracket c \rrbracket\})$ such that $C \llbracket c \rrbracket \equiv H\delta \llbracket h\delta \rrbracket \vee D$, then we have $\mathcal{R} \not\models_{jind} H\delta\theta$ ($\theta \in \mathit{sol}(h\delta)$). Hence, $D = \emptyset$ and δ is the identity, since $C \llbracket c \rrbracket$ is minimum in $\cup_i \mathcal{E}_i$ wrt subsumption ordering. Therefore, $H \notin (\mathcal{E} \setminus \{C\})$. Moreover, $H \notin \mathcal{H}$, otherwise the inference Inductive Narrowing could also be applied to $C \llbracket c \rrbracket$, in contradiction with previous cases. Hence, Subsumption cannot be applied to $C \llbracket c \rrbracket$. □

Our inference system can refute false conjectures. This result is a consequence of the following lemma.

Lemma 3.6 *Let* $(\mathcal{E}_i, \mathcal{H}_i) \vdash_\mathcal{I} (\mathcal{E}_{i+1}, \ \mathcal{H}_{i+1})$ *be a derivation step. If* $\mathcal{R} \models_{jind} \mathcal{E}_i \cup \mathcal{H}_i$ *then* $\mathcal{R} \models_{jind} \mathcal{E}_{i+1} \cup \mathcal{H}_{i+1}$.

Proof. Let $C[\![c]\!]$ be a clause in $\mathcal{E}_i$ and $(\mathcal{E}_i \cup \{C[\![c]\!]\}, \mathcal{H}_i) \vdash_{\mathcal{I}} (\mathcal{E}_{i+1}, \mathcal{H}_{i+1})$ be a derivation step obtained by the application of an inference to $C[\![c]\!]$ and assume that $\mathcal{R} \models_{jind} \mathcal{E}_i \cup \mathcal{H}_i$. We can show that $\mathcal{R} \models_{jind} \mathcal{E}_{i+1} \cup \mathcal{H}_{i+1}$ by a case analysis according to the rule applied to $C[\![c]\!]$. □

Theorem 3.7 (Soundness of disproof) *If a derivation starting from* $(\mathcal{E}_0, \emptyset)$ *returns the pair* $(\mathsf{Disproof}, \mathcal{H})$, *then* $\mathcal{R} \not\models_{jind} \mathcal{E}_0$.

Proof. Let $C[\![c]\!]$ be the premise of the instance of the rule Disproof which has been applied. The clause $C[\![c]\!]$ is a constructor clause, and contains ground irreducible terms only. Otherwise Inductive Narrowing would apply. Moreover, the rule Validity cannot be applied to $C[\![c]\!]$, hence $C[\![c]\!]$ is not a joinable inductive theorem of $\mathcal{R}$ and $\mathcal{R} \not\models_{jind} \mathcal{E}_k$. By lemma 3.6, we conclude that $\mathcal{R} \not\models_{jind} \mathcal{E}_0$. □

The derivation of Failure means that we cannot conclude, however, as stated in the following theorem, this never happens providing that $\mathcal{R}$ is strongly complete.

Theorem 3.8 (Refutational completeness) *Assume that* $\mathcal{R}$ *is strongly complete and let* $\mathcal{E}_0$ *be a set of decorated constrained clauses. If* $\mathcal{R} \not\models_{jind} \mathcal{E}_0$, *then all fair derivations starting from* $(\mathcal{E}_0, \emptyset)$ *end up with* $(\mathsf{Disproof}, \mathcal{H})$.

Proof. By Theorem 3.5, every derivation $(\mathcal{E}_0, \emptyset) \vdash_{\mathcal{I}} (\mathcal{E}_1, \mathcal{H}_1) \vdash_{\mathcal{I}} \cdots$ is not successful. Since we consider only fair derivations, it means that either the rule Disproof or the rule Failure is applied. Assume that Failure is applied to a clause $C[\![c]\!]$ which belongs to $\mathcal{E}_k$ for some k. We shall prove that $C[\![c]\!]$ is a constructor clause, which contradicts the fact that Disproof is not applicable to $C[\![c]\!]$, according to its definition in Figure 2.

Assume indeed that $C[\![c]\!]$ contains a term of the form $f(t_1, \ldots, t_n)$, where $f \in \mathcal{D}$ and for all $i \in [1..n]$, $t_i \in T(\mathcal{C}, \mathcal{X})$. The constraint c is satisfiable, otherwise Deletion could be applied. Let $\tau \in sol(c)$. By Lemma 3.1, for each $x \in var(C)$, $x\tau$ is in $\mathcal{R}_{\mathcal{C}}$-normal form. We have now two possibilities:
(i) for one $i \in [1..n]$, $t_i\tau$ is reducible. In this case, there exists a substitution σ such that $\tau = \sigma\theta$ and $t_i[\![c]\!] \vdash^{+} t_i\sigma[\![c']\!]$ and $t_i\sigma$ contains as a subterm an instance of a left-hand side of rule of $\mathcal{R}_{\mathcal{C}}$. Therefore, either Rewriting or Partial Splitting can be applied to $t_i\sigma[\![c']\!]$. It implies that Inductive Narrowing can be applied to $C[\![c]\!]$, which is a contradiction.
(ii) every $t_i\tau$ is irreducible. The term $f(t_1, \ldots, t_n)\tau$ is reducible at root position because f is strongly complete wrt $\mathcal{R}$. Then there exists σ such that $\tau = \sigma\theta$ and $f(t_1, \ldots, t_n)[\![c]\!] \vdash^{+} f(t_1, \ldots, t_n)\sigma[\![c']\!]$ and moreover $f(t_1, \ldots, t_n)\sigma$ is an instance of a left-hand side of rule of $\mathcal{R}_{\mathcal{D}}$. Therefore, either Rewriting or Rewrite Splitting can be applied. Indeed the application condition of the latter inference is a consequence of the strongly completeness of $\mathcal{R}$. Hence, the inference Inductive Narrowing can be applied to $C[\![c]\!]$, which is a contradiction. In conclusion, the clause $C[\![c]\!]$ contains only constructor terms. □

4 Verification of a Key Distribution Protocol

We describe in this section how to apply the above implicit induction procedure to the verification of a security protocol, where a security property is expressed as a joinable inductive conjecture. Following [12], we perform induction on protocol execution traces which are recursively defined by a terminating CCTRS $\mathcal{R}$. Since $\mathcal{R}$ is generally not ground-confluent, we need to consider *all* the traces for the verification of a property, hence the restriction to joinable inductive theorem proving (instead of classical inductive theorem proving).

We consider a simplification (without certificates and timestamps) of a key distribution protocol of Denning & Sacco [9] for a symmetric key exchange in an asymmetric cryptosystem. Assume some sorts Nat, Bool, Name, Id, Key, Msg, $\mathsf{MsgList}$, with the subsort relations: $\mathsf{Name} \subseteq \mathsf{Msg}$ and $\mathsf{Key} \subseteq \mathsf{Msg}$. The messages exchanged during the protocol execution are abstracted by well sorted terms built with constructor symbols $\mathsf{pair} : \mathsf{Msg} \times \mathsf{Msg} \rightarrow \mathsf{Msg}$ and projections $\mathsf{fst}, \mathsf{snd} : \mathsf{Msg} \rightarrow \mathsf{Msg}$ which follow the rules:

$$\mathsf{fst}(\mathsf{pair}(x_1, x_2)) \rightarrow x_1 \quad \text{and} \quad \mathsf{snd}(\mathsf{pair}(x_1, x_2)) \rightarrow x_2 \tag{2}$$

decryption in symmetric key cryptography:

$$\mathsf{dec}\big(\mathsf{enc}(x, y), y\big) \rightarrow x \tag{3}$$

(the variables x and y correspond respectively to the encrypted plaintext and the encryption key), decryption in public (asymmetric) key cryptography:

$$\mathsf{adec}\big(\mathsf{aenc}(x, y), \mathsf{inv}(y)\big) \rightarrow x \quad \text{and} \quad \mathsf{adec}\big(\mathsf{aenc}(x, \mathsf{inv}(y)), y\big) \rightarrow x \tag{4}$$

where inv is an idempotent operator, following the rule:

$$\mathsf{inv}(\mathsf{inv}(y)) \rightarrow y \tag{5}$$

The operator inv associates to a public encryption key its corresponding private key (for decryption), and conversely. The symbols of cryptographic operators have the following profiles: $\mathsf{pub} : \mathsf{Name} \rightarrow \mathsf{Key}$, $\mathsf{inv} : \mathsf{Key} \rightarrow \mathsf{Key}$, $\mathsf{enc}, \mathsf{aenc}, \mathsf{dec}, \mathsf{adec} : \mathsf{Msg} \times \mathsf{Msg} \rightarrow \mathsf{Msg}$, inv is called *secret* and all the others are called *public*. We consider also a public constructor $\mathsf{sent} : \mathsf{Id} \times \mathsf{Name} \times \mathsf{Name} \times \mathsf{Msg} \rightarrow \mathsf{Msg}$ to encapsulate messages with a header. Its first argument is a message identifier, the second and third arguments are respectively the names of sender and receiver of the message and the last argument is the message itself. The public constructor symbol $\mathsf{body} : \mathsf{Msg} \rightarrow \mathsf{Msg}$ can be used for removing the header, with the rule:

$$\mathsf{body}(\mathsf{sent}(x_i, x_a, x_b, x)) \rightarrow x \tag{6}$$

We assume moreover some additional secret constructors for Boolean: $\mathsf{true}, \mathsf{false} : \mathsf{Bool}$, natural numbers $0 : \mathsf{Nat}$, $s : \mathsf{Nat} \to \mathsf{Nat}$, lists of messages, $\mathsf{nil} : \mathsf{EventList}$, $:: \; : \mathsf{Msg} \times \mathsf{MsgList} \to \mathsf{MsgList}$ and constant values used in the protocol messages: $K : \mathsf{Key}$, $S : \mathsf{Msg}$. Finally, we assume that the set of names of honest agents (*i.e.* the set of terms of sort Name) is a (possibly infinite) regular tree set[5] whose terms are made only of public constructor symbols.

Let us denote $\mathcal{R}_{\mathcal{C}}$ the set of the above rules, which are sometimes referred as *explicit destructors* rules in the protocol verification literature. The constrained tree grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ contains the following sorted non terminals: $\llcorner \mathsf{pair}(x_1, x_2) \lrcorner$, $\llcorner \mathsf{enc}(x, y) \lrcorner$, $\llcorner \mathsf{aenc}(x, y) \lrcorner$, $\llcorner \mathsf{inv}(v) \lrcorner$, $\llcorner \mathsf{aenc}(x, \mathsf{inv}(y)) \lrcorner$, $\llcorner \mathsf{sent}(x_i, x_a, x_b, x) \lrcorner$, Name, $\llcorner x \lrcorner^{\mathsf{Key}}$, $\llcorner x \lrcorner^{\mathsf{Msg}}$, $\llcorner x \lrcorner^{\mathsf{List}}$, $\llcorner x \lrcorner^{\mathsf{Bool}}$, $\llcorner x \lrcorner^{\mathsf{Nat}}$ and $\llcorner x \lrcorner^{\mathsf{red}}$. We assume that Name is the initial non-terminal of a regular tree grammar generating the constructor terms of sort Name. The constrained production rules of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ are (M represents below any non-terminal of sort Msg):

$$
\begin{array}{c}
\llcorner x \lrcorner^{\mathsf{Nat}} := 0 \mid s(\llcorner x \lrcorner^{\mathsf{Nat}}) \quad \llcorner x \lrcorner^{\mathsf{List}} := \mathsf{nil} \mid M :: \llcorner x \lrcorner^{List} \quad \llcorner x \lrcorner^{\mathsf{Key}} := K \mid \mathsf{pub}(\mathsf{Name}) \\
\llcorner \mathsf{enc}(x, y) \lrcorner := \mathsf{enc}(M_1, M_2) \qquad \llcorner \mathsf{inv}(y) \lrcorner := \mathsf{inv}(\llcorner x \lrcorner^{\mathsf{Key}}) \\
\llcorner \mathsf{aenc}(x, y) \lrcorner := \mathsf{aenc}(M_1, M_2) \quad \llcorner \mathsf{aenc}(x, \mathsf{inv}(y)) \lrcorner := \mathsf{aenc}(M, \llcorner \mathsf{inv}(y) \lrcorner) \\
\llcorner \mathsf{pair}(x_1, x_2) \lrcorner := \mathsf{pair}(M_1, M_2) \quad \llcorner x \lrcorner^{\mathsf{red}} := \mathsf{fst}(\llcorner \mathsf{pair}(x_1, x_2) \lrcorner) \mid \mathsf{snd}(\llcorner \mathsf{pair}(x_1, x_2) \lrcorner) \\
\llcorner \mathsf{sent}(x_i, x_a, x_b, x) \lrcorner := \mathsf{sent}(\llcorner x \lrcorner^{\mathsf{Id}}, \llcorner x \lrcorner^{\mathsf{Name}}, \llcorner x \lrcorner^{\mathsf{Name}}, M) \quad \llcorner x \lrcorner^{\mathsf{red}} := \mathsf{body}(\llcorner \mathsf{sent}(x_i, x_a, x_b, x) \lrcorner) \\
\llcorner x \lrcorner^{\mathsf{Msg}} := \mathsf{dec}(\llcorner \mathsf{enc}(x, y) \lrcorner, M) \, [\![y \not\approx M]\!] \quad \llcorner x \lrcorner^{\mathsf{red}} := \mathsf{dec}(\llcorner \mathsf{enc}(x, y) \lrcorner, M) \, [\![y \approx M]\!] \\
\llcorner x \lrcorner^{\mathsf{Msg}} := \mathsf{adec}(\llcorner \mathsf{aenc}(x, y_1) \lrcorner, \llcorner \mathsf{inv}(y_2) \lrcorner) \, [\![y_1 \not\approx y_2]\!] \quad \llcorner x \lrcorner^{\mathsf{red}} := \ldots \, [\![y_1 \approx y_2]\!] \\
\llcorner x \lrcorner^{\mathsf{Msg}} := \mathsf{adec}(\llcorner \mathsf{aenc}(x, \mathsf{inv}(y)) \lrcorner, M) \, [\![y \not\approx M]\!] \quad \llcorner x \lrcorner^{\mathsf{red}} := \ldots \, [\![y \approx M]\!]
\end{array}
$$

The non terminal $\llcorner x \lrcorner^{\mathsf{red}}$ generates all $\mathcal{R}_{\mathcal{C}}$-reducible ground constructor terms, and the other n.t. generate all the ground constructor $\mathcal{R}_{\mathcal{C}}$-normal forms.

Following the approach of [12], we consider traces of messages modelled as lists (built with nil and $::$) and characterized by the defined symbol $\mathsf{trace} : \mathsf{Int} \times \mathsf{MsgList} \to \mathsf{Msg}$. The semantics of $\mathsf{trace}(n, \ell)$ is that ℓ is a set of messages that can be found in the network after at most n operations performed by honest or dishonest agents (altogether). The symbol trace is defined recursively by:

$$\mathsf{trace}(0, y) = y \tag{7}$$

and by extension with messages sent by the agents participating to the protocol (honest or not). In the case of the Denning & Sacco protocol, the conditional rule (DS-A) describes the honest agent x_a sending to agent x_b a message of identifier 1, which contains a freshly chosen symmetric key K for further secure

[5] we will not define explicitly a tree grammar for Name here, we just assume that it contains the constants A and B.

communications. (K is a constant constructor symbol):

$$\begin{array}{l}\mathsf{trace}(s(n), y) \rightarrow \\ \quad \mathsf{trace}\big(n, \mathsf{sent}(1, x_a, x_b, \mathsf{pair}(x_a, \mathsf{aenc}(\mathsf{aenc}(K, \mathsf{inv}(\mathsf{pub}(x_a))), \mathsf{pub}(x_b)))) :: y\big) \\ \hfill [\![x_a : \mathsf{Name}, x_b : \mathsf{Name}, x_a \not\approx x_b]\!] \quad (\text{DS-A})\end{array}$$

This key K is encrypted, for authentication purpose, using the asymmetric encryption function aenc and the secret key of x_a, represented as the inverse $\mathsf{inv}(\mathsf{pub}(x_a))$ of its public key $\mathsf{pub}(x_a)$. The result of this encryption is later encrypted with x_b's public key $\mathsf{pub}(x_b)$ so that only x_b shall be able to learn K. Moreover, x_a appends its name at the beginning of the message (using the pairing function pair) so that the receiver x_b knows which public key to use in order to obtain K.

In the second conditional rule (DS-B), the honest agent x_b, while reading a message x, expects that x has the above form.

$$\begin{array}{l}\mathsf{sent}(1, x'_a, x_b, x_m) \in y = \mathsf{true} \Rightarrow \mathsf{trace}\big(s(n), y\big) \rightarrow \mathsf{trace}(n, \mathsf{sent}(2, x_b, \mathsf{fst}(x_m), \\ \qquad \mathsf{enc}(S, \mathsf{adec}(\mathsf{adec}(\mathsf{snd}(x_m), \mathsf{inv}(\mathsf{pub}(x_b))), \mathsf{pub}(\mathsf{fst}(x_m))))) :: y\big) \quad (\text{DS-B})\end{array}$$

Then, he extracts the symmetric key K, applying twice the asymmetric decryption function adec to the second component of x, obtained by application of the projection function snd. This key K is then used by agent x_b to encrypt (with the function enc) a secret code S (S is a constant constructor symbol) that he wants to communicate to the agent x_a, and this ciphertext is sent in a message of identifier 2.

We assume asynchronous communication of the messages through an insecure public network controlled by a dishonest agent called attacker. The attacker is able to read and analyse any message sent to the network and to resend new messages composed from the information collected. Both the extraction of information from the messages read and the composition of new messages are modeled by the application of public constructor symbols and the reduction using the rules of $\mathcal{R}_\mathcal{C}$, which makes the framework with explicit destructors more uniform that others like *e.g.* so called "Dolev-Yao" models. Besides uniformity, the addition of explicit destructor rules makes the model strictly more expressive, in the sense that it captures more attacks [11]. These operations of the attacker are specified by the following rules for trace:

$$\mathsf{trace}(s(n), y) \rightarrow \mathsf{trace}(n, x :: y)\, [\![x : \mathsf{Init}]\!] \qquad (\text{att-init})$$

$$x_1 \in y = \mathsf{true}, \ldots, x_k \in y = \mathsf{true} \Rightarrow \mathsf{trace}(s(n), y) \rightarrow \mathsf{trace}(n, f(x_1, \ldots, x_k) :: y) \qquad (\text{att-anlz})$$

In the rule (att-init), Init is a non-terminal which generates a regular tree language representing the initial knowledge of the attacker. In this example, we assume that this language contains all the terms of sort Name and Id. The rule (att-anlz) above represents actually one conditional rule for each public constructor symbol f of arity k. In our example, $k = 1$ and f is pub, fst, snd, body or $k = 2$ and f is pair, enc, dec, aenc, adec.

The defined function $\in$: $\mathsf{Msg} \times \mathsf{MsgList} \to \mathsf{Bool}$ follows the three rules:

$$x \in \mathsf{nil} \to \mathsf{false},\ x_1 \in x_2 :: y \to \mathsf{true}\,[\![x_1 \approx x_2]\!],\ x_1 \in x_2 :: y \to x_1 \in y\,[\![x_1 \not\approx x_2]\!]$$

Note that the set of the above rules form a CCTRS sufficiently complete and terminating, but not ground-confluent.

We construct a the grammar $\mathcal{G}$ by intersection between $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ and a regular tree grammar $\mathcal{B}ad$ which generates the bad traces. The grammar $\mathcal{G}$ is built with a product construction and for the sake of readability, every non-terminal of $\mathcal{G}$ will be denoted below $N_1 \cap N_2$ where N_1 is a non-terminal of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ and N_2 is a non-terminal of $\mathcal{B}ad$. We assume two non-terminal in the grammar $\mathcal{B}ad$:

- a non-terminal $B^{\mathsf{DS}}_{\mathsf{auth}}$ which generates the set of lists (built with nil, $::$ and the other constructor symbols) containing a message of the form $\mathsf{sent}(2, B, A, \ldots)$ not preceded by a message of the form $\mathsf{sent}(1, A, B, \ldots)$. Such a list corresponds to an *authentication* flaw. Note that this set is a regular tree language.
- a non-terminal B_{sec} which generates the set of lists containing the constant S. Such a list corresponds to an *secrecy* flaw: it indicates that the secret value S is publicly revealed.

The two conjectures (C_{auth}, C_{sec}) express that a trace in the language of bad traces (characterized respectively by the two above non terminals of $\mathcal{B}ad$) is not a protocol trace:

$$y \neq \mathsf{trace}(n, \mathsf{nil})\ [\![y : \llcorner y \lrcorner^{\mathsf{List}} \cap B^{\mathsf{DS}}_{\mathsf{auth}}, n : \llcorner x \lrcorner^{\mathsf{Nat}}]\!] \qquad (C_{\mathsf{auth}})$$
$$y \neq \mathsf{trace}(n, \mathsf{nil})\ [\![y : \llcorner y \lrcorner^{\mathsf{List}} \cap B_{\mathsf{sec}}, n : \llcorner x \lrcorner^{\mathsf{Nat}}]\!] \qquad (C_{\mathsf{sec}})$$

More precisely, (C_{auth}) expresses that no authentication flaw (man-in-the-middle attack) occurs during protocol executions, and (C_{sec}) expresses that the constant S remains secret to the attacker. The negation $y \neq \mathsf{trace}(n, \mathsf{nil})$ should be understood as $y = \mathsf{trace}(n, \mathsf{nil}) \Rightarrow \mathsf{true} = \mathsf{false}$. Note that the above variables y and n are constrained to be instantiated by terms generated by $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ starting respectively from the non-terminals $\llcorner y \lrcorner^{\mathsf{List}}$ and $\llcorner x \lrcorner^{\mathsf{Nat}}$, and y is moreover constrained to be instantiated by terms generated by $\mathcal{B}ad$.

The application of our procedure shows that none of the conjectures is a joinable inductive theorems of $\mathcal{R}$, by induction on traces.
Among the instances of the conjecture (C_{auth}) generated[6] by application of the production rules of $\mathcal{G}$, we have the instance where the variable y is replaced by nil and n is replaced by $s^8(0)$, denoted 8 below. We can show that

[6] The procedure generates all the instances which are smaller than $d(\mathcal{R})$ (the maximum depth of the left-hand sides of rules of $\mathcal{R}$).

$$
\begin{aligned}
\mathsf{trace}\big(8,\mathsf{nil}\big) &\xrightarrow[(\text{att-init})]{*} \mathsf{trace}\big(5, I :: A :: B :: 1 :: \mathsf{nil}\big)\\
&\xrightarrow[(\text{att-anlz})]{*} \mathsf{trace}\big(2, m_I :: I :: A :: B :: 1 :: \mathsf{nil}\big)\\
&\xrightarrow[(\text{att-anlz})]{} \mathsf{trace}\big(1, \mathsf{sent}(1, I, B, m_I) :: m_I :: I :: A :: B :: 1 :: \mathsf{nil}\big)\\
&\qquad \text{let } \ell = \mathsf{sent}(1, I, B, m_I) :: m_I :: I :: A :: B :: 1 :: \mathsf{nil}\\
&\xrightarrow[(\text{DS-B})]{} \mathsf{trace}\big(0, \mathsf{sent}(2, B, \mathsf{fst}(m_I), \mathsf{enc}(S,\\
&\qquad \mathsf{adec}(\mathsf{adec}(\mathsf{snd}(m_I), \mathsf{inv}(\mathsf{pub}(B))), \mathsf{pub}(\mathsf{fst}(m_I))))) :: \ell\big)\\
&\xrightarrow[\mathcal{R}_{\mathcal{C}}]{*} \mathsf{trace}\big(0, \mathsf{sent}(2, B, A, m_B) :: \ell\big)\\
&\xrightarrow[(7)]{} \mathsf{sent}(2, B, A, m_B) :: \ell
\end{aligned}
$$

Figure 3: An authentication attack on DS protocol

this instance is a counterexample for conjecture (C_{auth}), with the normalization with $\mathcal{R}$ presented in Figure 3, where $m_I = \mathsf{pair}(A, \mathsf{aenc}(A, \mathsf{pub}(B))))$ and $m_B = \mathsf{sent}(2, B, A, \mathsf{enc}(S, \mathsf{adec}(A, \mathsf{pub}(A))))$ (A, B and I are arbitrary distinct constructor terms of sort Name). The normalization of Figure 3 indicates quite legibly an authentication attack on the protocol. In the first steps of the reduction, the attacker builds a message $\mathsf{sent}(1, I, B, m_I)$ using its initial knowledge (generated by $\mathcal{G}$ from the no-terminal Init), with rule (att-init) and some public constructor symbols, with rule (att-anlz). Then B reads this message and believes that it originated from A. He therefore sends to A an answer which is reduced by $\mathcal{R}_{\mathcal{C}}$ into: $\mathsf{sent}(2, B, A, m_B)$. Hence, the list obtained after the reduction in Figure 3 belongs to $L(\mathcal{G}, B^{\mathrm{DS}}_{\mathsf{auth}})$, since this message $\mathsf{sent}(2, B, A, m_B)$ is not preceded in the list by a message of the form $\mathsf{sent}(1, A, B, \ldots)$ (this indicates the authentication flaw). It means that the instance of conjecture (C_{auth}) is reduced to a clause of the form $\mathsf{sent}(2, B, A, m_B) :: \ell \neq \mathsf{sent}(2, B, A, m_B) :: \ell$, which leads to a case of **disproof**.

For Conjecture (C_{sec}), we consider now $n = s^{12}(0)$ and the reduction in Figure 5. The first steps of this figure are the same as in Figure 3. In the next steps, the attacker builds (with rules (att-init) and (att-anlz)) a fake key $\mathsf{adec}(A, \mathsf{pub}(A))$ which he uses latter in order to decipher the message m_B from B and recover S. Hence, the lists obtained in Figure 5 belongs to $L(\mathcal{G}, B_{\mathsf{sec}})$. It means that the instance of conjecture (C_{sec}) are reduced to a clause $\ell'' \neq \ell''$, which leads to a case of **disproof**.

5 Shamir-Rivest-Adleman Three Pass Protocol

We consider the same signature as in Section 4 and also the same constructor CTRS $\mathcal{R}_{\mathcal{C}}$ extended with the following commutativity-like rule for the

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathsf{trace}(12,\mathsf{nil}) \xrightarrow{*} \mathsf{trace}\big(4, \mathsf{sent}(2,B,A,m_B) :: \ell\big)} \\
\xrightarrow[(\mathsf{att\text{-}anlz})]{} & \mathsf{trace}\big(3, \mathsf{body}(\mathsf{sent}(2,B,A,m_B)) :: \mathsf{sent}(2,B,A,m_B) :: \ell\big) \\
\xrightarrow[\mathcal{R}_{\mathcal{C}}]{} & \mathsf{trace}\big(3, m_B :: \mathsf{sent}(2,B,A,m_B) :: \ell\big) \\
 & \text{let } \ell' = m_B :: \mathsf{sent}(2,B,A,m_B) :: \ell \\
\xrightarrow[(\mathsf{att\text{-}anlz})]{*} & \mathsf{trace}\big(1, \mathsf{adec}(A,\mathsf{pub}(A)) :: \ell'\big) \\
\xrightarrow[(\mathsf{att\text{-}anlz})]{} & \mathsf{trace}\big(0, \mathsf{dec}(m_B, \mathsf{adec}(A,\mathsf{pub}(A))) :: \mathsf{adec}(A,\mathsf{pub}(A)) :: \ell'\big) \\
\xrightarrow[\mathcal{R}_{\mathcal{C}}]{*} & \mathsf{trace}\big(0, S :: \mathsf{dec}(m_B, \mathsf{adec}(A,\mathsf{pub}(A))) :: \mathsf{adec}(A,\mathsf{pub}(A)) :: \ell'\big) \\
\xrightarrow[(7)]{} & S :: \mathsf{dec}(m_B, \mathsf{adec}(A,\mathsf{pub}(A))) :: \mathsf{adec}(A,\mathsf{pub}(A)) :: \ell' = \ell''
\end{array}
$$

Figure 4: An attack on the secrecy of S for DS protocol

encryption operator:

$$\mathsf{aenc}(\mathsf{aenc}(x,k_1),k_2) = \mathsf{aenc}(\mathsf{aenc}(x,k_2),k_1)\ [\![k_1 > k_2]\!] \qquad (8)$$

The protocol runs between two agents x_a and x_b in 3 pass, which are described by the following 3 CCTRS rules of $\mathcal{R}_{\mathcal{D}}$.

$$\mathsf{trace}(s(n),y) = \mathsf{trace}\big(n, \mathsf{sent}(1,x_a,x_b,\mathsf{pair}(x_a,\mathsf{aenc}(S,\mathsf{key}(x_a)))) :: y\big)$$
$$[\![x_a, x_b : \mathsf{Name}, x_a \neq x_b]\!] \quad (\mathsf{RSA\text{-}A1})$$

Initially, the honest agent x_a sends to x_b a message containing a secret value S encrypted with its own public key $\mathsf{pub}(x_a)$. The ciphertext is sent in a pair, along with the identity of x_a.

$$\mathsf{sent}(x'_a,x_b,x) \in y = \mathsf{true} \Rightarrow$$
$$\mathsf{trace}(s(n),y) = \mathsf{trace}\big(n, \mathsf{sent}(2,x_b,\mathsf{fst}(x),\mathsf{aenc}(\mathsf{snd}(x),\mathsf{key}(x_b))) :: y\big) \quad (\mathsf{RSA\text{-}B})$$

While reading a message x from x'_a, x_b is not able to decipher it and recover S, because he does not know the private key of A. Instead, he sends an answer obtained by encrypting again the message read with its public key $\mathsf{key}(x_b)$. The message sent by x_b has hence the form: $\mathsf{aenc}\big(\mathsf{aenc}(S,\mathsf{key}(x_a)),\mathsf{key}(x_b)\big)$ which, by the rule (8) of $\mathcal{R}_{\mathcal{C}}$ is equivalent to $\mathsf{aenc}\big(\mathsf{aenc}(S,\mathsf{key}(x_b)),\mathsf{key}(x_a)\big)$.

$$\mathsf{sent}(x_a,x_b,x) \in y = \mathsf{true}, \mathsf{sent}(x'_b,x_a,x) \in y = \mathsf{true} \Rightarrow$$
$$\mathsf{trace}(s(n),y) = \mathsf{trace}\big(n, \mathsf{sent}(3,x_a,x_b,\mathsf{adec}(x,\mathsf{inv}(\mathsf{key}(x_a)))) :: y\big) \quad (\mathsf{RSA\text{-}A2})$$

After reading the message from x_b, x_a decrypts it with its private key $\mathsf{inv}(\mathsf{key}(x_a))$, and send the result $\mathsf{aenc}(S, \mathsf{key}(x_b))$. Then, B is able to decipher this message and recover S.

We consider the same rules of $\mathcal{R}_\mathcal{D}$ for $\in$, trace (7) and for the attacker (att-init and att-anlz) as in Section 4. We assume moreover here that the language generated by the non-terminal Init (initial knowledge of the attacker) contains the private key $\mathsf{inv}(\mathsf{key}(I))$.

Like in Section 4, we construct the constrained tree grammar $\mathcal{G}$ by intersection of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ with a regular tree grammar $\mathcal{B}ad$. We consider again the non-terminal B_{sec} which generate the set of lists containing the constant S (like in Section 4) and another non-terminal $B^{\mathrm{RSA}}_{\mathsf{auth}}$ for authentication flaws, which generates the lists containing a message of the form $\mathsf{sent}(1, A, B, \ldots)$ followed by a message $\mathsf{sent}(3, A, B, \ldots)$, without a message of the form $\mathsf{sent}(2, B, A, \ldots)$ in between. The conjectures are the same as in Section 4, (C_{auth} and C_{sec}), with $B^{\mathrm{RSA}}_{\mathsf{auth}}$ instead of $B^{\mathrm{DS}}_{\mathsf{auth}}$ in (C_{auth}).

Again, we show with our procedure that these conjectures are not joinable inductive theorems of $\mathcal{R}$ (by induction on traces), revealing known authentication and secrecy flaws of the protocol.

Let us consider an instance of Conjecture (C_{sec}) where y is replaced by nil and n is replaced by 11. The reductions in Figure 5 shows that it is a counter example and hence that the protocol has a secrecy flaw. The first part of the reduction suggests also a counter-example for Conjecture (C_{auth}), demonstrating an authentication flaw of the protocol.

6 Towards Joinable Inductive Validation of Protocols

Let us modify the protocol rules of Section 4 in order to fix the above attacks. We add a $\mathsf{pair}(x_a, x_b)$ along with the key K in the first message:

$$\mathsf{trace}(s(n), y) \to \mathsf{trace}\big(n, \mathsf{sent}(1, x_a, x_b, \mathsf{pair}(x_a, \mathsf{aenc}(\mathsf{aenc}(\mathsf{pair}(\mathsf{pair}(x_a, x_b), K), \\ \mathsf{inv}(\mathsf{pub}(x_a))), \mathsf{pub}(x_b)))) :: y\big) \; [\![x_a : \mathsf{Name}, x_b : \mathsf{Name}, x_a \neq x_b]\!] \quad \text{(DS-A')}$$

Before sending the second message, x_b checks first the pair $\mathsf{pair}(x_a, x_b)$ sent in the ciphertext (we let $k = \mathsf{adec}(\mathsf{adec}(\mathsf{snd}(x_m), \mathsf{inv}(\mathsf{pub}(x_b))), \mathsf{pub}(\mathsf{fst}(x_m)))$):

$$\mathsf{sent}(1, x'_a, x_b, x_m) \in y = \mathsf{true}, \mathsf{snd}(\mathsf{fst}(k)) = x_b, \mathsf{fst}(\mathsf{fst}(k)) = \mathsf{fst}(x_m) \Rightarrow \\ \mathsf{trace}(s(n), y) \to \mathsf{trace}(n, \mathsf{sent}(2, x_b, \mathsf{fst}(x_m), \mathsf{enc}(S, k))) :: y) \quad \text{(DS-B')}$$

We present below some parts of the validation of the amended version of the protocol with our procedure, i.e. the proof that the conjecture C_{sec} is a joinable inductive theorem of the above specification. The proof is much more difficult than in the previous sections. Indeed, we need here to verify all the execution traces in order to validate the protocol, since $\mathcal{R}$ is not ground-confluent (by definition of *joinable* inductive theorems). In comparison, it is sufficient to

$$
\begin{aligned}
\mathsf{trace}(11,\mathsf{nil}) &\xrightarrow[(\text{RSA-A1})]{} \mathsf{trace}\big(10,\mathsf{sent}(1,A,B,\mathsf{pair}(A,\mathsf{aenc}(S,\mathsf{key}(A))))::\mathsf{nil}\big)\\
&\xrightarrow[(\text{att-anlz})]{} \mathsf{trace}\big(9,\mathsf{body}(\mathsf{sent}(1,A,B,\mathsf{pair}(A,\mathsf{aenc}(S,\mathsf{key}(A)))))::\ldots\big)\\
&\xrightarrow[\mathcal{R}_{\mathcal{C}}]{} \mathsf{trace}\big(9,\mathsf{pair}(A,\mathsf{aenc}(S,\mathsf{key}(A)))::\ldots\big)\\
&\xrightarrow[(\text{att-anlz})]{} \mathsf{trace}\big(8,\mathsf{fst}(\mathsf{pair}(A,\mathsf{aenc}(S,\mathsf{key}(A))))::\ldots\big)\\
&\xrightarrow[\mathcal{R}_{\mathcal{C}}]{} \mathsf{trace}\big(8,\mathsf{aenc}(S,\mathsf{key}(A))::\ldots\big)\\
&\xrightarrow[(\text{att-init})]{} \mathsf{trace}\big(7,I::\mathsf{aenc}(S,\mathsf{key}(A))::\ldots\big)\\
&\xrightarrow[(\text{att-anlz})]{} \mathsf{trace}\big(6,\mathsf{key}(I)::I::\mathsf{aenc}(S,\mathsf{key}(A))::\ldots\big)\\
&\xrightarrow[(\text{att-anlz})]{} \mathsf{trace}\big(5,\mathsf{aenc}(\mathsf{aenc}(S,\mathsf{key}(A)),\mathsf{key}(I))::\ldots\big)\\
&\xrightarrow[\mathcal{R}_{\mathcal{C}}]{} \mathsf{trace}\big(5,\mathsf{aenc}(\mathsf{aenc}(S,\mathsf{key}(I)),\mathsf{key}(A))::\ldots\big)\\
&\xrightarrow[(\text{att-anlz})]{} \mathsf{trace}\big(4,\mathsf{sent}(2,I,A,\mathsf{aenc}(\mathsf{aenc}(S,\mathsf{key}(I)),\mathsf{key}(A)))::\ldots\big)\\
&\xrightarrow[(\text{RSA-A2})]{} \mathsf{trace}\big(3,\mathsf{sent}(3,A,B,\\
&\qquad \mathsf{adec}(\mathsf{aenc}(\mathsf{aenc}(S,\mathsf{key}(I)),\mathsf{key}(A)),\mathsf{inv}(\mathsf{key}(A))))::\ldots\big)\\
&\xrightarrow[\mathcal{R}_{\mathcal{C}}]{} \mathsf{trace}\big(3,\mathsf{sent}(3,A,B,\mathsf{aenc}(S,\mathsf{key}(I)))::\ldots\big)\\
&\xrightarrow[(\text{att-anlz})]{} \mathsf{trace}\big(2,\mathsf{body}(\mathsf{sent}(3,A,B,\mathsf{aenc}(S,\mathsf{key}(I))))::\ldots\big)\\
&\xrightarrow[\mathcal{R}_{\mathcal{C}}]{} \mathsf{trace}\big(2,\mathsf{aenc}(S,\mathsf{key}(I))::\ldots\big)\\
&\xrightarrow[(\text{att-init})]{} \mathsf{trace}\big(1,\mathsf{inv}(\mathsf{key}(I))::\mathsf{aenc}(S,\mathsf{key}(I))::\ldots\big)\\
&\xrightarrow[(\text{att-anlz})]{} \mathsf{trace}\big(0,\mathsf{adec}(\mathsf{aenc}(S,\mathsf{key}(I)),\mathsf{inv}(\mathsf{key}(I)))::\ldots\big)\\
&\xrightarrow[\mathcal{R}_{\mathcal{C}}]{} \mathsf{trace}\big(0,S::\ldots\big)\\
&\xrightarrow[(7)]{} S::\ldots
\end{aligned}
$$

Figure 5: An attack on the secrecy of S for RSA protocol

find one erroneous trace in order to show that the protocol is flawed.

The application of the procedure generates several subgoals, amongst them:

$$y \neq \mathsf{nil}\,[\![y:\llcorner y\lrcorner^{\mathsf{List}}\cap B\mathsf{sec}]\!]$$

$$y \neq \mathsf{trace}(n,x::\mathsf{nil})\;[\![y:\llcorner y\lrcorner^{\mathsf{List}}\cap B_{\mathsf{sec}},\,n:\llcorner x\lrcorner^{\mathsf{Nat}},\,x:\mathsf{Init}]\!]$$

$$x_1\in y=\mathsf{true},\ldots,x_k\in y=\mathsf{true}\Rightarrow y\neq\mathsf{trace}(n,f(x_1,\ldots,x_k))::\mathsf{nil})$$
$$[\![y:\llcorner y\lrcorner^{\mathsf{List}}\cap B^{\mathsf{DS}}_{\mathsf{sec}},\,n:\llcorner x\lrcorner^{Nat}]\!]$$

Let us recall that Init is a non-terminal of a regular tree grammar generating the language of the initial knowledge of the attacker. This language contains all the ground constructor terms of sort Name and Id in our example. In the third subgoal, f denotes any public constructor symbol of arity k. In our example, $k = 1$ and f is pub, fst, snd, body or $k = 2$ and and f is pair, enc, dec, aenc, adec.

The proof of the first subgoal is immediate, but the other subgoals need more developments and the interactive addition of some lemmas in order to derive a proof. We are working on an extension of our inference system with new simplification rules in order to avoid the divergence during the validation of correct authentication protocols.

References

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 104–115, 2001.

[2] A. Bouhoula. Automated theorem proving by test set induction. *Journal of Symbolic Computation*, 23(1):47–77, 1997.

[3] A. Bouhoula and F. Jacquemard. Automated induction for complex data structures. Research Report LSV-05-11, Laboratoire Spécification et Vérification, ENS Cachan, France, July 2005.

[4] A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 1995.

[5] A. Bundy and G. Steel. Attacking group protocols by refuting incorrect inductive conjectures. *Journal of Automated Reasoning*, vol. 36, numbers 1-2, pages 149-176. January 2006.

[6] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. http://www.grappa.univ-lille3.fr/tata, 2002.

[7] H. Comon and F. Jacquemard. Ground reducibility is exptime-complete. *Information and Computation*, 187(1):123–153, 2003.

[8] H. Comon-Lundh. *Handbook of Automated Reasoning*, chapter Inductionless Induction. Elsevier, 2001.

[9] D. E. Denning and G. M. Sacco. Timestamps in Key Distribution Protocols. In *Communications of the ACM*, 1981.

[10] N. Dershowitz and J.-P. Jouannaud. *Rewrite systems*, chapter Handbook of Theoretical Computer Science, Volume B, pages 243–320. Elsevier, 1990.

[11] C. Lynch and C. Meadows. On the relative soundness of the free algebra model for public key encryption. *Electr. Notes Theor. Comput. Sci.*, 125(1):43–54, 2005.

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