Verifying Regular Trace Properties of Security Protocols with Explicit Destructors and Implicit Induction

Adel Bouhoula, Florent Jacquemard

Verifying Regular Trace Properties of Security Protocols with Explicit Destructors and Implicit Induction*

Adel Bouhoula[1] and Florent Jacquemard[2]

[1] École Supérieure des Communications de Tunis, Tunisia. bouhoula@planet.tn
[2] INRIA Futurs & LSV UMR CNRS–ENSC, France. florent.jacquemard@inria.fr

Abstract. We present a procedure for the verification of cryptographic protocols based on a new method for automatic implicit induction theorem proving for specifications made of conditional and constrained rewrite rules. The method handles axioms between constructor terms which are used to introduce explicit destructor symbols for the specification of cryptographic operators. Moreover, it can deal with non-confluent rewrite systems. This is required in the context of the verification of security protocols because of the non-deterministic behavior of attackers. Our induction method makes an intensive use of constrained tree grammars, which are used in proofs both as induction schemes and as oracles for checking validity and redundancy criteria by reduction to an emptiness problem. The grammars make possible the development of a generic framework for the specification and verification of protocols, where the specifications can be parametrized with (possibly infinite) regular sets of user names or attacker's initial knowledge and complex security properties can be expressed, referring to some fixed regular sets of bad traces representing potential vulnerabilities. We present some case studies giving very promising results, for the detection of attacks (our procedure is complete for refutation), and also for the validation of protocols.

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 [15] has been applied to many case studies. In 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 also a good expertise (if a proof fails with Isabelle, it is difficult to conclude whether the proof attempt failed or the conjecture to be proved is not valid).

* This work has been partially supported by the grant INRIA–DGRSRT 06/I09.

Bundy and Steel [8] 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* [11] 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 and convergence is difficult to achieve.

In this paper we present a new method for the formal verification of security protocols based on an *implicit induction* procedure. The protocol, the insecure communication network (attackers) and the security assumptions are modeled with an equational specification which is passed to an inductive theorem prover in order to validate the protocol or to derive an attack. The advantage of this procedure is that it is automatic and returns readable proofs or counter-examples.

The specifications are strongly typed and follow a constructor discipline: we distinguish in the signature the *constructor symbols*, used to build terms representing the values of the computation, in our case the list (trace) of messages exchanged. The other symbols, called *defined symbols*, represent functions defined on these values (for instance, we use below a predicate trace characterising the protocol traces) and are specified by Horn clauses.

Equational axioms between terms made of constructors are very difficult to deal with in automated induction and are generally not allowed. Our procedure however allows such axioms, and we use them 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]. Explicit destructors both simplify the specification of attacker's capabilities and increase the expressiveness of specification as models with explicit destructors are strictly more expressive than models based on free algebra, in the sense that they captures more attacks [14].

Our induction procedure introduces another important novelty, compared to other implicit induction techniques, since it handles specifications 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 users. Such a model, relevant in the context of security, can not be expressed with a ground-confluent specification.

The axioms of the specification contain constraints such as equations, disequations and membership to fixed term languages (characterized by tree grammars). The membership constraints come almost for free as our induction procedure is based on constrained tree grammars. This feature appeared however extremely useful in a setting of protocol specification. It permits to parametrize the specifications of protocols and attackers with (possibly infinite) regular sets of user names or attacker's initial knowledge. Moreover, and more important, it also allows to deal with a rich language of security properties, not limited to the confidentiality of some fixed piece of data as it is the case in many approaches.

Indeed, assume given a set $\mathcal{B}$ of *bad traces* (lists of messages exchanged corresponding to an attack) which is a regular tree language (of constructor terms), characterized by a tree grammar. We can express as a conjecture to be proved the property that every protocol trace (as defined by the predicate trace) does not belong to $\mathcal{B}$. This allows to write a wide range of security properties, like for instance variants of authenticity.

In contrast to the technique of [8], 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 permits in particular to focus on traces of events in normal form, and consequently to minimize the set of traces to be checked. Since our procedure is refutationally complete (under some conditions for the specification) its application on any flawed protocol will return a readable attack in finite (and typically very small) time and in a completely automatic way, as illustrated by the examples of Section 3. Moreover, it can also help in protocol validation (proof that there is no attack), though the interactive addition of lemmas may be required for that purpose, see the example in Section 4.

2 Preliminaries

We consider a many-sorted signature $\mathcal{F}$. As explained in introduction, $\mathcal{F}$ is partitioned into a subset $\mathcal{C}$ of *constructor symbols* and a subset $\mathcal{D}$ of *defined symbols*. Each symbol f is given with a profile $f : S_1 \times \ldots \times S_n \to S$ where $S_1, \ldots, S_n, S$ are sorts and n is the *arity* of f. We note $\mathcal{T}(\mathcal{F}, \mathcal{X})$ (resp. $\mathcal{T}(\mathcal{C}, \mathcal{X})$) the set of well-sorted terms over $\mathcal{F}$ (resp. constructor well-sorted terms) with variables in $\mathcal{X}$ and $\mathcal{T}(\mathcal{F})$ (resp. $\mathcal{T}(\mathcal{C})$) and the subsets of variable-free terms, or *ground* terms. 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.

Conditional constrained rewriting. We shall consider below *constraints* which are Boolean combinations of atoms of the form $P(t_1, \ldots, t_n)$ where P belongs to a fixed language of constraints predicates interpreted over terms of $\mathcal{T}(\mathcal{C})$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C}, \mathcal{X})$. The *solutions* of a constraint c, whose set is denoted $sol(c)$, are (constructor) substitutions σ grounding for all terms in c and such that $c\sigma$ is interpreted to true. *Constrained terms* have the form $t \,[\![c]\!]$ where $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and c is a constraint and *conditional constrained rewrite rules* are constrained Horn clauses such as:

$$u_1 = v_1, \ldots, u_n = v_n \Rightarrow \ell \to r \,[\![c]\!]$$

where $u_1, v_1, \ldots, u_n, v_n, \ell, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, the terms ℓ and r (called resp. left- and right-hand side of the rule, $\ell \to r$ is an oriented equation) are linear[3] and have the same sort, and c is a constraint. Our procedure takes as input a constrained constructor rewrite system (CTRS) $\mathcal{R}_\mathcal{C}$, which is a set of rules such that $n = 0$ (no conditions) and $\ell, r \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ and a conditional constrained rewrite system (CCTRS) $\mathcal{R}_\mathcal{D}$, with rules such that $n \geq 0$, $\ell = f(\ell_1, \ldots, \ell_k)$ where $f \in \mathcal{D}$ and $\ell_1, \ldots, \ell_n, r \in \mathcal{T}(\mathcal{C}, \mathcal{X})$. We note $\mathcal{R} = \mathcal{R}_\mathcal{C} \uplus \mathcal{R}_\mathcal{D}$.

A term $t\,[\![d]\!]$ rewrites to $s\,[\![d]\!]$ by the above rule, 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 $i \in [1..n]$; where $\downarrow_\mathcal{R}$ denotes $\xrightarrow[\mathcal{R}]{*} \circ \xleftarrow[\mathcal{R}]{*}$ and $\xrightarrow[\mathcal{R}]{*}$ is the reflexive transitive closure of $\xrightarrow[\mathcal{R}]{}$. If there exists such a term s, then $t\,[\![d]\!]$ is called *reducible*, otherwise it is called a *normal form*. A constrained term $t\,[\![c]\!]$ 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 $\mathcal{R}$.

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.

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 $\mathcal{R} \models_{jind} a = b\,[\![c]\!]$) 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 3 and 4.

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 $a = b$ is an inductive theorem (since $a \xleftrightarrow[\mathcal{R}]{*} b$) but it is not a joinable inductive theorem (as a and b are $\mathcal{R}$-normal forms). On the other hand, $a = b \Rightarrow f(x) = c$ is a joinable inductive theorem but not an inductive theorem (for the same reasons).

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

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 $\llcorner t \lrcorner := f(\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner)\,[\![c]\!]$ where $f \in \mathcal{F}$, $\llcorner t \lrcorner$, $\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner \in Q$ and c is a constraint.

[3] Note that assuming that l and r are linear is not restrictive since non linearities may be expressed as equalities between variables in c.

$$\mathsf{fst}(\mathsf{pair}(x_1,x_2)) \to x_1 \qquad \mathsf{snd}(\mathsf{pair}(x_1,x_2)) \to x_2 \qquad \mathsf{inv}(\mathsf{inv}(y)) \to y$$
$$\mathsf{dec}(\mathsf{enc}(x,y),y) \to x \quad \mathsf{adec}(\mathsf{aenc}(x,y),\mathsf{inv}(y)) \to x \quad \mathsf{adec}(\mathsf{aenc}(x,\mathsf{inv}(y)),y) \to x$$

Figure 1: Constructor rules

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 $\vdash^x_{\mathcal{G}}$ on constrained terms is defined by:

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

if there exists $\llcorner u \lrcorner := f(\llcorner u_1 \lrcorner,\ldots,\llcorner u_n \lrcorner)\,[\![c]\!] \in \Delta$ 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 variable x, constrained to be in the language defined by the non-terminal $\llcorner u \lrcorner$, is replaced by $f(x_1,\ldots,x_n)$ where $x_1,\ldots,x_n$ are constrained to the respective languages of $\llcorner u_1 \lrcorner,\ldots,\llcorner u_n \lrcorner$. 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 $L(\mathcal{G},\llcorner u \lrcorner)$ 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 shall use below such membership constraints in order to restrict a term to a given sort or a given regular tree language.

3 Verification of a Key Distribution Protocol

In this section, we describe in an example how to specify a protocol, its environment and security properties with conditional constrained rewrite rules, and how the implicit induction procedure of Section 5 can be applied to the verification of the security properties, expressed as a joinable inductive conjectures.

Signature. Assume some sorts Nat, Bool, Name, Id, Key, Msg, MsgList, with the subsort relations: Name $\subseteq$ Msg and Key $\subseteq$ Msg. The messages exchanged during the protocol execution are abstracted by well sorted terms built with constructor symbols pair : Msg $\times$ Msg $\to$ Msg, and projections fst, snd : Msg $\to$ Msg, encryption and decryption in symmetric and asymmetric key cryptography enc, aenc, dec, adec, all with profile Msg $\times$ Msg $\to$ Msg and which follow the rules in Figure 1. The variables x represents the encrypted plaintext and the y is a symmetric or a public encryption key. The idempotent operator inv : Key $\to$ Key associates to a public key its corresponding private key (for decryption), and conversely; We assume moreover an operator pub : Name $\to$ Key which associates to the identity of a user its public key. The symbol inv is called *secret* and all the others symbols are called *public*.

Let us also consider a public constructor $\mathsf{sent} : \mathsf{Id} \times \mathsf{Name} \times \mathsf{Name} \times \mathsf{Msg} \to \mathsf{Msg}$ used 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 $\mathsf{body} : \mathsf{Msg} \to \mathsf{Msg}$ can be used for removing the header, with the rule: $\mathsf{body}\big(\mathsf{sent}(x_i, x_a, x_b, x)\big) \to x$.

We assume moreover some additional secret constructors for Boolean: $\mathsf{true}, \mathsf{false} : \mathsf{Bool}$, for natural numbers $0 : \mathsf{Nat}$, $s : \mathsf{Nat} \to \mathsf{Nat}$, for lists of messages, $\mathsf{nil} : \mathsf{MsgList}$, $:: : \mathsf{Msg} \times \mathsf{MsgList} \to \mathsf{MsgList}$ and constant values used in the protocol messages: $K : \mathsf{Key}$, $S : \mathsf{Msg}$. Finally, we assume that the set of names of honest users (*i.e.* the set of terms of sort Name) is a (possibly infinite) regular tree set[4] whose terms are made only of public constructor symbols.

Let us denote $\mathcal{R}_{\mathcal{C}}$ the set of rewrite rules given above, which are sometimes referred as *explicit destructors* rules in the protocol verification literature. We propose in Appendix A a constrained tree grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ which generates the constructor normal forms. It contains in particular the non terminals $\llcorner y \lrcorner^{\mathsf{List}}$ and $\llcorner x \lrcorner^{\mathsf{Nat}}$ generating normal forms of respective sorts List and Nat.

Protocol. We consider a simplification (without certificates and timestamps) of a key distribution protocol of Denning & Sacco [13] for a symmetric key exchange in an asymmetric cryptosystem. Following the approach of [15], we consider traces of messages modelled as lists (built with nil and $::$) and characterized by the defined symbol $\mathsf{trace} : \mathsf{Int} \times \mathsf{MsgList} \to \mathsf{Msg}$. In $\mathsf{trace}(n, \ell)$, ℓ is a list of messages exchanged (protocol trace) and n can be seen as a resource consumed by each operation executed by an honest or dishonest agent. As we shall see below (in the description of the conjectures proved), the principle of our verification method is to perform an induction on the initial value of this resource (at the beginning of the protocol).

The symbol trace is defined recursively by extension with messages sent by the users participating to the protocol (honest or not). In the case of the Denning & Sacco protocol, the conditional rule (DS-A) of $\mathcal{R}_{\mathcal{D}}$ describes the user x_a sending to user x_b a message with identifier 1, which contains a freshly chosen symmetric key K for further secure communications:

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

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 then 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 pair) so that the receiver x_b knows which public key to use in order to recover K.

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

In the second conditional rule (DS-B) of $\mathcal{R}_\mathcal{D}$, the honest user x_b, while reading a message x_m, expects that x_m has the above form (though he does not check this) and extracts the symmetric key K, applying twice the asymmetric decryption function adec to the second component of x_m, obtained by application of the projection function snd. This key K is then used by x_b to encrypt (with the function enc) a secret code S that he wants to communicate to the user x_a, and this ciphertext is sent in a message with identifier 2.

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

Attacker. We assume asynchronous communication of the messages through an insecure public network controlled by a dishonest user 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 $\mathcal{R}_\mathcal{C}$. This makes the framework with explicit destructors more uniform that others (often called "Dolev-Yao" models) where information extraction is modeled with extra ad-hoc inference rules. Note that we do not need here the extra predicates analyze and synthesis of [15] for the specification of trace. Besides uniformity, the addition of explicit destructor rules makes the model strictly more expressive, in the sense that it captures strictly more attacks, like for instance the attack described below. The operations of the attacker are specified by the following rules of $\mathcal{R}_\mathcal{D}$ for the specification of trace:

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

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

In the rule (att-init), Init is an extra non-terminal of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ 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. In the rule (att-anlz), $x_i \in y$ has to be read as $x_i \in y = \mathsf{true}$. This rule actually represents one conditional rule for each public constructor symbol f of arity k.

The function $\in$: Msg $\times$ MsgList $\rightarrow$ Bool is defined by the three rules of $\mathcal{R}_\mathcal{D}$:

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

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

Security properties. We construct a tree grammar $\mathcal{G}$ by intersection of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ and a regular tree grammar $\mathcal{B}ad$ which generates 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 particular non-terminal in the grammar $\mathcal{B}ad$:

– a non-terminal $B^{\mathrm{DS}}_{\mathsf{auth}}$ 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 a *secrecy* flaw: it indicates that the secret value S is publicly revealed.

The two conjectures (C_{auth}), (C_{sec}) express that every bad trace y cannot be a trace of the protocol obtained in n steps, for any n:

$$\mathsf{trace}(n, \mathsf{nil}) \neq \mathsf{trace}(0, y)\, [\![y : {\llcorner} y^{\mathsf{List}} {\lrcorner} \cap B^{\mathrm{DS}}_{\mathsf{auth}}, n : {\llcorner} x^{\mathsf{Nat}} {\lrcorner}]\!] \qquad (C_{\mathsf{auth}})$$

$$\mathsf{trace}(n, \mathsf{nil}) \neq \mathsf{trace}(0, y)\, [\![y : {\llcorner} y^{\mathsf{List}} {\lrcorner} \cap B_{\mathsf{sec}}, n : {\llcorner} x^{\mathsf{Nat}} {\lrcorner}]\!] \qquad (C_{\mathsf{sec}})$$

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 $\mathsf{trace}(n, \mathsf{nil}) \neq \mathsf{trace}(0, y)$ should be understood as $\mathsf{trace}(n, \mathsf{nil}) = \mathsf{trace}(0, y) \Rightarrow \mathsf{true} = \mathsf{false}$. 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 with the non-terminals ${\llcorner} y^{\mathsf{List}} {\lrcorner}$ and ${\llcorner} x^{\mathsf{Nat}} {\lrcorner}$, and y is moreover constrained to be instantiated by terms generated by $\mathcal{B}ad$.

Note that (C_{auth}) and (C_{sec}) have the same form, only the regular tree grammar $\mathcal{B}ad$ differs. In general, with this approach, we can express that there is no intersection between protocol traces and bad traces for any regular set of bad traces, which makes quite a rich language of security properties.

Disproofs. The application of our procedure shows that none of the conjectures is a joinable inductive theorems of $\mathcal{R}$, by induction on traces, revealing attacks on the protocol.
Among the instances of the conjecture (C_{auth}) generated[5] 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 this instance is a counterexample for the conjecture (C_{auth}), with the normalization with $\mathcal{R}$ presented in Figure 2, 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 2 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 ($\mathsf{att\text{-}init}$) and some public constructor symbols, with rule ($\mathsf{att\text{-}anlz}$). 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 2 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

[5] The procedure generates all the instances which are smaller than $d(\mathcal{R})$ (the maximum depth of the left-hand sides of rules of $\mathcal{R}$), see Section 5.

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

Figure 2: An authentication attack on DS protocol

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

Figure 3: An attack on the secrecy of S for DS protocol

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 3. The first steps of this figure are the same as in Figure 2. 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 3 belongs to $L(\mathcal{G}, B_{\mathsf{sec}})$. It means that the instance of conjecture (C_{sec}) are reduced to a clause $\mathsf{trace}(0, \ell'') \neq \mathsf{trace}(0, \ell'')$, which leads to a case of **disproof**.

Shamir-Rivest-Adleman Three Pass Protocol. Another example of derivation of an attack is proposed in [6] (see also Appendix B and Figure 6). We won't reproduce it in details here. We would just like to outline one interesting use of constraints in explicit destructor axioms in this example. Indeed, the protocol RSA 3-pass relies on a commutativity-like property for the encryption operator. In our model, it is expressed by the following rule of $\mathcal{R}_\mathcal{C}$:

$$\mathsf{aenc}(\mathsf{aenc}(x, k_1), k_2) = \mathsf{aenc}(\mathsf{aenc}(x, k_2), k_1) \, [\![k_1 > k_2]\!] \tag{1}$$

Note the addition of the ordering constraint, for termination purposes.

4 Towards Joinable Inductive Validation of Protocols

Let us modify the protocol rules of Section 3 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 (by definition of *joinable* inductive theorems), since $\mathcal{R}$ is not ground-confluent. In comparison, it is sufficient to find one erroneous trace in order to show that the protocol is flawed.

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

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

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.

5 Implicit Inductive Theorem Proving procedure

We present in this section a goal-directed inductive theorem proving procedure for conditional and constrained specifications.

This procedure belongs to the family of *implicit induction* (in the lines of [7]) and combines the power of two classical methods for automatic induction: *explicit*

induction and *proof by consistency* [11]. As outlined above, the procedure supports features which are generally not found in former inductive theorem proving approaches, like handling non ground-confluent rewrite systems, axioms between constructors (used here for specifying explicit destructors) or the parametrization of the specification and conjectures with given regular set of terms. A key for these characteristics is that 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 [7, 4] permits precisely to handle constrained rewrite rules between constructors, (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 terms representing specific values or traces, see Section 3.

5.1 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.* [11]. 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 [5] and exemplified in the Section 3, 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. 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, (see Section 3). The intersection between a constrained tree grammar and a regular tree grammar is a constrained tree grammar.

We call a constrained term $t \,[\![c]\!]$ *decorated* if $c = x_1{:}\llcorner u_1 \lrcorner \wedge \ldots \wedge x_n{:}\llcorner u_n \lrcorner \wedge d$, $\{x_1, \ldots, x_n\} = var(t)$, $\llcorner u_i \lrcorner \in Q$ and $sort(u_i) = sort(x_i)$ 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 [5] 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 $\mathcal{R} \models_{jind} a = b \,[\![c]\!]$

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 [9], some restrictions on $\mathcal{R}_\mathcal{C}$ are given [5] which ensure the decidability of the three above problem. The rewrite systems described in Sections 3 and 4 fulfill these restrictions.

5.2 Simplification Rules

We present in Figure 4 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 5.1, 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 for all $i \leq k$, $D_i \,[\![c]\!] \ll C \,[\![c]\!]$ where $\{D_1 \,[\![c]\!], \ldots, D_k \,[\![c]\!]\}$ are all the clauses obtained by one-step rewriting with $\mathcal{R}$ from $C \,[\![c]\!]$.

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

if $\mathcal{R} \models_{jind} \Gamma_1\sigma_1 \,[\![c \wedge c_1\sigma_1]\!] \vee \ldots \vee \Gamma_n\sigma_n \,[\![c \wedge c_n\sigma_n]\!]$, $C|_{p_i} > r_i\sigma_i$ and $\{C|_{p_i}\} >^{mul} \Gamma_i\sigma_i$ where the $\Gamma_i\sigma_i \Rightarrow l_i\sigma_i \to r_i\sigma_i \,[\![c_i\sigma_i]\!]$, $i \in [1..n]$, are all the instances of rules $\Gamma_i \Rightarrow l_i \to r_i \,[\![c_i]\!] \in \mathcal{R}_\mathcal{D}$ such that $l_i\sigma_i = C|_{p_i}$

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

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

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

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

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

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

Figure 4: System $\mathcal{S}$: simplification rules

$$\textsf{Simplification} \quad \frac{(\mathcal{E} \cup \{C\,[\![c]\!]\}, \mathcal{H})}{(\mathcal{E} \cup \mathcal{E}', \mathcal{H})}$$
if $\mathcal{I}ndvar(C\,[\![c]\!]) = \emptyset$ and $C\,[\![c]\!] \vdash_{\mathcal{S}} \mathcal{E}'$

$$\textsf{Inductive Narrowing} \quad \frac{(\mathcal{E} \cup \{C\,[\![c]\!]\}, \mathcal{H})}{(\mathcal{E} \cup \mathcal{E}_1 \cup \ldots \cup \mathcal{E}_n, \mathcal{H} \cup \{C\,[\![c]\!]\})}$$
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. $C\,[\![c]\!] \vdash_{\mathcal{G}}^{+} C_i\,[\![c_i]\!]$

$$\textsf{Subsumption} \quad \frac{(\mathcal{E} \cup \{C\,[\![c]\!]\}, \mathcal{H})}{(\mathcal{E}, \mathcal{H})}$$
if $C\,[\![c]\!]$ is subsumed by another clause of $\mathcal{R} \cup \mathcal{E} \cup \mathcal{H}$

$$\textsf{Disproof} \quad \frac{(\mathcal{E} \cup \{C\,[\![c]\!]\}, \mathcal{H})}{(\textsf{Disproof}, \mathcal{H})}$$
if $C\,[\![c]\!]$ is a constructor clause and no other rule applies to $C\,[\![c]\!]$

$$\textsf{Failure} \quad \frac{(\mathcal{E} \cup \{C\,[\![c]\!]\}, \mathcal{H})}{(\textsf{Failure}, \mathcal{H})}$$
if $C\,[\![c]\!]$ is not a constructor clause and no other rule applies to the clause $C\,[\![c]\!]$

Figure 5: System $\mathcal{I}$: inference rules for joinable-induction

5.3 Main inference system

The main inference system $\mathcal{I}$ is displayed in Figure 5. 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 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 1. *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 4). 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 $\mathcal{E}$).

5.4 Soundness and Completeness

Our inference system is sound, and refutationally complete.

Definition 2. *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. The proof of the following theorems can be found in the long version [6] of this extended abstract.

Theorem 1 (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* $\mathcal{R} \models_{jind} \mathcal{E}_0$*.*

The following theorem states that the derivation of Disproof by our inference system is a correct refutation of the conjecture.

Theorem 2 (Soundness of disproof). *If a derivation starting from* $(\mathcal{E}_0, \emptyset)$ *returns the pair* $(\mathsf{Disproof}, \mathcal{H})$*, then* $\mathcal{R} \not\models_{jind} \mathcal{E}_0$*.*

The derivation of Failure means that we cannot conclude, however, this never happens providing the property of strongly completeness for $\mathcal{R}$. 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 $\mathcal{R}$.

Definition 3. *Let* $f \in \mathcal{D}$ *and let:* $\left\{ \Gamma_1 \Rightarrow f(t_1^1, \ldots, t_k^1) \rightarrow r_1 [\![c_1]\!], \ldots, \Gamma_n \Rightarrow f(t_1^n, \ldots, t_k^n) \rightarrow r_n [\![c_n]\!] \right\}$ *be a maximal subset of rules of* $\mathcal{R}_{\mathcal{D}}$ *whose left-hand sides are identical up to variable renaming* $\mu_1, \ldots, \mu_n$ i.e. $f(t_1^1, \ldots, t_k^1)\mu_1 = \ldots = f(t_1^n, \ldots, t_k^n)\mu_n$*. We say that* f *is* strongly complete *wrt* $\mathcal{R}$ *(see [4]) if* f *is sufficiently complete wrt* $\mathcal{R}$ *and* $\mathcal{R} \models_{jind} \Gamma_1\mu_1 [\![c_1\mu_1]\!] \vee \ldots \vee \Gamma_n\mu_n [\![c_n\mu_n]\!]$ *for every subset of* $\mathcal{R}$ *as above. The system* $\mathcal{R}$ *is said strongly complete if every* $f \in \mathcal{D}$ *is strongly complete wrt* $\mathcal{R}$*.*

Theorem 3 (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* $(\mathsf{Disproof}, \mathcal{H})$*.*

Conclusion

We have developed a procedure for proving joinable inductive theorems of conditional and constrained constructor based specifications which may be non confluent. This procedure is shown correct and refutationally complete, and has been applied to the verification of security properties of cryptographic protocols, both for the research of attacks or protocol validation.

A closely related first order model, also based on trace, was proposed in [10]. This exact model was defined in order to prove a theoretical result on the minimal number of user names required in order to prove security properties. To our knowledge, this model has not been applied in practice for protocol verification.

We are planing several development of this method for protocol verification. First, a natural case study for induction are group protocols, see *e.g.* [8], with some induction on the number of participants.

Several procedures permit automatic validation of protocols described by first order specifications, e.g. [2], but they generally rely on over-approximating models, and are not suitable for the research of attacks, as they generate false positives. Trace based models, like the one presented here, are not approximated and hence appropriate for the search of attack, but automatic protocol validation in such models is considered as a difficult problem. The main difficulty is to generate invariants about the set of data that the attacker can not deduce. Systems like Securify [12] or Hermes [3] are based on some fixed generic invariants. We would like to study the problem of the automatic generation of appropriate ad-hoc invariants, based on the theoretical framework proposed in this paper.

Acknowledgments. The authors wish to thank Hubert Comon-Lundh for the fruitful discussions they have had about this method and the reviewers for their useful remarks.

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. B. Blanchet. Automatic Proof of Strong Secrecy for Security Protocols In IEEE Symp. on Security and Privacy, pages 86-100, 2004.
3. L. Bozga, Y.'Lakhnech, M. Périn. HERMES: An Automatic Tool for Verification of Secrecy in Security Protocols. In Proc. of the 15th Computer-Aided Verification conf. (CAV'03), vol. 2725 of Spinger LNCS, 2003.
4. A. Bouhoula. Automated theorem proving by test set induction. *Journal of Symbolic Computation*, 23(1):47–77, 1997.
5. A. Bouhoula and F. Jacquemard. Automated induction for complex data structures. Research Report LSV-05-11, Laboratoire Spécification et Vérification, 2005.
6. A. Bouhoula and F. Jacquemard. Tree Automata, Implicit Induction and Explicit Destructors for Security Protocol Verification. Research Report LSV-07-10, 2007.
7. A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 1995.

8. 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.
9. 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.
10. H. Comon and V. Cortier. Security properties: two agents are sufficient. Science of Computer Programming 50(1-3), pages 51-71, Elsevier, 2004.
11. H. Comon-Lundh. *Handbook of Automated Reasoning*, chapter Inductionless Induction. Elsevier, 2001.
12. V. Cortier, J. Millen, and H. Rueß. Proving secrecy is easy enough. In Proc. 14th IEEE Computer Security Foundations Workshop (CSFW'01), pages 97-110. IEEE Comp. Soc. Press, 2001.
13. D. E. Denning and G. M. Sacco. Timestamps in Key Distribution Protocols. In *Communications of the ACM*, 1981.
14. 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.
15. L. C. Paulson. The inductive approach to verifying cryptographic protocol. *Journal of Computer Security*, 6:85–128, 1998.

Appendix

A Normal form constrained tree grammar for Section 3

The constrained tree grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ for the generation of constructor normal forms 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):

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

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.

B Shamir-Rivest-Adleman Three Pass Protocol

We consider the same signature as in Section 3 and also the same constructor CTRS $\mathcal{R}_\mathcal{C}$ extended with the following commutativity-like rule for the encryption operator:

$$\mathsf{aenc}(\mathsf{aenc}(x, k_1), k_2) = \mathsf{aenc}(\mathsf{aenc}(x, k_2), k_1) \, [\![k_1 > k_2]\!] \tag{2}$$

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

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

Initially, the honest user 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.

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

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 (2) of $\mathcal{R}_\mathcal{C}$ is equivalent to $\mathsf{aenc}\big(\mathsf{aenc}(S, \mathsf{key}(x_b)), \mathsf{key}(x_a)\big)$.

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

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 and for the attacker ($\mathsf{att\text{-}init}$ and $\mathsf{att\text{-}anlz}$) as in Section 3. 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 3, 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 3) 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 $\mathsf{sent}(2, B, A, \ldots)$

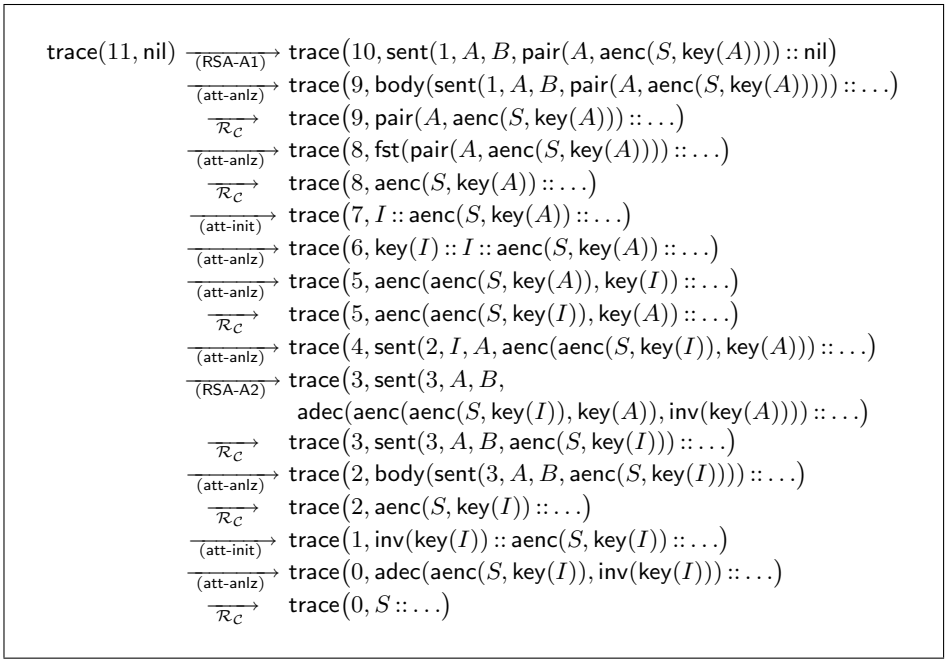

Figure 6: An attack on the secrecy of S for RSA protocol

in between. The conjectures are the same as in Section 3, (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 3 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.