Security Protocol Verification with Implicit Induction and Explicit Destructors [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 a key distribution cryptographic protocol. The method can handle axioms between constructor terms, a feature generally not supported by other induction procedure. We use such axioms in order to specify explicit destructors representing cryptographic operators.

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 [9] proposes an inductive approach to verify cryptographic protocols which has been applied to several case studies during the past years. 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 even for simple protocols. Moreover, 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 [4] derive attacks on faulty protocols specified in first-order logic using a *proof by consistency* technique. Such technique is sometimes also called *inductionless induction* [7] 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.

[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 paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

In this paper we present some ideas for the formal verification security protocols with an extension of an *implicit induction* procedure [2]. The main novelty of this procedure is that it handles axioms between constructors. Such 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]. In contrast to the technique of [4], 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 [2] 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. As shown by the example, it generates readable counter examples, from which an attack can be effectively recovered. It also permits the validation of authentication protocols.

2 Implicit Inductive Theorem Proving procedure

We briefly present in this section an adaptation of an inductive theorem proving procedure that we have proposed in [2] and that will be applied to the protocol specifications. We assume given a many-sorted signature $\mathcal{F} = \mathcal{C} \uplus \mathcal{D}$, where $\mathcal{C}$ is a set of *constructor symbols*, and $\mathcal{D}$ is a set of *defined symbols*, and a finite set $\mathcal{L}$ of predicate symbols with a recursive Boolean interpretation in the domain of ground constructor terms $\mathcal{T}(\mathcal{C})$. Typically, $\mathcal{L}$ contains the syntactic equality and disequality $\approx$ and $\not\approx$ and membership $x{:}L$ to a fixed regular tree language $L \subseteq \mathcal{T}(\mathcal{C})$. A *constraint* is a Boolean combinations of atoms of the form $P(t_1, \ldots, t_n)$ where $P \in \mathcal{L}$ and $t_1, \ldots, t_n$ are in $\mathcal{T}(\mathcal{C}, \mathcal{X})$, the set of constructor terms with variables. 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 and the set of solutions of c is denoted by $sol(c)$. We use the notation $t\,[\![c]\!]$ for *constrained terms*.

A *conditional constrained rewrite rule* is a constrained equational Horn clause of the form $u_1 = v_1, \ldots, u_n = v_n \Rightarrow \ell \to r\,[\![c]\!]$ ($\ell \to r$ is an oriented equation). When $n = 0$, the clause 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). A term $t\,[\![d]\!]$ rewrites to $s\,[\![d]\!]$ by the above rule of $\mathcal{R}$, denoted by $t\,[\![d]\!] \xrightarrow[\mathcal{R}]{} s\,[\![d]\!]$, if $t|_p = l\sigma$ 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 $i \leq n$ [4].

Let $\mathcal{R}$ be a terminating constraint rewrite system. A constrained equation

[4] $u_i\sigma \downarrow_{\mathcal{R}} v_i\sigma$ stands for $\exists w,\ u \xrightarrow[\mathcal{R}]{*} w \xleftarrow[\mathcal{R}]{*} v$, $\xrightarrow[\mathcal{R}]{*}$ is the reflexive transitive closure of $\xrightarrow[\mathcal{R}]{}$.

$a = b \,[\![c]\!]$ is a called an *inductive theorem* of $\mathcal{R}$ if for all substitution $\sigma \in sol(c)$ grounding for $a = b$, $\mathcal{R} \models a\sigma = b\sigma$; it is a *strong inductive theorem* of $\mathcal{R}$ (denoted by $\mathcal{R} \models_{sind} a = b \,[\![c]\!]$) if for all σ and all $\mathcal{R}$-normal forms n_a, n_b respectively of $a\sigma$, $b\sigma$, we have $n_a \equiv n_b$. These notions are extended to clauses as expected. The two notions coincide when $\mathcal{R}$ is ground confluent. The definition of strong inductive theorems is motivated by the application presented in Section 3.

Our inductive theorem proving method is a goal-directed proof technique which handles constructor based conditional and constrained specifications containing in particular constrained rewrite rules between constructors. This feature, which is generally not supported in former inductive theorem proving approaches, permits us to deal with explicit definition of cryptographic operators in Section 3. This procedure belongs to the family of *implicit induction* (in the lines of [3]) and combines the power of two classical methods for automatic induction: *explicit induction* and *proof by consistency* [7]. It is based on constrained tree grammar, a tree generator whose production rules are constrained. The grammar is computed automatically from the given specification and is used both as an explicit induction scheme, to trigger the induction steps, and for deciding a deletion criteria, as summarized in the following description of the main steps of our procedure (see [2] for details). We start with a conjecture (goal) C and a CCTRS $\mathcal{R}$, with a subset $\mathcal{R}_\mathcal{C}$ of constrained rewrite rules between constructor terms and such that all the rules of $\mathcal{R}_\mathcal{D} := \mathcal{R} \setminus \mathcal{R}_\mathcal{C}$ are conditional constrained rules of the form $\Gamma \Rightarrow f(\ell_1, \ldots, \ell_k) \to r \,[\![c]\!]$ with $f \in \mathcal{D}$, $\ell_1, \ldots, \ell_k, r \in \mathcal{T}(\mathcal{C}, \mathcal{X})$.

(i) compute a constrained tree grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ which generates the set of ground constructor terms in normal form for $\mathcal{R}_\mathcal{C}$,

(ii) for each goal (or subgoal) C, generate instances of C by using the production rules of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ (instead of a test set). We obtain $C_1, \ldots, C_n$.

(iii) **for each** C_i, do:
  (a) **if** C_i is a tautology or C_i is a constructor clause and can be detected as strongly inductively valid **then** delete it
  (b) **else** reduce C_i into $D_{i,1}, \ldots, D_{i,k}$, using the axioms [5] and ind. hyp., if possible,
  (c) **otherwise** (if reduction is not possible) **disproof**

(iv) **if** (iii) did not fail with **disproof then** C becomes an *induction hypothesis* and every $D_{i,j}$ becomes a new subgoal, **go to** (ii)

If every subgoal is deleted, then C is a strong inductive theorem of $\mathcal{R}$. Termination of the process may be achieved if necessary by incorporating appropriate lemmas.

[5] If $\mathcal{R}$ is not ground confluent, we consider all the (one step) reductions with $\mathcal{R}$.

Constrained tree grammars permit an exact representation of the set of ground constructor terms irreducible by a given CTRS like $\mathcal{R}_\mathcal{C}$. Under some assumptions, like sufficient completeness of $\mathcal{R}$ and termination of $\mathcal{R}_\mathcal{C}$, this language is a set of representatives of the minimal Herbrand model of $\mathcal{R}$. For this reason, such formalisms have been studied in many works related to inductive theorem proving, see *e.g.* [7]. Due to space limitation we shall not give the formal definitions of these grammars, of the non-terminal replacement performed at step (ii), and of the automatic construction of $\mathcal{G}_{\rm NF}(\mathcal{R}_\mathcal{C})$ at step (i) (the reader may refer to [2]). An example of of $\mathcal{G}_{\rm NF}(\mathcal{R}_\mathcal{C})$ is given in the next section. The simplifications at step (b) are simple application of the rewrite rules or other rules for case analysis presented [2].

The test of strong validity at step (a) is based on a reduction to a constrained tree grammar non-emptiness problem (does there exist at least one term generated by a given grammar), using $\mathcal{G}_{\rm NF}(\mathcal{R}_\mathcal{C})$. It requires that the subgoal S is *ground irreducible*, a notion often central in proof by consistency procedures [7] and which can be decided by similar reductions, as in [6]. This method can be effective for some classes of grammars presented *e.g.* in [5] (see [2] for details).

Like in [2], this procedure is sound and refutationally complete when the given CCTRS is sufficiently complete and the constructor subsystem $\mathcal{R}_\mathcal{C}$ is terminating. Without the above hypotheses, it still remains sound and refutationally complete for conjectures whose variables are constrained to belong to constructor $\mathcal{R}_\mathcal{C}$-normal form languages defined by non terminals of $\mathcal{G}_{\rm NF}(\mathcal{R}_\mathcal{C})$ (we shall proceed this way in the next section).

3 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 protocols, where a security property is expressed as a strong inductive conjecture. Following [9], we perform induction on protocol execution traces which are recursively defined by a 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 strong 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 [8] for a symmetric key exchange in an asymmetric cryptosystem. Assume some sorts Nat, Bool, Name, 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 constructor terms build with symbols $\mathsf{pair} : \mathrm{Msg} \times \mathrm{Msg} \to \mathrm{Msg}$ and projections $\mathsf{fst}, \mathsf{snd} : \mathrm{Msg} \to \mathrm{Msg}$ following the rules $\mathsf{fst}(\mathsf{pair}(x_1, x_2)) \to x_1$ and $\mathsf{snd}(\mathsf{pair}(x_1, x_2)) \to x_2$, decryption in symmetric key cryptography: $\mathsf{dec}\big(\mathsf{enc}(x, y), y\big) \to x$ (the variables x and y correspond respectively to the encrypted plaintext and the encryption key), decryp-

tion in public (asymmetric) key cryptography: $\mathsf{adec}\big(\mathsf{aenc}(x,y),\mathsf{inv}(y)\big) \to x$ and $\mathsf{adec}\big(\mathsf{aenc}(x,\mathsf{inv}(y)),y\big) \to x$ where inv is an idempotent operator, following the rule $\mathsf{inv}(\mathsf{inv}(y)) \to y$, which associates to a public encryption key its corresponding private key (for decryption), and conversely. The symbols of cryptographic operators have the following profiles: $\mathsf{pub} : \text{Name} \to \text{Key}$, $\mathsf{inv} : \text{Key} \to \text{Key}$, $\mathsf{enc}, \mathsf{aenc}, \mathsf{dec}, \mathsf{adec} : \text{Msg} \times \text{Msg} \to \text{Msg}$, inv is called *secret* and all the others are called *public*. We consider also a secret constructor $\mathsf{sent} : \text{Name} \times \text{Name} \times \text{Msg} \to \text{Msg}$ to add headers to messages, and a public constructor symbol $\mathsf{body} : \text{Msg} \to \text{Msg}$ for removing it with the rule $\mathsf{body}(\mathsf{sent}(x_a, x_b, x)) \to x$. We assume moreover some additional secret constructors for Boolean: $\mathsf{true}, \mathsf{false} : \text{Bool}$, natural numbers $0 : \text{Nat}$, $s : \text{Nat} \to \text{Nat}$, lists of messages, $\mathsf{nil} : \text{EventList}$, $:: : \text{Msg} \times \text{MsgList} \to \text{MsgList}$ and constant values used in the protocol messages: $K : \text{Key}$, $S : \text{Msg}$. 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 (which we won't define explicitly) 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 $\mathcal{G}_{\text{NF}}(\mathcal{R}_\mathcal{C})$ has sorted non terminals: $\llcorner\mathsf{pair}(x_1,x_2)\lrcorner$, $\llcorner\mathsf{enc}(x,y)\lrcorner$, $\llcorner\mathsf{aenc}(x,y)\lrcorner$, $\llcorner\mathsf{inv}(v)\lrcorner$, $\llcorner\mathsf{aenc}(x,\mathsf{inv}(y))\lrcorner$, $\llcorner\mathsf{sent}(x_a,x_b,x)\lrcorner$, Name, $\llcorner x\lrcorner^{\text{Key}}$, $\llcorner x\lrcorner^{\text{Msg}}$, $\llcorner x\lrcorner^{\text{List}}$, $\llcorner x\lrcorner^{\text{Bool}}$, $\llcorner x\lrcorner^{\text{Nat}}$ and $\llcorner x\lrcorner^{\text{red}}$. 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 $\mathcal{G}_{\text{NF}}(\mathcal{R}_\mathcal{C})$ are the following (M represents below any non-terminal of sort Msg):

$$\llcorner x\lrcorner^{\text{Nat}} := 0 \mid s(\llcorner x\lrcorner^{\text{Nat}}) \qquad \llcorner x\lrcorner^{\text{List}} := \mathsf{nil} \mid M :: \llcorner x\lrcorner^{List} \qquad \llcorner x\lrcorner^{\text{Key}} := K \mid \mathsf{pub}(\text{Name})$$

$$\llcorner\mathsf{enc}(x,y)\lrcorner := \mathsf{enc}(M_1,M_2) \qquad \llcorner\mathsf{inv}(y)\lrcorner := \mathsf{inv}(\llcorner x\lrcorner^{\text{Key}})$$

$$\llcorner\mathsf{aenc}(x,y)\lrcorner := \mathsf{aenc}(M_1,M_2) \qquad \llcorner\mathsf{aenc}(x,\mathsf{inv}(y))\lrcorner := \mathsf{aenc}(M,\llcorner\mathsf{inv}(y)\lrcorner)$$

$$\llcorner\mathsf{pair}(x_1,x_2)\lrcorner := \mathsf{pair}(M_1,M_2) \qquad \llcorner x\lrcorner^{\text{red}} := \mathsf{fst}\big(\llcorner\mathsf{pair}(x_1,x_2)\lrcorner\big) \mid \mathsf{snd}\big(\llcorner\mathsf{pair}(x_1,x_2)\lrcorner\big)$$

$$\llcorner\mathsf{sent}(x_a,x_b,x)\lrcorner := \mathsf{sent}(\llcorner x\lrcorner^{\text{Name}},\llcorner x\lrcorner^{\text{Name}},M) \qquad \llcorner x\lrcorner^{\text{red}} := \mathsf{body}\big(\llcorner\mathsf{sent}(x_a,x_b,x)\lrcorner\big)$$

$$\llcorner x\lrcorner^{\text{Msg}} := \mathsf{dec}\big(\llcorner\mathsf{enc}(x,y)\lrcorner,M\big)\,[\![y \not\approx M]\!] \qquad \llcorner x\lrcorner^{\text{red}} := \mathsf{dec}\big(\llcorner\mathsf{enc}(x,y)\lrcorner,M\big)\,[\![y \approx M]\!]$$

$$\llcorner x\lrcorner^{\text{Msg}} := \mathsf{adec}\big(\llcorner\mathsf{aenc}(x,y_1)\lrcorner,\llcorner\mathsf{inv}(y_2)\lrcorner\big)\,[\![y_1 \not\approx y_2]\!] \qquad \llcorner x\lrcorner^{\text{red}} := \ldots\,[\![y_1 \approx y_2]\!]$$

$$\llcorner x\lrcorner^{\text{Msg}} := \mathsf{adec}\big(\llcorner\mathsf{aenc}(x,\mathsf{inv}(y))\lrcorner,M\big)\,[\![y \not\approx M]\!] \qquad \llcorner x\lrcorner^{\text{red}} := \ldots\,[\![y \approx M]\!]$$

The non terminal $\llcorner x\lrcorner^{\text{red}}$ 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 [9], we consider traces of messages modelled as lists, characterized by the symbol $\mathsf{trace} : \text{MsgList} \to \text{Bool}$, which is defined recursively by $\mathsf{trace}(\mathsf{nil}) = \mathsf{true}$ and by extension with messages sent by the agents participating to the protocol (honest or not). In the case of the Den-

ning & Sacco protocol, the conditional rule (1) describes the honest agent x_a sending to agent x_b a freshly chosen symmetric key K for further secure communications. This key 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.

$$\mathsf{trace}(y) = \mathsf{true} \Rightarrow \mathsf{trace}\big(\mathsf{sent}(x_a, x_b, \mathsf{pair}(x_a, \mathsf{aenc}(\mathsf{aenc}(K, \mathsf{inv}(\mathsf{pub}(x_a))), \mathsf{pub}(x_b)))) :: y\big) \to \mathsf{true}\,[\![x_a : \mathrm{Name}, x_b : \mathrm{Name}, x_a \not\approx x_b]\!] \quad (1)$$

In the second conditional rule (2), the honest agent x_b, while reading a message x, expects that x has the above form. 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 that he wants to communicate to the agent x_a.

$$\mathsf{trace}(y) = \mathsf{true}, \mathsf{sent}(x'_a, x_b, x_m) \in y \Rightarrow \mathsf{trace}(\mathsf{sent}(x_b, \mathsf{fst}(x_m), \mathsf{enc}(S, \mathsf{adec}(\mathsf{adec}(\mathsf{snd}(x_m), \mathsf{inv}(\mathsf{pub}(x_b))), \mathsf{pub}(\mathsf{fst}(x_m))))) :: y) \to \mathsf{true} \quad (2)$$

We assume that the messages are sent and read through an insecure public network controlled by an attacker. The attacker is able to read any message sent to the network, to extract information from collected messages by applying public constructor symbols and the rules of $\mathcal{R}_\mathcal{C}$, and to resend composed messages to the network. The information extraction is modelled by the defined function $\mathsf{analyze} : \mathrm{Nat} \times \mathrm{MsgList} \to \mathrm{MsgList}$; given a list ℓ of messages exchanged, we define $\mathsf{analyze}(\ell)$ is the smallest set containing ℓ, and closed under application of public constructor functions:

$$\mathsf{analyze}(0, y) \to y \qquad \mathsf{analyze}(s(n), y) \to \mathsf{analyze}(n, x :: y)\,[\![x : \mathrm{Name}]\!]$$
$$x \in y \Rightarrow \mathsf{analyze}(s(n), y) \to \mathsf{analyze}(n, f_1(x) :: y)$$
$$x_1 \in y, x_2 \in y \Rightarrow \mathsf{analyze}(s(n), y) \to \mathsf{analyze}(n, f_2(x_1, x_2) :: y)$$

The two last rules are for every public constructor f_1 of arity 1 (pub, fst, snd, body) and f_2 of arity 2 (pair, enc, dec, aenc, adec). The defined function $\in : \mathrm{Msg} \times \mathrm{MsgList} \to \mathrm{Bool}$ follows the rules: $x \in \mathsf{nil} \to \mathsf{false}$, $x_1 \in x_2 :: y \to \mathsf{true}\,[\![x_1 \approx x_2]\!]$ and $x_1 \in x_2 :: y \to x_1 \in y\,[\![x_1 \not\approx x_2]\!]$. The ability of the attacker to send fake message is modelled by this last conditional rule (3):

$$\mathsf{trace}(y) = \mathsf{true}, x \in \mathsf{analyze}(n, y) = \mathsf{true} \Rightarrow \mathsf{trace}(\mathsf{sent}(I, x_a, x) :: y) \to \mathsf{true}\,[\![x_a : \mathrm{Name}]\!] \quad (3)$$

This conjecture expresses that the constant S remains secret to the attacker:

$$\mathsf{trace}(y) = \mathsf{true} \Rightarrow S \in \mathsf{analyze}(n, y) \rightarrow \mathsf{false}\, [\![y : {}_{\llcorner}y^{\mathrm{List}}_{\lrcorner}, n : {}_{\llcorner}x^{\mathrm{Nat}}_{\lrcorner}]\!]$$

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 ${}_{\llcorner}y^{\mathrm{List}}_{\lrcorner}$ and ${}_{\llcorner}x^{\mathrm{Nat}}_{\lrcorner}$. The application of our procedure shows that this conjecture is not a strong inductive theorem of $\mathcal{R}$, by induction on traces.
Among the instances of the goal C generated[6] by application of the production rules of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$, we have $C\sigma$ where the substitution σ associates the variable y with $\ell = \mathsf{sent}(B, A, \mathsf{enc}(S, \mathsf{adec}(A, \mathsf{pub}(A)))) :: \mathsf{sent}(I, B, \mathsf{pair}(A, \mathsf{aenc}(A, \mathsf{pub}(B)))) :: \mathsf{nil}$ and n with $s(s(s(s(0))))$, denoted 4 below (A, B and I are arbitrary distinct constructor terms of sort Name).

We can show that $C\sigma$ is a counterexample (which indicates an attack). In fact, the $\mathcal{R}$-normal form of $\mathsf{trace}(\ell)$ is true, and $\mathsf{analyze}(5, \ell)$ can be normalized as follows (let $\ell' = \mathsf{enc}(S, \mathsf{adec}(A, \mathsf{pub}(A))) :: \ell$): $\mathsf{analyze}(5, \ell) \rightarrow \mathsf{analyze}(4, \ell') \rightarrow \mathsf{analyze}(3, A :: \ell') \rightarrow \mathsf{analyze}(2, \mathsf{pub}(A) :: A :: \ell') \rightarrow \mathsf{analyze}(1, \mathsf{adec}(A, \mathsf{pub}(A)) :: \mathsf{pub}(A) :: A :: \ell') \rightarrow \mathsf{analyze}(0, \mathsf{dec}(\mathsf{enc}(S, \mathsf{adec}(A, \mathsf{pub}(A))), \mathsf{adec}(A, \mathsf{pub}(A))) :: \ldots) \rightarrow \mathsf{analyze}(0, S :: \ldots) \rightarrow S :: \ldots$. Hence, the conjecture is reduced to $\mathsf{true} = \mathsf{true} \Rightarrow \mathsf{true} \rightarrow \mathsf{false}$, which leads to a case of **disproof**.

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

$$\mathsf{trace}(y) = \mathsf{true} \Rightarrow \mathsf{trace}(\mathsf{sent}(x_a, x_b, \mathsf{pair}(x_a, \mathsf{aenc}(\mathsf{aenc}(\mathsf{pair}(\mathsf{pair}(x_a, x_b), K), \mathsf{inv}(\mathsf{pub}(x_a))), \mathsf{pub}(x_b))) :: y) \rightarrow \mathsf{true}\, [\![x_a : \mathrm{Name}, x_b : \mathrm{Name}, x_a \neq x_b]\!] \quad (1')$$

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

$$\mathsf{trace}(y) = \mathsf{true}, \mathsf{sent}(x'_a, x_b, x_m) \in y, \mathsf{snd}(\mathsf{fst}(k)) = x_b, \mathsf{fst}(\mathsf{fst}(k)) = \mathsf{fst}(x_m) \Rightarrow \mathsf{trace}(\mathsf{sent}(x_b, \mathsf{fst}(x_m), \mathsf{enc}(S, k))) :: y) \rightarrow \mathsf{true} \quad (2')$$

Let us complete the defined function trace with the following axioms:

$$\mathsf{trace}(y) = \mathsf{false} \Rightarrow \mathsf{trace}(x :: y) \rightarrow \mathsf{false},$$
$$\mathsf{trace}(y) = \mathsf{true}, x \notin \mathsf{analyze}(n, y) = \mathsf{true} \Rightarrow \mathsf{trace}(\mathsf{sent}(I, x_a, x) :: y) \rightarrow \mathsf{false}\, [\![x_a : \mathrm{Name}]\!] \ldots$$

In order to prove that the above conjecture is now true for the modified version of the protocol, we generate the following subgoals ($f_1 \in \{\mathsf{pub}, \mathsf{fst}, \mathsf{snd}, \mathsf{body}\}$

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

and $f_2 \in \{\mathsf{pair},\ \mathsf{enc},\ \mathsf{dec},\ \mathsf{aenc},\ \mathsf{adec}\}$):

$$\mathsf{trace}(y) = \mathsf{true} \Rightarrow S \in y \rightarrow \mathsf{false}\,[\![y : \llcorner y \lrcorner^{\mathrm{List}}]\!]$$

$$\mathsf{trace}(y) = \mathsf{true} \Rightarrow S \in \mathsf{analyze}(n, x :: y) \rightarrow \mathsf{false}\,[\![y : \llcorner y \lrcorner^{\mathrm{List}}, n : \llcorner x \lrcorner^{\mathrm{Nat}}, x : \mathrm{Name}]\!]$$

$$\mathsf{trace}(y) = \mathsf{true}, x \in y \Rightarrow S \in \mathsf{analyze}(n, f_1(x) :: y) \rightarrow \mathsf{false}\,[\![y : \llcorner y \lrcorner^{\mathrm{List}}, n : \llcorner x \lrcorner^{\mathrm{Nat}}]\!]$$

$$\mathsf{trace}(y) = \mathsf{true}, x_1 \in y, x_2 \in y \Rightarrow S \in \mathsf{analyze}(n, f_2(x_1, x_2) :: y) \rightarrow \mathsf{false}\,[\![y : \llcorner y \lrcorner^{\mathrm{List}}, n : \llcorner x \lrcorner^{\mathrm{Nat}}]\!]$$

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. The difficulty here is that we need to verify all the execution traces in order to certify a security protocol when $\mathcal{R}$ is not ground confluent (definition of *strong* inductive theorems), whereas it is sufficient to find one erroneous trace in order to show that the protocol is flawed. 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] Adel Bouhoula and Florent Jacquemard. Automated induction for complex data structures. Research Report LSV-05-11, Laboratoire Spécification et Vérification, ENS Cachan, France, July 2005.

[3] Adel Bouhoula and Michaël Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 1995.

[4] Alan Bundy and Graham Steel. Attacking group protocols by refuting incorrect inductive conjectures. *Journal of Automated Reasoning*, Special Issue on Automated Reasoning for Security Protocol Analysis:1–28, december 2005.

[5] Hubert Comon, Max Dauchet Rémy Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. http://www.grappa.univ-lille3.fr/tata, 2002.

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

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

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

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