Certifying cryptographic protocols by abstract model-cheking and proof concretization

R. Janvier, Y. Lakhnech, and M. Périn

VERIMAG, 2 av. de Vignate, 38610 Gières, France.

Abstract. In this paper, we report on our effort in enhancing our model-checker for cryptographic protocols with the ability to automatically generate a deductive proof that the protocol meets its specification. More specifically, we discuss a technique that allows to transform an abstract proof extracted from the model-checker to a proof that can be checked independently of the abstracting and model-checking process.

Model-checking, introduced in [7, 20], has enabled automatic verification of industrial size applications. It has proven successful in verifying hardware designs as well as software applications. A key element in the application of model-checking is the use of abstraction techniques that reduce the verification of a system with a large state space to the conservative verification of an abstract system with a tractable state space. The abstracting process involves designing an abstraction relation, computing an abstract system and transforming the concrete property into an abstract one.

When abstract executions that violate the abstract property exist, the model-checker reports at least one. Thus, a counter-example can be considered as a "certificate/proof" that the system does not satisfy its property. In the last few years, many techniques have been developed for analyzing abstract counter-examples with the goal of either reporting a concrete execution that violates the property or refining the abstracting process (e.g. [6, 11, 9]). When the model-checker does not find a counter-example one may conclude that the concrete system satisfies its specification. However, this conclusion is only justified in case:

1. The abstracting process is sound meaning that the abstract system and the abstract property over-approximate the concrete system.
2. The model-checker does not contain itself a bug which causes it to report absence of counter-examples while, in fact, the checked system is faulty.

Thus, because of the absence of a certificate, it might be natural to consider a positive answer of a model-checker with some diffidence.

An alternative approach to model-checking is *deductive verification*. It consists in constructing an *explicit* proof that the system satisfies its specification. Such a proof can be validated with the help of a proof-checker (that is a crucial but small part of a theorem prover). Moreover, it does not matter *how* such a proof has been constructed or found. All that matters is that it is a proof obtained using a sound deduction system.

Clearly, one might argue that also a model-checker constructs a proof. However, such a proof is hidden inside the model-checker. Moreover, and may be even more importantly, in case of abstraction-based model-checking the model-checker constructs a proof for the **abstract system**.

In this paper, we report on our effort in enhancing our Model-Checker for Cryptographic Protocols, HERMES [3, 4, 25, 5], with the ability to automatically generate a deductive proof that the protocol meets its specification. In the context of certification, we seek for the smallest Trusted Computing Base as possible. Our work aims at removing the model-checker and the abstraction it uses from the TCB. It is therefore mandatory to study algorithms that allow to have *model-checker independent proofs*.

Our approach in developing such algorithms consists in two phases:

1. The first phase consists in instrumenting our model-checker in order to obtain a deductive proof for the abstract system. This is the *proof extraction* process.
2. The second phase consists in transforming the abstract proof into a proof for the concrete system. This is, the *proof concretization* process. A distinguishing feature of our approach is that we target a general purpose theorem prover in contrast to a tailored proof system. More specifically, we target the COQ theorem prover which is based on the calculus of inductive constructions (CIC) [23, 2].

We present our approach in the context of cryptographic protocols for the verification of these systems is particularly challenging because they are concurrent infinite-state systems that manipulate terms as data. However our approach is not specific for this type of systems.

In this extended abstract, we focus on proof concretization as it is less studied than proof extraction. Indeed, several proof extraction algorithms have been developed, e.g., in [18, 21, 17, 15, 22] both for linear-time and branching-time properties.

Our approach for the concretization of the abstract proof relies on the idea that a proof made in the abstract theory can be seen as a proof over the equivalence classes defined by the abstraction. Since the predicates of the concrete theory do not evaluate to the same boolean value for all

equivalent concrete models, we associate to each concrete predicate a new predicate which can be seen as an extension or restriction to the equivalence classes. Those predicates are designed to be uniform on equivalence classes, that is, they evaluate to the same boolean value on all the elements of a class. Then, the abstract predicates occurring in the abstract proof term are replaced by the uniform predicates: An abstract predicate is replaced by an extension (resp. restriction) if it is an over- (resp. under-) approximation of the concrete predicate. The replacement is done in such a way that ensures that the structure of the formulas, and the proof tree, remain unchanged. We then obtain proof obligations to complete the concretized proof tree. It turns out that in many cases these proof obligations can be proved automatically, as it is the case for HERMES.

Related Work There is not, to the authors knowledge, an earlier systematic study of the proof concretization problem except the work by K. Namjoshi [16]. Namjoshi's work considers a reacher set of properties, namely μ-calculus definable properties. On the other hand, it targets a specific deductive proof system developed by the same author to represent proofs of correctness of μ-calculus properties.

In [11], an algorithm for concretizing a BDD that represents the set of abstract reachable states is presented. The so-obtained concrete invariant is then used in a deductive proof rule for proving invariance properties. An algorithm is presented in [1, 19] that allows to heuristically generalize an invariant of a small instance of a parameterized system to a candidate invariant of all instances. The so-obtained candidate invariant is subsequently checked for validity. As the small instance is, in general, not a safe abstraction of the entire system there is no guarantee that the generalized invariant is indeed an invariant of the entire system.

The BLAST toolkit allows the verification of drivers encoded in C using the abstraction-based model-checking methodology. It also allows to automatically generate an easily checkable correctness certificate of the considered driver [10]. The concretization procedure that we present in the paper as well as the certification process of BLAST consider invariance properties. Moreover, the latter is tuned for predicate-abstraction.

Outline of the paper Section 1 briefly describes a verification technique and a tool for verifying secrecy properties of cryptographic protocols. We explain in Section 2 how the verification tool has been instrumented to help building a proof of the secrecy of an *abstraction of the protocol*. Section 3 and 4 are dedicated to the concretization process that ends with a

foundational proof that the *actual protocol* satisfies the secrecy property. Conclusions and future work are discussed in Section 5.

1 Verifying secrecy properties of cryptographic protocols

We give a sketchy idea of the technique and the tool, called HERMES, that we developed to verify secrecy properties of cryptographic protocols in Dolev and Yao's model of intruder. A formal and complete presentation of this method can be found in [3]. Cryptographic protocols can be modeled as a set of transitions of the form $t \rightarrow t'$ where t, t' are *terms* constructed by applying pairing and the operator $\{t\}_k$ (that denotes the encryption of a term t by a key k), to some free variables and to the *parameters of the protocol sessions*, which are the identities of agents, the fresh nonces, and the fresh keys of these sessions. A secrecy goal states that several designated terms representing the *secrets* should not be deducible by an intruder whose capacities are defined by the additional transitions of Figure 1, known as Dolev and Yao's model [8]. These transitions can be seen as a deductive system. It defines the messages that the intruder can deduce and forge from the messages sent on the network during the protocols execution: this model identifies the log of the network with the knowledge of the intruder.

pairing:	$t,\ t' \rightarrow (t,t')$	***decryption***:	$\{t\}_k,\ k^{-1} \rightarrow t$
splitting:	$(t,t') \rightarrow t,\ t'$	***encryption***:	$t,\ k \rightarrow \{t\}_k$

In Dolev and Yao's model the network is represented by a set of messages, say $\mathcal{N}$. This transition system, denoted by DY in the sequel, defines the intruder capacities: a transition adds its right-hand side to $\mathcal{N}$ if its left-hand side matches elements of $\mathcal{N}$.

Fig. 1. The model of the intruder defined by Dolev and Yao

HERMES checks that messages sent by honest agents cannot be used by the intruder to obtain the exchanged secrets. It is based on the computation of a set of *dangerous messages* that allow the intruder to deduce the secrets with the unwitting help of honest agents playing the protocol. This set draws sufficient conditions on the initial knowledge of the intruder to ensure that the secrets exchanged by the protocol will be preserved. Section A of the appendix illustrates the verification problem on Needham-Schroeder-Lowe's authentication protocol [13].

A cryptographic protocol $\mathscr{P}$ defines a transition system R that corresponds to the transition of an unbounded number of session of the protocol

$\mathscr{P}$ and the transition of Dolev and Yao's intruder. Given a protocol $\mathscr{P}$ and a set of secrets S, HERMES produces an abstract verification problem $(R^\sharp, S^\sharp)$ using four abstraction relations: on agents, nonces, messages and transitions. It generates the abstraction $S^\sharp$ of S, and computes a finite transition system $R^\sharp = \mathscr{P}^\sharp \cup \text{DY}$ that is an over-approximation of R. The verification is then based on the computation of the predecessors of the abstract secrets $S^\sharp$ for the transition relation $R^\sharp$. HERMES starts with the set of secrets as dangerous messages ($\mathcal{D}^\sharp := S^\sharp$). It computes $pre_{R^\sharp}(\mathcal{D}^\sharp)$, the predecessors of $\mathcal{D}^\sharp$ for the transition relation $R^\sharp$; adds them to $\mathcal{D}^\sharp$; and repeats these two steps until reaching a set of dangerous messages that is stable for $pre_{R^\sharp}$. It is useful for Section 2 to mention that each step involves solving unification problems.

$$\frac{Q \cap Q_0 = \emptyset \qquad pre_R(Q) \subseteq Q}{pre_R^*(Q) \cap Q_0 = \emptyset}$$

$$\forall Q, Q_0, R, \quad pre_R^*(Q) \cap Q_0 = \emptyset \;\Leftrightarrow\; Q \cap R^*(Q_0) = \emptyset$$

Fig. 2. Principle of reachability analysis for the operator pre [14]

The output of HERMES consists in a set of dangerous messages $\mathcal{D}^\sharp$ that satisfies: $S^\sharp \subseteq \mathcal{D}^\sharp$ and $pre_{R^\sharp}(\mathcal{D}^\sharp) \subseteq \mathcal{D}^\sharp$. Based on the properties of Figure 2 for the operator pre, it turns out that:

$$\mathcal{D}^\sharp \cap \mathcal{N}_{init}^\sharp = \emptyset \Rightarrow \mathcal{D}^\sharp \cap R^{\sharp *}(\mathcal{N}_{init}) = \emptyset$$

Using the fact $S^\sharp \subseteq \mathcal{D}^\sharp$, HERMES returns the *verdict at the abstract level*:

$$\mathcal{D}^\sharp \cap \mathcal{N}_{init}^\sharp = \emptyset \Rightarrow S^\sharp \cap R^{\sharp *}(\mathcal{N}_{init}) = \emptyset.$$

It means that the secrets are not reachable by $R^\sharp$ if the initial knowledge of the intruder $\mathcal{N}_{init}^\sharp$ does not contain any dangerous messages. Section A provides an example of interpretation of HERMES' output.

The validity of HERMES' results depends on the correctness of its implementation and the safetyness of the abstraction. In the context of certification our aim is to produce a *concrete verdict* that neither depends on the verification tool we used, nor on the abstraction we applied:

$$\mathcal{D} \cap \mathcal{N}_{init} = \emptyset \Rightarrow S \cap R^*(\mathcal{N}_{init}) = \emptyset.$$

In order to produce a proof of that verdict we proceed in two steps: we instrument the verification tool to get a proof of the property at the

abstract level, then we build a proof of the verdict on the actual system by applying a concretization function to the abstract proof term.

2 Automatic generation of an abstract proof

We present our technique in the context of cryptographic protocol and we illustrate our presentation with examples treated with HERMES. However, the technique can be generalized to other verification tools of safety properties. Section D defines at the semantic level what it means for a protocol to preserves secrecy, and gives a sketch of the proof. We distinguish two parts in that proof: the lemmas that are proved once for all (*e.g.,* the principle of reachability analysis of Figure 2), and the stability property $pre_{R^\sharp}(\mathcal{D}^\sharp) \subseteq \mathcal{D}^\sharp$ that depends on the cryptographic protocol and the secrets we consider. From now on, we focus on the stability property, and we start by generating a proof that it holds at the abstract level.

The proof of the stability of $\mathcal{D}^\sharp$ is built using proof tactics which are generated during execution of an instrumented version of HERMES. A tactic is an heuristic that guides the proof construction. It is given as a collection of proof schemes guarded by syntactic conditions which control its triggering. So, the order in which they are generated by the instrumented algorithm does not matter. Actually, the proof construction puts no constraints on the control flow of the instrumented algorithm. It only matters to provide the arguments needed by the proof and to define criteria that recognize where these arguments are helpful.

We consider proof terms built from the inference rules of the sequent calculus (see Figure 7). The main difficulty in the automatic generation of the proof arises for properties with an existential quantification: The inference rule $(\exists_i)$ that leads to a conclusion $\exists x\ \varphi$ requires to exhibit a witness that satisfies the property φ. We address this problem by running the instrumented version of HERMES on the already stable set $\mathcal{D}^\sharp$. During this run, all attempts to add a "dangerous" message m fails for the reason that it is already covered by a message m' of $\mathcal{D}^\sharp$, and we store the relation (m, m') between the new message and its witness of membership in $\mathcal{D}^\sharp$. This information is stored in a tactic generated by the membership function. It is then used during the proof process to feed the rule $(\exists_i)$. The main effort to obtain the proof of stability was to design the criterion that triggers this tactic. This experiment gave us confidence that it is possible to develop a general framework that allow the developers of a verification algorithm to associate a proof tactics to the computation step of an algorithm.

3 Concretizing abstract proofs

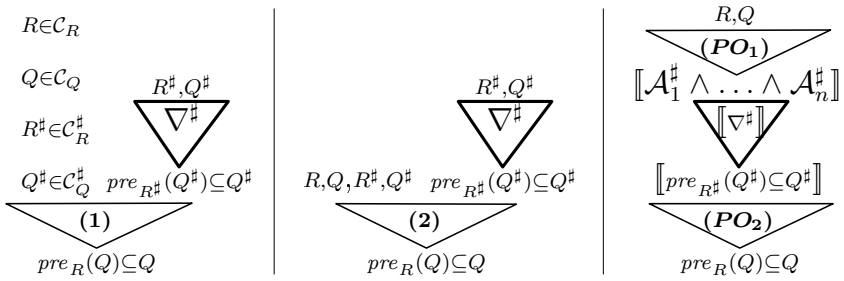
The instrumentation of the verification tool provides us with tactics that drive the COQ proof-engine in order to produce a proof of the stability property $pre_{R^\sharp}(Q^\sharp) \subseteq Q^\sharp$ for an abstract transition system $R^\sharp$ and a property $Q^\sharp$ ([1]). We investigate three techniques that capitalize on that proof to obtain a foundational proof of the stability property $pre_R(Q) \subseteq Q$ at the concrete level. Figure 3 illustrates three possible proofs. They differ on the specialization of the proof toward the specific system and property we consider: *1)* the most general proof consists in proving that the abstraction α is safe for a class $\mathcal{C}_R$ of transition systems and a class $\mathcal{C}_Q$ of properties, that is, $(pre_{R^\sharp}(Q^\sharp) \subseteq Q^\sharp) \Rightarrow (pre_R(Q) \subseteq Q)$ holds for any transition system R in $\mathcal{C}_R$ and any property Q in $\mathcal{C}_Q$. This is the kind of results we find in literature on abstractions to ensures that the verification at the abstract level entails the property at the concrete level. *2)* An easier proof consists in proving the previous result only for a specific system R and a specific property Q. This proof and the previous one require to deal with the definitions of the abstraction relation, and with the transition systems and the properties at both the concrete and abstract levels.

We advocate a third way that we believe is simpler because it only deals with the concrete/actual system and property: *3)* We use a concretization function $[\![.]\!]$ on proof terms and formulas that automatically transforms the proof at the abstract level into a *valid proof term at the concrete level*. The hypothesis and conclusion of the proof are not exactly the desired ones, so, to complete the proof, we have to discharge two proof obligations PO_1 and PO_2 shown in Figure 3. Let us put in word the proof obligation PO_1. The abstract proof depends on the axioms, say $\mathcal{A}_1^\sharp, \ldots, \mathcal{A}_n^\sharp$, resulting from the definition of the abstract stability problem $(R^\sharp, Q^\sharp)$. Similarly, the definition of the concrete problem (R, Q) generates concrete axioms. Then, the concretization process ends with the obligations to prove that the axioms of the concrete problem entails the concretization of the axioms used in the abstract proof of stability.

We show in the next section that this approach has some advantages: First, the proof obligations are expressed at the concrete level and the final proof only relies on the definitions of the actual system and property. So, the developper that attempts to produce a proof of its protocol does not have to understand the abstraction used in the verification tool. Second, depending on the property, only a subset of the abstract axioms

[1] As usual in literature on model-checking, we use the same notation to denote a predicate and the set of states that satisfy this predicate.

The three proofs of the stability property $pre_R(Q) \subseteq Q$ make use of the proof term $\nabla^\sharp$ generated by the verification tool. The two first technique are based on a proof of correctness of the abstraction. The third technique builds the concretization of the proof term, resulting in two proof obligations at the concrete level, that is, they do not refer to any abstract domains.



(1) **Correctness of an abstraction** $\alpha = (\alpha_R, \alpha_Q)$

$$\alpha, \mathcal{C}_R^\sharp, \mathcal{C}_Q^\sharp, \mathcal{C}_R, \mathcal{C}_Q \vdash \forall R \in \mathcal{C}_R,\ \forall R^\sharp \in \mathcal{C}_R^\sharp,\ \forall Q \in \mathcal{C}_Q,\ \forall Q^\sharp \in \mathcal{C}_Q^\sharp,$$
$$\Big(\alpha_R(R, R^\sharp)\ \wedge\ \alpha_Q(Q, Q^\sharp)\ \wedge\ pre_{R^\sharp}(Q^\sharp) \subseteq Q^\sharp\Big)$$
$$\Rightarrow pre_R(Q) \subseteq Q$$

(2) **Correctness of an abstraction restricted to a given** $R, R^\sharp$ **and** $Q, Q^\sharp$

$$\alpha, R^\sharp, Q^\sharp, R, Q \vdash pre_{R^\sharp}(Q^\sharp) \subseteq Q^\sharp \Rightarrow pre_R(Q) \subseteq Q$$

(3) **Two proof obligations at the concrete level**

(OP_1) **Validity of the concretization of the top-most properties** $\mathcal{A}_1^\sharp, \ldots, \mathcal{A}_n^\sharp$ **used in the** $\nabla^\sharp$ **proof of** $R^\sharp, Q^\sharp \vdash pre_{R^\sharp}(Q^\sharp) \subseteq Q^\sharp$

$$R, Q \vdash [\![\mathcal{A}_1^\sharp \wedge \ldots \wedge \mathcal{A}_n^\sharp]\!] \qquad \textit{with } [\![\phi^\sharp]\!] = \textit{concretization of } \phi^\sharp$$

(OP_2) **The concretization of the abstract property entails the concrete property**

$$R, Q \vdash [\![pre_{R^\sharp}(Q^\sharp) \subseteq Q^\sharp]\!] \Rightarrow pre_R(Q) \subseteq Q$$

Fig. 3. Three ways to lift an abstract property

are actually used in the proof and leads to proof obligations. Hence, the final proof is reduced to what is really needed.

4 Concretization of the proof of stability

This section describes the concretization process and the resulting proof obligations. An abstraction relation $\alpha \subseteq Q \times Q^\sharp$ induces a relation on the concrete domain, $\sim \subseteq Q \times Q$, defined as:

$$q \sim q' \stackrel{def}{=} \exists q^\sharp \in Q^\sharp,\ \alpha(q, q^\sharp) \wedge \alpha(q', q^\sharp)$$

Based on this remark, we present a concretization process that builds a proof of the concrete stability property from the abstract proof generated by the verification tool. The resulting proof does not mention the abstraction, neither the abstract transition system $R^\sharp$, nor the abstract property $Q^\sharp$; instead it reasons on equivalence[2] at the concrete level.

We first compute a definition of $\sim$ that is independent of the abstract domain. We exploit the predicative definition of the abstraction relation to produce a logical characterization of all abstract objects. In the case of HERMES, the abstraction relations on agents, nonces, messages, transitions are all governed by pairs of predicates (C_i, A_i) respectively defined on the concrete and abstract domain:

$$\alpha = \bigcup_{i \in [0,n]} \{(q, q^\sharp) \mid C_i(q) \wedge A_i(q^\sharp)\}.$$

The definition of $\sim$ can then be rewritten into an equivalent form:

$$\begin{array}{rcl} q \sim q' & \stackrel{def}{=} & \exists q^\sharp \in Q^\sharp,\ \alpha(q, q^\sharp) \wedge \alpha(q', q^\sharp) \\ & \equiv & \exists q^\sharp \in Q^\sharp,\ \bigvee_{i \in [0,n]} C_i(q) \wedge C_i(q') \wedge A_i(q^\sharp) \\ & \equiv & \bigvee_{i \in [0,n]} \exists q^\sharp \in Q^\sharp,\ C_i(q) \wedge C_i(q') \wedge A_i(q^\sharp) \\ & \equiv & \bigvee_{i \in [0,n]} C_i(q) \wedge C_i(q') \wedge (\exists q^\sharp \in Q^\sharp,\ A_i(q^\sharp)) \end{array}$$

Finally, a satisfiability analysis reduces each sub-formula $\exists q^\sharp \in Q^\sharp,\ A_i(q^\sharp)$ to a boolean value and the verification tools outputs *a definition of* $\sim$ *which only depends on the concrete predicates* C_i. Section E and F illustrate this principle on two abstractions used in HERMES.

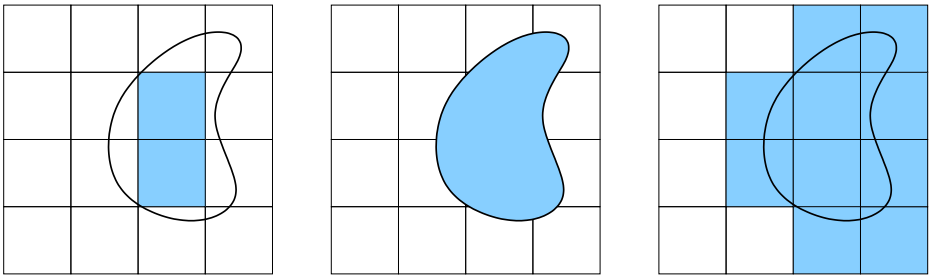
Our goal is to transform the abstract proof term into a concrete one by removing all reference to the abstract domains: the symbols of quantified variables do not need to be changed, they are domain independant;

[2] If the abstraction relation is a total function, then $\sim$ is an equivalence relation. This remark can help getting the intuition but it is not a pre-requisite of our method.

all abstract equalities can be replaced by $\sim$-equivalences ; and each abstract constant $q^\sharp$ can be replaced by a constant symbol q that satisfies the disjunction of all the $C_i(q)$ such that $A_i(q^\sharp)$ evaluates to true. These constraints on q are added as hypothesis in the concrete proof.

We now detail the replacement of abstract predicates. Intuitively, we can replace a function by another and preserve the validity of a proof as long as the two functions evaluate to the same values on equivalent arguments. We cannot simply replace an abstract predicate $P^\sharp$ by the original concrete predicate P for, in general, the predicate P does not evaluate to the value $P^\sharp(c^\sharp)$ on all the concrete values abstracted on $c^\sharp$. So, we associate to each abstract predicate $P^\sharp$ a concrete predicate P^c that is built from the original concrete predicate P, and that have a uniform valuation with respect to the relation $\sim$, meaning that $v_1 \sim v_2 \Rightarrow P^c(v_1) = P^c(v_2)$. For this purpose, we introduce two predicate transformers $^+/^-$ that define the uniform extension/restriction of a concrete predicate P with a free variable x. Figure 4 illustrates these definitions graphically.

$$P^+(x) \stackrel{def}{=} \exists y,\ y \sim x \wedge P(y) \quad \text{and} \quad P^-(x) \stackrel{def}{=} \forall y,\ y \sim x \Rightarrow P(y).$$



The abstraction induces a relation $\sim$ on the concrete domain. The grid represents a partition of the concrete domain along the equivalence classes of the relation $\sim$.

Fig. 4. Uniform restriction/extension of a predicate P with respect to a relation $\sim$

We can now define the concretization of the abstract proof terms. In order to get a valid proof term, the resulting transformation must not invalidate the proof steps of the abstract proof term. This puts strong constraints on the transformation: the concretization function $[\![.]\!]$ must be a morphism on proof terms, and it must preserve the structure of formulas. The concretization of formulas is presented in Figure 5. The interesting cases are

We use x, y for variables, c for constants, P for predicates, ϕ for formulas. The symbol v can denote either a constant value or a variable. We use $\sharp$ to denote their abstact version. All the definitions extend in the obvious way to predicates on tuples.

Concretization of formulas

$$
\begin{aligned}
[\![\neg\phi^\sharp]\!] &= \neg[\![\phi^\sharp]\!] \\
[\![\phi_1^\sharp \otimes \phi_2^\sharp]\!] &= [\![\phi_1^\sharp]\!] \otimes [\![\phi_2^\sharp]\!] \ \textit{for} \otimes \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\} \\
[\![\mathcal{Q}x\ \phi^\sharp]\!] &= \mathcal{Q}x\ [\![\phi^\sharp]\!] \qquad \textit{for } \mathcal{Q} \in \{\forall, \exists\}
\end{aligned}
$$

$$
\begin{aligned}
[\![P^\sharp(v^\sharp)]\!] &= [\![P^\sharp]\!]([\![v^\sharp]\!]) \\
[\![x]\!] &= x \quad \textit{if } x \textit{ is a symbol of variable} \\
[\![c^\sharp]\!] &= c^c \quad \textit{and} \ \ \mathsf{Hyp} := \mathsf{Hyp} \cup \left\{\ [\![\alpha(c^c, c^\sharp)]\!]\ \right\}
\end{aligned}
$$

$$
\begin{aligned}
[\![x_1 =_\mathcal{T}^\sharp x_2]\!] &= [\![x_1]\!] \sim_\mathcal{T} [\![x_2]\!] \ \textit{where } \mathcal{T} \textit{ denotes the type of } x_i \textit{ among} \\
&\qquad\qquad \textit{agents, nonces, messages, transitions} \\
[\![\alpha(x, c^\sharp)]\!] &= \textstyle\bigvee_i C_i(x) \wedge A_i(c^\sharp) \quad \textit{from the definition of } \alpha \\
&= \begin{cases} \mathit{false} & \textit{if } \bigvee_i A_i(c^\sharp) \equiv \mathit{false} \\ \Big(C_{i_1} \vee \ldots \vee C_{i_p}\Big)(x) & \textit{for } i_k \in \{i \mid A_i(c^\sharp) \equiv \mathit{true}\} \end{cases}
\end{aligned}
$$

$$
[\![P^\sharp]\!] = P^c \ \textit{with}\ \mathsf{Def} :=
$$

$$
\mathsf{Def} \cup \left\{
\begin{array}{ll}
\mathsf{let}\ P^c(\boldsymbol{x}) = P^+(\boldsymbol{x}) = \exists \boldsymbol{y}\ \ \boldsymbol{y} \sim \boldsymbol{x} \wedge P(\boldsymbol{y}) & \textit{if } P^\sharp \textit{ over-approximates } P \\
\mathsf{let}\ P^c(\boldsymbol{x}) = P^-(\boldsymbol{x}) = \forall \boldsymbol{y}\ \ \boldsymbol{y} \sim \boldsymbol{x} \Rightarrow P(\boldsymbol{y}) & \textit{if } P^\sharp \textit{ under-approximates } P \\
\mathsf{let}\ P^c(\boldsymbol{x}) = P(\boldsymbol{x}) & \textit{otherwise}
\end{array}
\right\}
$$

Concretization of a proof sentence

A proof sentences $\mathcal{T}, FV, \mathcal{H} \vdash \varphi$ mentions the theory $\mathcal{T}$, the free variables FV and the active hypothesis $\mathcal{H}$ (in this order) on the left-hand side of the deduction symbol $\vdash$, and the property φ to be proved on the right.

$$
\begin{aligned}
[\![\mathcal{T}, FV, \mathcal{H} \vdash \varphi]\!] &= [\![\mathcal{T}, FV, \mathcal{H}]\!] \vdash [\![\varphi]\!] \\
&\qquad \textit{with} \qquad [\![\mathcal{T}, FV, \mathcal{H}]\!] = \mathsf{Def}, \quad FV \cup FV(\mathsf{Hyp}), \quad \mathsf{Hyp} \cup [\![\mathcal{T}]\!] \cup [\![\mathcal{H}]\!]
\end{aligned}
$$

Concretization of an inference rules of a proof term

$$
\left[\!\!\left[\frac{\Phi_1 \ldots \Phi_n}{\Phi}\ (\mathit{rule}) \right]\!\!\right] = \frac{[\![\Phi_1]\!] \ldots [\![\Phi_n]\!]}{[\![\Phi]\!]}\ (\mathit{rule}) \qquad \textit{where } \Phi \textit{ denotes a proof sentence}
$$

Fig. 5. The concretization function

those of abstract constants and predicates. The concretization replaces the abstract constant $c^\sharp$ by a symbol c^c and produces hypothesis on c^c which express that c^c is abstracted on $c^\sharp$. Note that they only use concrete predicates. The concretization of predicates requires the advice of the developer of the verification tool to choose the right concretization. Intuitively, each symbol of an abstract predicate $P^\sharp$ is replaced by P^+ (resp. P^-, P) if $P^\sharp$ is an over (resp. under, exact) approximation of P. In practice, we do not go through the proof term to apply these replacements, we only redefine the existing predicates[3]. Eventually, the concretization function returns: 1) Def, the definitions of uniform predicates P^c that refer to the predicates P of the concrete system 2) Hyp, the hypothesis on the symbols denoting concrete constants, and 3) a proof term in which all proof steps are valid and which contains no reference to the abstract system, nor to the abstraction relation.

Figure 6 completes the picture with the construction of the concrete proof. The verification tools produces tactics which drive the COQ engine to build the abstract proof term $\nabla^\sharp$ that demonstrates the property $\varphi^\sharp$. Then, the concretization function transforms this term into a concrete proof terms $[\![\nabla^\sharp]\!]$. This term gives a proof of $[\![\varphi^\sharp]\!]$, the concretization of the abstract property. The proof $[\![\nabla^\sharp]\!]$ relies on the concretization of the abstract axioms $\mathcal{A}_1^\sharp, \ldots, \mathcal{A}_n^\sharp$ used in $\nabla^\sharp$. The first proof obligation PO_1 requires to answer the question: does the axioms of the concrete theory entails the concretization of those abstract axioms? The second obligation PO_2 requires to prove that the concretization of the abstract property $[\![\varphi^\sharp]\!]$ entails the concrete property φ.

In general, PO_2 is trivial since the abstraction has been designed to be fine enough with respect to the property φ. The application to HERMES, presented in section G, has shown that all the proof obligations can be automatically discarded.

5 Conclusion and future work

We presented a technique for producing proofs of the results generated by tools based on combination of model-checking and abstractions. The proofs obtained are independant of correctness the tool and the abstractions it uses. Our technique works in two steps. First a proof of the abstract property is generated on the abstract theory. Then this proof is syntacticaly lifted into a partial proof of the wanted property, alongside proof

[3] To avoid confusion, the concretization function pretends to replace each symbol of an abstract predicate $P^\sharp$ by an corresponding symbol of a concrete predicate P^c.

The proof terms are build with the inference rules of the sequent calculus for first order logic (see Figure 7). The symbol $\nabla^\sharp$ denotes the proof term produced by the verification tool as a certificate of the abstract property. The proof of the concrete property φ, denoted by ∇ below, uses the concretization of $\nabla^\sharp$ and three proof obligations:
(PO_3) The concretization of an abstract constants $c^\sharp$ can be any constant c^c which satisfies the conditions of the abstraction relation $\alpha(c^c, c^\sharp)$. These conditions are reduced to disjunctions of concrete predicates on c^c, then gathered into Hyp during the concretization (see Figure 5). The proof obligation PO_3 requires to prove the existence of concrete elements that satisfy the hypothesis, meaning that there exists a pre-image for each abstract constants $c^\sharp$ used in the proof $\nabla^\sharp$. The proof of an existential property is done by exhibiting a witness. It can be automated by instrumenting the verification tool so that it provides a possible concretization for each abstract constant.
(PO_1) Proof obligation PO_1 requires to show that the concrete version $\mathcal{A}^c$ of the axioms used in the abstract proof are valid for the concrete elements that satisfy the hypothesis Hyp.
(PO_2) This last proof obligation requires that the concretization of the abstract property entails the concrete property. Note that the concrete proof (and so, the proof obligations) are done in the concrete theory $\mathcal{T}$ extended with the auxilliary definitions, Def, of uniform predicates introduced during the concretization.

We use $\forall_{FV(\mathsf{Hyp})}$ (resp. $\exists_{FV(\mathsf{Hyp})}$) to denote the universal (existential) quantification on all free variables of Hyp. The step $(\dagger)$ in ∇ is valid since $FV(\mathsf{Hyp}) \cap FV(\mathcal{T} \cup \mathsf{Def}) = \emptyset$.

$$\nabla_1 \stackrel{def}{=} \left\{ \dfrac{\dfrac{PO_1}{\mathcal{T} \cup \mathsf{Def},\ \mathsf{Hyp} \vdash \mathcal{A}_1^c \wedge \ldots \wedge \mathcal{A}_n^c} \qquad \dfrac{\dfrac{\dfrac{[\![\nabla^\sharp]\!]}{[\![\mathcal{A}_1^\sharp \wedge \ldots \wedge \mathcal{A}_n^\sharp]\!] \vdash [\![\varphi^\sharp]\!]}}{\vdash [\![\mathcal{A}_1^\sharp \wedge \ldots \wedge \mathcal{A}_n^\sharp]\!] \Rightarrow [\![\varphi^\sharp]\!]}\ {\scriptstyle (1)}}{\mathcal{T} \cup \mathsf{Def},\ \mathsf{Hyp} \vdash \mathcal{A}_1^c \wedge \ldots \wedge \mathcal{A}_n^c \Rightarrow [\![\varphi^\sharp]\!]}\ {\scriptstyle (\Rightarrow_i)}}{\mathcal{T} \cup \mathsf{Def},\ \mathsf{Hyp} \vdash [\![\varphi^\sharp]\!]}\ {\scriptstyle (\Rightarrow_e)} \right.$$

$$\nabla \stackrel{def}{=} \left\{ \dfrac{\dfrac{PO_3}{\mathcal{T} \cup \mathsf{Def} \vdash \exists_{FV(\mathsf{Hyp})}\ \mathsf{Hyp}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{PO_1 \quad [\![\nabla^\sharp]\!]}{\nabla_1}}{\mathcal{T} \cup \mathsf{Def},\ \mathsf{Hyp} \vdash [\![\varphi^\sharp]\!]} \qquad \dfrac{PO_2}{\mathcal{T} \cup \mathsf{Def},\ \mathsf{Hyp} \vdash [\![\varphi^\sharp]\!] \Rightarrow \varphi}\ {\scriptstyle (\Rightarrow_e)}}{\mathcal{T} \cup \mathsf{Def},\ \mathsf{Hyp} \vdash \varphi}}{\mathcal{T} \cup \mathsf{Def} \vdash \mathsf{Hyp} \Rightarrow \varphi}\ {\scriptstyle (\Rightarrow_i)}}{\mathcal{T} \cup \mathsf{Def} \vdash \forall_{FV(\mathsf{Hyp})}\ (\mathsf{Hyp} \Rightarrow \varphi)}\ {\scriptstyle (\forall_i)\ (\dagger)}}{\mathcal{T} \cup \mathsf{Def} \vdash \varphi}\ {\scriptstyle (\exists_e,\ \forall_e,\ \Rightarrow_e)} \right.$$

In the case of HERMES we apply this proof construction with $\varphi \stackrel{def}{=} pre_R(\mathcal{D}) \subseteq \mathcal{D}$, and a theory $\mathcal{T}$ that is an axiomatization of R and $\mathcal{D}$. The proof term ∇ is the concrete proof of the stability property for the concrete problem $(R, \mathcal{D})$. The two remaining proofs obligations PO_1 and PO_2 we obtain are presented in Section G.

Fig. 6. The construction of the concrete proof

obligations. The obtained proof does not rely on the abstract theory, the abstraction or the way the tool produced the result.

In the case of HERMES, our verification tool for secrecy of cryptographic protocol, our aim was to get a fully automatic certification process. Then, in addition to the concretization, we designed tactics for the proof obligations. The obligation PO_2 is trivial and it is always discarded (see section G). Some obligations of PO_1 are resolved by our tactics but there still are protocols that show up cases which are not captured. A prototype of HERMES is available on-line from the authors' web-page [25]. When HERMES's model-checker obtains a positive verdict, its back-end produces both the abstract and concrete proofs that the protocol preserves secrecy. Both proofs are submitted to the COQ proof-checker and its outputs is presented to the user.

The continuation of this work is twofold: On the one hand, we try to build a framework that allows to compare the three proof techniques of Figure 3 and to get some sufficient conditions that ensures the validity of our proof obligations. On the other hand, we now focus on the concretization of tactics instead of the proof term in order to produce a readable concrete proof. Indeed, the users of proof-engine build their proofs at the level of tactics. With some habits the sequence of tactics becomes a quite readable proof (while proof term are not understandable). The tactic level underlines the main steps, whereas the details are treated by general powerful tactics. We experiment this idea on two challenging problems for automatic proof generation at the concrete level: First, the certification of safety property of parametric systems, *e.g.,* the bakery protocol of Lamport [12]. Our interest in this problem is to generate an inductive proof of its correctness from a proof on a bounded number of systems. The second case study is the certification of a reader/writer protocol used for implementing synchronous communication on an asynchronous platform [24]. This simple protocol ensures that the implementation respects the frontier of synchronization of the ideal synchronous model. It is interesting for the property to prove and the abstraction are not usual.

References

1. Tamarah Arons, Amir Pnueli, Sitvanit Ruah, Jiazhao Xu, and Lenore D. Zuck. Parameterized verification with automatically computed inductive assertions. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 221–234. Springer, 2001.
2. Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*, volume XXV of *Texts*

in Theoretical Computer Science. An EATCS Series. Springer, 2004. 469 p., Hardcover. ISBN: 3-540-20854-2.

3. L. Bozga, Y. Lakhnech, and M. Périn. Abstract interpretation for secrecy using patterns. In *TACAS'03*, volume 2619 of *LNCS*, 2003.
4. L. Bozga, Y. Lakhnech, and M. Périn. Hermes: An automatic tool for verification of secrecy in security protocols. In *15th International Conference on Computer Aided Verification (CAV),*, volume 2725 of *LNCS*, 2003.
5. L. Bozga, Y. Lakhnech, and M. Périn. Pattern-based abstraction for verifying secrecy in protocols. *STTT: International Journal on Software Tools for Technlogy Transfer*, 8(1):57–76, 2005.
6. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification*, LNCS, pages 154–169. Springer-Verlag, 2000.
7. Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, 1981. Springer-Verlag.
8. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2), 1983.
9. Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. Abstractions from proofs. In Neil D. Jones and Xavier Leroy, editors, *POPL*, pages 232–244. ACM, 2004.
10. Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, George C. Necula, Gregoire Sutre, and Westley Weimer. Temporal-safety proofs for systems code. In *Proceedings of the 14th International Conference on Computer-Aided Verification*, pages pp. 526–538. Lecture Notes in Computer Science 2404, Springer-Verlag, 2002.
11. Y. Lakhnech, S. Bensalem, S. Owre, and S. Berezin. Incremental verification by abstraction. In *TACAS 2001*, volume 2031 of *lncs*, 2001.
12. Leslie Lamport. A new solution of dijkstra concurrent programming problem. *Communication of the ACM*, 17(8):453–455, 1976.
13. G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using FDR. In T. Margaria and B. Steffen, editors, *Proceedings of TACAS'96 - Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, Passau, Germany*, volume 1055 of *LNCS*. Springer, March 1996.
14. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, 1995.
15. Kedar S. Namjoshi. Certifying model checkers. *Lecture Notes in Computer Science*, 2102:2–??, 2001.
16. Kedar S. Namjoshi. Lifting temporal proofs through abstractions. In *VMCAI*, 2003.
17. Doron Peled, Amir Pnueli, and Lenore D. Zuck. From falsification to verification. In Ramesh Hariharan, Madhavan Mukund, and V. Vinay, editors, *FSTTCS*, volume 2245 of *Lecture Notes in Computer Science*, pages 292–304. Springer, 2001.
18. David A. Plaisted and Yunshan Zhu. Ordered semantic hyper-linking. *J. Autom. Reason.*, 25(3):167–217, 2000.
19. Amir Pnueli, Sitvanit Ruah, and Lenore D. Zuck. Automatic deductive verification with invisible invariants. In Tiziana Margaria and Wang Yi, editors, *TACAS*, volume 2031 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2001.
20. Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *Proceedings of the 5th Colloquium on International Symposium on Programming*, pages 337–351, London, UK, 1982. Springer-Verlag.

21. Abhik Roychoudhury, C. R. Ramakrishnan, and I. V. Ramakrishnan. Justifying proofs using memo tables. In *Principles and Practice of Declarative Programming*, pages 178–189, 2000.
22. L. Tan and R. Cleaveland. Evidence-based model checking, 2002.
23. The Coq Development Team. *The Coq Proof Assistant Reference Manual Version 8.0*. Logical Project, January 2005.
24. Stavros Tripakis, Christos Sofronis, Norman Scaife, and Paul Caspi. Semantics-preserving and memory-efficient implementation of inter-task communication on static-priority or edf schedulers. In Wayne Wolf, editor, *EMSOFT*, pages 353–360. ACM, 2005.
25. `http://www-verimag.imag.fr/ Liana.Bozga/eva/hermes.php`.

$$\frac{}{\mathcal{T},\mathcal{H}\vdash\varphi}\ (hyp)\ \ \varphi\in\mathcal{H} \qquad \frac{}{\mathcal{T},\mathcal{H}\vdash\varphi}\ (axiom)\ \ \varphi\in\mathcal{T}$$

$$\frac{\mathcal{T},\mathcal{H}\vdash\bot}{\mathcal{T},\mathcal{H}\vdash\varphi}\ (\bot_e) \qquad \frac{\mathcal{T},\mathcal{H}\cup\{\varphi\}\vdash\bot}{\mathcal{T},\mathcal{H}\vdash\neg\varphi}\ (\neg_i) \qquad \frac{\mathcal{T},\mathcal{H}\vdash\varphi\quad\mathcal{T},\mathcal{H}\vdash\neg\varphi}{\mathcal{T},\mathcal{H}\vdash\bot}\ (\neg_e)$$

$$\frac{\mathcal{T},\mathcal{H}\cup\{\varphi_1\}\vdash\varphi_2}{\mathcal{T},\mathcal{H}\vdash\varphi_1\Rightarrow\varphi_2}\ (\Rightarrow_i) \qquad \frac{\mathcal{T},\mathcal{H}\vdash\varphi_1\Rightarrow\varphi_2\quad\mathcal{T},\mathcal{H}\vdash\varphi_1}{\mathcal{T},\mathcal{H}\vdash\varphi_2}\ (\Rightarrow_e)$$

$$\frac{\mathcal{T},\mathcal{H}\vdash\varphi_1\quad\mathcal{T},\mathcal{H}\vdash\varphi_2}{\mathcal{T},\mathcal{H}\vdash\varphi_1\wedge\varphi_2}\ (\wedge_i) \qquad \frac{\mathcal{T},\mathcal{H}\vdash\varphi_1\wedge\varphi_2}{\mathcal{T},\mathcal{H}\vdash\varphi_1}\ (\wedge_e) \qquad \frac{\mathcal{T},\mathcal{H}\vdash\varphi_1\wedge\varphi_2}{\mathcal{T},\mathcal{H}\vdash\varphi_2}\ (\wedge_e)$$

$$\frac{\mathcal{T},\mathcal{H}\vdash\varphi_1}{\mathcal{T},\mathcal{H}\vdash\varphi_1\vee\varphi_2}\ (\vee_i) \qquad \frac{\mathcal{T},\mathcal{H}\vdash\varphi_1}{\mathcal{T},\mathcal{H}\vdash\varphi_1\vee\varphi_2}\ (\vee_i)$$

$$\frac{\mathcal{T},\mathcal{H}\vdash\varphi_1\vee\varphi_2\quad\mathcal{T},\mathcal{H}\cup\{\varphi_1\}\vdash\varphi_3\quad\mathcal{T},\mathcal{H}\cup\{\varphi_2\}\vdash\varphi_3}{\mathcal{T},\mathcal{H}\vdash\varphi_3}\ (\vee_e)$$

$$\frac{\mathcal{T},\mathcal{H}\vdash\forall x\ \varphi}{\mathcal{T},\mathcal{H}\vdash\varphi[x\leftarrow t]}\ (\forall_e) \qquad \frac{\mathcal{T},\mathcal{H}\vdash\varphi}{\mathcal{T},\mathcal{H}\vdash\forall x\ \varphi}\ (\forall_i)\ \ x\ \textit{does not appear in}\ \mathcal{T}\cup\mathcal{H}$$

$$\frac{\mathcal{T},\mathcal{H}\vdash\varphi[x\leftarrow t]}{\mathcal{T},\mathcal{H}\vdash\exists x\ \varphi}\ (\exists_i) \qquad \frac{\mathcal{T},\mathcal{H}\vdash\exists x\ \varphi_1\quad\mathcal{T},\mathcal{H}\cup\{\varphi_1\}\vdash\varphi_2}{\mathcal{T},\mathcal{H}\vdash\varphi_2}\ (\exists_e)\ \ \begin{array}{l}x\ \textit{does not appear}\\ \textit{in}\ \mathcal{T}\cup\mathcal{H}\ \textit{nor in}\ \varphi_2\end{array}$$

Fig. 7. The inference rules of the sequent calculus

A The Needham-Schroeder-Lowe's authentication protocol

A session of the protocol distinguishes from the other sessions by its session identifier and the identities of the agents that play the protocol (two agents for NSL's protocol). We denote the parameters of a session by π, its

components are referred as S_π for the session identifier, A_π for the agent playing the role of the initiator, and B_π for the responder.

$$\mathscr{P}(\pi) \stackrel{def}{=} \left\{ \begin{array}{rl} \textit{initiator:} & \_ \rightarrow \{A_\pi, Nonce(\pi,1)\}_{K_{B_\pi}} \\ \textit{responder:} & \{x,n\}_{K_{B_\pi}} \rightarrow \{B_\pi, n, Nonce(\pi,2)\}_{K_x} \\ \textit{initiator:} & \{B_\pi, Nonce(\pi,1), n'\}_{K_{B_\pi}} \rightarrow \{n'\}_{K_{B_\pi}} \end{array} \right\}$$

where $x, n, n' \in Var$ denote free variables; K_X (resp. K_X^{-1}) is the public (resp. secret) key of the agent X and $Nonce(\pi, p)$ denote a fresh number created in Session π.

The secrets of the protocol are defined with respect to the session parameters:

$$Secret(\pi) = \{\ K_{A_\pi}^{-1},\ K_{B_\pi}^{-1},\ Nonce(\pi,2)\ \}$$

B Principle of the verification tool for secrecy of cryptographic protocols

From $\mathscr{P}$ and *Secret*, HERMES computes a set $\mathcal{D}$ that is an over-approximation of the messages that allow the intruder to deduce some of the secrets of Session π. For the Needham-Schroeder-Lowe's protocol, the resulting set $\mathcal{D}$ includes for instance:

$$\{\ \ \{i, Nonce(\pi,2)\}_{K_h} \mid i,h \in Agents,\ K_i^{-1} \in \mathcal{N} \wedge K_h^{-1} \notin \mathcal{N}\ \ \}$$

Although the secret $Nonce(\pi,2)$ is encrypted with a safe key (since $K_h^{-1} \notin \mathcal{N}$), these messages indirectly reveal the secret. Indeed, a honest agent h that takes part in an session $\pi' = (s,i,h)$ of the protocol would accept such a message as the left-hand side of the second transition. Applying the protocol, he would then respond $\{h, Nonce(\pi,2), Nonce(\pi',2)\}_{K_i}$ which can be decrypted for K_i^{-1} is known on the network. He would then involuntarily help the intruder to get the secret of Session π.

C Interpretation of the dangerous messages returned by HERMES

The dangerous messages put constraints on the knowledge of the intruder, *i.e.* the messages that belong to $\mathcal{N}$. The protocol can safely be used if $\mathcal{D} \cap \mathcal{N} = \emptyset$. Since a nonce is a fresh value created during a session of the protocol, all the dangerous messages that contains a nonce of Session π

cannot be present on the network before the beginning of that session. Hence, these constraints are removed. Eventually, there is no constraint left and we can conclude that the protocol preserves the secrets.

D The proof of secrecy preservation in Dolev and Yao's model

The transition system associated to a protocol $\mathscr{P}$ simulates an unbounded number of sessions of the protocol in the presence of an intruder that is modeled by the transitions system DY of Dolev and Yao.

$$R \stackrel{def}{=} \text{DY} \bigcup_{\pi \in Session} \mathscr{P}(\pi)$$

The certified verdict of HERMES is a formal proof in COQ of the property:

$$(P_1) \quad \forall \pi \in Session,\ \forall \mathcal{N} \subseteq Message,\\ \mathcal{D}(\pi) \cap \mathcal{N} = \emptyset \Rightarrow Secret(\pi) \cap R^*(\mathcal{N}) = \emptyset$$

where $\mathcal{D}(\pi)$ is the set of dangerous messages computed by HERMES from the protocol $\mathscr{P}$ and the secrets *Secret*. It satisfies $Secret(\pi) \subseteq \mathcal{D}(\pi)$ by construction.

The proof of (P_1) starts by a skolemization that fixes an arbitrary Session π_0. Then, we prove the implication using the principle of induction associated to the operator *pre* (see Figure 2). It requires to exhibit a set that is stable for the pre_R operator (Property P_2 below). Our verification tool, HERMES, outputs such a set $\mathcal{D}_{\pi_0}$ that we use to conclude the proof noticing that:

1. $Secret(\pi_0) \subseteq \mathcal{D}_{\pi_0}$ *by construction of* $\mathcal{D}_{\pi_0}$
2. $\forall Q, Q',\ pre_R^*(Q) \cap Q' = \emptyset \ \Leftrightarrow\ Q \cap R^*(Q') = \emptyset$ *from [14]*

The correction of these steps is proved once for all and the only proof-step specific to the given protocol and secrets is that of the stability of $\mathcal{D}_{\pi_0}$:

$$(P_2) \quad pre_R(\mathcal{D}_{\pi_0}) \subseteq \mathcal{D}_{\pi_0} \stackrel{def}{=} \forall (t, t') \in R,\ \forall \sigma,\quad t'\sigma \in \mathcal{D}_{\pi_0} \Rightarrow t\sigma \in \mathcal{D}_{\pi_0}$$

where σ denotes a substitution of variables by messages.

E The abstraction on agents in HERMES

The semantics of cryptographic protocols considers a set of infinite identities of agents which is partitioned into the honest ones that did not publish their secret key k^{-1} and the dishonest ones that revealed their keys to the intruder. The abstraction on agents maps the honest ones onto the identity $\boldsymbol{h}^\sharp$ and the dishonest ones onto $\boldsymbol{i}^\sharp$. In the concrete proof, this abstraction is replaced by an equivalence relation $\sim_A$ that only refers to the concrete semantics: $a \sim_A b \stackrel{def}{=} (k_a^{-1} \notin \mathcal{N}_{init}) \Leftrightarrow (k_b^{-1} \notin \mathcal{N}_{init})$.

Definitions of the concrete notion of agents:

- *Agent* denotes an infinite set of identities of honest and dishonest agents
- the predicate "honest" on agents is defined by: $H(a) \stackrel{def}{=} k_a^{-1} \notin \mathcal{N}_{init}$

Definition of an abstraction relation between $Agent$ **and** $Agent^\sharp = \{\boldsymbol{h}^\sharp, \boldsymbol{i}^\sharp\}$

$\alpha_A \subseteq Agent \times Agent^\sharp$ is defined by two pairs of predicates:

$$(C_1(a),\ A_1(a^\sharp)) \stackrel{def}{=} (\ \ H(a),\ a^\sharp = \boldsymbol{h}^\sharp)$$
$$(C_2(a),\ A_2(a^\sharp)) \stackrel{def}{=} (\neg H(a),\ a^\sharp = \boldsymbol{i}^\sharp)$$

then $\alpha_A = \{\ (a, \boldsymbol{h}^\sharp) \mid H(a)\ \} \cup \{\ (a, \boldsymbol{i}^\sharp) \mid \neg H(a)\ \}$

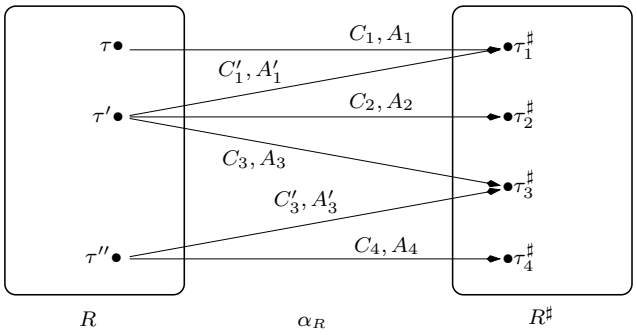
The abstraction relation α_A **induces a relation** $\sim_A \subseteq Agent \times Agent$

Note that $\sim_A$ is an equivalence relation since α_A is a total function.

$$\begin{aligned} a \sim_A b &\equiv \exists a^\sharp \in Agent^\sharp,\ \alpha_A(a, a^\sharp) \wedge \alpha_A(b, a^\sharp) \\ &\equiv (H(a) \wedge H(b)) \vee (\neg H(a) \wedge \neg H(b)) \ \equiv\ H(a) \Leftrightarrow H(b) \\ &\equiv (k_a^{-1} \notin \mathcal{N}_{init}) \Leftrightarrow (k_b^{-1} \notin \mathcal{N}_{init}) \end{aligned}$$

F The abstraction relation on transition systems used in HERMES

The abstraction relation $\alpha_R \subseteq R \times R^\sharp$ on transition systems that we use in HERMES is not a function: a transition that contains free variables can have several abstract images. Moreover, two different concrete transitions can share an abstract image. We illustate this phenomena on an ad'hoc example: the transition $\{\boldsymbol{h}^\sharp, \boldsymbol{h}^\sharp, n\}_{K_{\boldsymbol{h}^\sharp}} \rightarrow \{\boldsymbol{h}^\sharp, \{n\}_{K_{\boldsymbol{h}^\sharp}}\}_{K_{\boldsymbol{h}^\sharp}}$ belongs to the abstract images of the transitions $\{A, x, n\}_{K_A} \rightarrow \{A, \{n\}_{K_x}\}_{K_A}$ and $\{x, B, n\}_{K_B} \rightarrow \{x, \{n\}_{K_A}\}_{K_x}$. Therefore, the abstract relation depicted by the graph below can arise in HERMES for some protocols:



This abstraction relation on transition systems induces the concrete relation $\sim_R = \{(\tau, \tau'), (\tau', \tau'')\}$ between transitions that have an abstract image in common. The abstraction relation is governed by concrete predicates (C_i, A_i). The predicates A_i that bear on the abstract transition can be eliminated from the definition of $\sim_R$: they can be reduced to a boolean value by enumeration on the finite domain of abstract transitions. So, it is possible to produce a concrete definition of the relation $\sim_R$ that does not mention $R^\sharp$ or α_R:

$$\tau \sim_R \tau' \stackrel{def}{=} ((C_1 \vee C_1')(\tau) \wedge (C_1 \vee C_1')(\tau')) \vee ((C_3 \vee C_3')(\tau) \wedge (C_3 \vee C_3')(\tau'))$$

G The proof of stability generated by HERMES

Starting with a concrete transition system R, HERMES generates an abstract transition system $R^\sharp$. Then, it computes a set $\mathcal{D}^\sharp$ of dangerous messages that is stable for the operator $pre_{R^\sharp}$. Finally, it drives the COQ proof-engine to output a proof of the abstract stability property $pre_{R^\sharp}(\mathcal{D}^\sharp) \subseteq \mathcal{D}^\sharp$ and produces a predicate $\mathcal{D}^\sharp$ that characterizes this set. Then, HERMES' back-end computes the concretization of the definition of the predicate $\mathcal{D}^\sharp$ and takes the resulting formula for the definition of the concrete predicate $\mathcal{D}$. It applies the concretization function to the abstract proof term with the choices to replace $[\![R^\sharp]\!]$ by R^+ (since $R^\sharp$ is an over approximation of R) and $[\![\mathcal{D}^\sharp]\!]$ by $\mathcal{D}$ (by construction). The concretization results in a proof of the concrete stability property $pre_R(\mathcal{D}) \subseteq \mathcal{D}$ that depends on two proof obligations.

- $\boldsymbol{PO_1}$: $\mathcal{T} \cup \mathsf{Def},\ \mathsf{Hyp} \vdash \mathcal{A}_1^c \wedge \ldots \wedge \mathcal{A}_n^c$

  The abstract transition system is defined by the axiom:

  $$\forall t, t' \quad R^\sharp(t, t') \stackrel{def}{\Leftrightarrow} \exists \sigma\ (t = t_1\sigma \wedge t' = t_1'\sigma) \vee \ldots \vee (t = t_n\sigma \wedge t' = t_n'\sigma)$$

The abstract proof only use the left-to-right implication of ($\overset{def}{\Leftrightarrow}$), so we have to prove the concretization of this implication, that is,

$$\begin{aligned}&[\![\forall t, t' \;\; R^\sharp(t,t') \Rightarrow \exists \sigma \; (t = t_1^\sharp \sigma \wedge t' = t_1'^\sharp \sigma) \vee \ldots \vee (t = t_n^\sharp \sigma \wedge t' = t_n'^\sharp \sigma)]\!]\\ \equiv\; &\forall t, t' \;\; R^+(t,t') \Rightarrow \exists \sigma \; (t \sim t_1\sigma \wedge t' \sim t_1'\sigma) \vee \ldots \vee (t \sim t_n\sigma \wedge t' \sim t_n'\sigma)\end{aligned}$$

under the hypothesis $\mathsf{Hyp} = [\![\alpha_R((t_1,t_1'),(t_1^\sharp,t_1'^\sharp))]\!] \wedge \ldots \wedge [\![\alpha_R((t_n,t_n'),(t_n^\sharp,t_n'^\sharp))]\!]$

This proof obligation actually corresponds to the property $R^+ \Rightarrow [\![\textit{def. of } R^\sharp]\!]$, meaning exactly that $R^\sharp$ is an over approximation of R.

The abstract proof also uses the axiom that defines the set $\mathcal{D}^\sharp$ of dangerous messages. It produces the obligation $[\![\mathcal{D}^\sharp]\!] \Leftrightarrow [\![\textit{def. of } \mathcal{D}^\sharp]\!]$. This property holds by construction since the concretization replaces the symbol $\mathcal{D}^\sharp$ by[4] $\mathcal{D}$ and HERMES defines $\mathcal{D}$ as $[\![\textit{def. of } \mathcal{D}^\sharp]\!]$. This proof is done automatically.

• $\boldsymbol{PO_2}$: $\mathcal{T} \cup \mathsf{Def},\ \mathsf{Hyp} \vdash [\![\varphi^\sharp]\!] \Rightarrow \varphi$

$$\begin{aligned}\varphi^\sharp &\overset{def}{=} pre_{R^\sharp}(\mathcal{D}^\sharp) \subseteq \mathcal{D}^\sharp \;\equiv\; \forall t_1, t_2 \; \forall \sigma \; R^\sharp(t_1,t_2) \wedge \mathcal{D}^\sharp(t_2\sigma) \Rightarrow \mathcal{D}^\sharp(t_1\sigma)\\ [\![\varphi^\sharp]\!] &= \forall t_1, t_2 \; \forall \sigma \; [\![R^\sharp]\!](t_1,t_2) \wedge [\![\mathcal{D}^\sharp]\!](t_2\sigma) \Rightarrow [\![\mathcal{D}^\sharp]\!](t_1\sigma)\\ &\equiv \forall t_1, t_2 \; \forall \sigma \; R^+(t_1,t_2) \wedge \mathcal{D}(t_2\sigma) \Rightarrow \mathcal{D}(t_1\sigma)\\ &\equiv \forall t_1, t_2 \; \forall \sigma \left(\exists t_1', t_2' \;\; (t_1,t_2) \sim_R (t_1',t_2') \wedge R(t_1',t_2')\right) \wedge \mathcal{D}(t_2\sigma) \Rightarrow \mathcal{D}(t_1\sigma)\end{aligned}$$

The obligation PO_2 requires to prove that the above property entails the desired concrete stability property:

$$\varphi \overset{def}{=} \forall t_1, t_2 \; \forall \sigma \; R(t_1,t_2) \wedge \mathcal{D}(t_2\sigma) \Rightarrow \mathcal{D}(t_1\sigma) \;\equiv\; pre_R(\mathcal{D}) \subseteq \mathcal{D}$$

The proof of PO_2 is trivial, since $R(t_1,t_2)$ implies $R^+(t_1,t_2)$. This proof has been fully automated: it uses a meta theorem $\forall P,\ P \Rightarrow P^+$, that is proved for any predicate.

[4] In fact $[\![\mathcal{D}^\sharp]\!]$ is replaced by $\mathcal{D}^c$ with the additional definition $\mathsf{let}\ \mathcal{D}^c(x) = \mathcal{D}(x)$.