Automated Unbounded Verification of Security Protocols

Y. Chevalier and L. Vigneron *

LORIA - UHP - UN2
Campus Scientifique, B.P. 239
54506 Vandœuvre-lès-Nancy Cedex, France
E-mail: {chevalie,vigneron}@loria.fr

Abstract. We present a new model for automated verification of security protocols, permitting the use of an unbounded number of protocol runs. We prove its correctness, completeness and also that it terminates. It has been implemented and its efficiency is clearly shown by the number of protocols successfully studied. In particular, we present an attack previously unreported on the *Denning-Sacco symmetric key protocol*.

Among the methods used for studying security protocols, model-checking has been successfully used in many different ways. The common point to all these methods is to consider *principals* who exchange *messages* that have a pattern described by a *protocol*. A security protocol designer aims at providing security properties to the principals who use the protocol. These methods use an *intruder*, someone who tries to violate the security properties, to find an *attack*. The model-checking methods diverge on the handling of an infinite number of principals. In this case, the problem of whether there exists an attack is undecidable [10] if principals can create new values different at each session. Some methods [18, 13] give bounds on the number of principals that need to be considered. The drawback is that those bounds are obtained by assumptions on the shape of the messages exchanged during a protocol run.

Other methods abstract the execution of a protocol so that it is not necessary to give the number of participants. This abstraction can be done using tree-automata [11, 15, 8], or by simplifications either on the nonces creation model [4], or on the complexity of the keys used [17]. In all these models, the assumptions are done uniformly over all principals, making them less expressive than model-checking. A more complete comparison of the tools is given in Section 6.

We propose a new model for handling an infinite number of sessions by assuming two different types of principals. Some *regular* principals create nonces, and can only participate to a bounded number of runs of the protocol, the bound being given before the execution. Besides these regular principals, we add *in parallel* a finite number of *puppet* principals, who are allowed to conduct as many runs of the protocol as the *intruder* wishes them to. The model for these

* Supported by the Information Society Technologies (IST) Programme, FET Open Assessment Project IST-2000-26410, and the ACI Cryptology Vernam.

principals is simplified, and these simplifications permit to show the termination of the system as a whole. The intruder uses these puppet principals to find an attack on the regular principals. Having two different models permits to keep the expressiveness of model-checking while studying specification with an infinite number of principals.

This paper is organized as follows. We first describe the model used (Section 1), then we sketch the proof of completeness of our method in this abstracted model of the principals (Section 2). The construction of the Oracle's rules is explained in Section 3, and in Section 4 we sketch the proofs of correctness and termination of this system. We then present experimental results on the protocol library given in [7], and comparison with the results of [9] in Section 5. These experimental results include an attack previously unreported on the *Denning-Sacco symmetric key protocol*. We end this paper with a detailed comparison of our method with other ones (Section 6).
The system presented in this paper is an extension of the "lazy intruder" one given in [5] (see also [14], and [1, 16] for related theoritical results) by the addition of the Oracle's rules. Other systems using different lazy strategies (but without Oracle) are presented in [3].

All the proofs sketched here can be found in [6].

1 Model Description

1.1 Overview of the Study of a Protocol

A protocol defines roles and messages, both finite. We study a finite number of different instances of roles. The regular instances, played by honest principals, are run only once. The other instances, called puppets, are run *in parallel*. There is a finite number of them, but they can be duplicated as many times as needed, providing a potentially infinite number of runs. We model an intruder who tries to find a flaw in the regular instances with the help of those puppet instances.

Regular Principals. For a principal, there is a finite number of runs of the protocol. Considering one run for one principal, its step k in the protocol can be written: $Step_k : M \to R$, meaning that it waits for a message M and when received, sends a response R. The messages M and R depend on the knowledge of this principal, denoted w_k: this knowledge is used for analyzing M, and for composing R. Some **variables** are used in w_k for representing the parts of the received messages whose value cannot be inferred unambiguously by the principal. A step is therefore:

$$Step_k : w_k, M \to w_{next(k)}, R$$

Applying this rule to a received message, unification permits to analyze this message by recognizing its known parts. The knowledge acquired from M is added to the principal's knowledge for its next step, yielding $w_{next(k)}$.

Since there is a finite number of principals, messages and runs, the number of w terms is also finite.

Intruder. His role is to try to compose the messages m_k of the protocol using his knowledge. This is written: $\textsc{comp}(m_k)$ $\textsc{from}$ $\textsc{Know}(I)$, where I is a set of terms. Following the Dolev-Yao's model, the intruder diverts all the messages sent by the principals, and tries to compose the messages awaited by the principals (see Section 1.3).

1.2 Protocol State and User Rules

Initial state and transitions. The *current state* of a protocol is written: $B \mid\mid \mathcal{E}$. B is a multiset of terms that models both the principals (by their w term) and the knowledge of the intruder. $\mathcal{E}$ is a set of constraints of the shape:

$$\begin{array}{l}\textsc{comp}(m_{1,1}), \ldots, \textsc{comp}(m_{1,k_1}) \ \textsc{from} \ \textsc{Know}(I_1);\\ \ldots; \textsc{comp}(m_{n,1}), \ldots, \textsc{comp}(m_{n,k_n}) \ \textsc{from} \ \textsc{Know}(I_n)\end{array}$$

The *initial state* of the protocol is $B_0 \mid\mid \emptyset$, where $\emptyset$ denotes the empty set of constraints, and B_0 is a ground set containing the initial knowledge of the regular principals and of the intruder ($\textsc{Know}(I_0)$).

The actions of a principal receiving a message at step k (m_k) and replying to it (r_k) are modeled by the following rule on the current state of the protocol:

$$\begin{array}{ll}(\mathsf{Message}) & w_k, \textsc{Know}(I), B \mid\mid \mathcal{E} \rightarrow\\ & w_{next(k)}, \textsc{Know}(I \cup r_k), B \mid\mid \mathcal{E}; \textsc{comp}(m_k) \ \textsc{from} \ \textsc{Know}(I)\end{array}$$

Rule description: this rule adds in the constraints set $\mathcal{E}$ the constraint that the intruder must be able to compose the message m_k from the knowledge he has inferred so far, and that he may add the message r_k, sent by the principal and intercepted, to his knowledge.
We assume that, for all Message rules, the following inclusion of sets of variables stands: $Var(r_k) \subseteq Var(m_k) \cup Var(\mathcal{E})$
Informally, this inclusion reflects the fact that the messages sent by a principal are composed from the initial knowledge of this principal (which is ground information), the nonces created during the current protocol run (which are constants), and the messages the principal has already received. As the principal may not be able to verify all of the contents of those received messages, it may introduce variables in constraints.

Knowledge properties. Let $I_1, \ldots, I_n$ be the n intruder's knowledge multisets inserted successively in $\mathcal{E}$ after n uses of the Message rule. Let $m_1, \ldots, m_n$ be the corresponding n messages to compose. We shall note that, at the time a knowledge multiset has been inserted into the constraints set, we have:

$$I_{i+1} = I_i \cup m_i, \quad i \in \{1, \ldots, n-1\}$$

Thus, at the time the $\textsc{Know}(I_i)$ terms were inserted into the constraints set, the I_i multisets formed a growing sequence for inclusion. We shall see later that this growth property can be kept during the constraints resolution.

1.3 Constraints Resolution Rules

The intruder is modeled by a set of rules. We took the classical Dolev-Yao intruder's model, in which one assumes *a)* you have to know the corresponding key in order to decompose a cipher, *b)* the intruder may compose *any* messages from the terms he already knows. The major difference so far is that we describe the actions of an intruder trying to decompose a message in order to prove he can compose it, whereas the original Dolev-Yao rules were only concerned with adding some new terms to the knowledge of the intruder. For conciseness, we note $\text{APPLY}(t_1, t_2)$ the result of the creation of a new term from t_1 and t_2.

The construction operations that are handled are hash function application, in which case $\text{APPLY}(h, t) = h(t)$, symmetric encryption ($\text{APPLY}(t_1, t_2) = \{t_1\}^{sym}t_2$), asymmetric encryption $\{t_1\}^{pub}t_2$ and messages concatenation $< t_1, t_2 >$. We also use an operator $\text{INV}()$ which maps an asymmetric key t to the key that is able to decode a message encrypted with t. We always assume that $\text{INV}(\text{INV}(t)) = t$.

Intruder's model rules. The following rules model the intruder's possible actions:

$(\mathcal{C}_{unif})$ $T, \text{COMP}(t)$ FROM $\text{KNOW}(s \cup I); \mathcal{E} \rightarrow$
$T\sigma$ FROM $\text{KNOW}((s \cup I)\sigma); \mathcal{E}\sigma \quad (\sigma = mgu(t, s))$

$(\mathcal{C}_{dec})$ $T, \text{COMP}(\text{APPLY}(t_1, t_2))$ FROM $\text{KNOW}(I); \mathcal{E} \rightarrow$
$T, \text{COMP}(t_1), \text{COMP}(t_2)$ FROM $\text{KNOW}(I); \mathcal{E}$

$(\mathcal{A}_{pub})$ T FROM $\text{KNOW}(\{t_1\}^{pub}t_2 \cup I); \mathcal{E} \rightarrow$
$\text{COMP}(\text{INV}(t_2))$ FROM $\text{KNOW}(\{t_1\}^{pub}t_2 \cup I)$;
T FROM $\text{KNOW}(\{t_1\}^{pub}t_2 \cup t_1 \cup I); \mathcal{E}$

$(\mathcal{A}_{sym})$ T FROM $\text{KNOW}(\{t_1\}^{sym}t_2 \cup I); \mathcal{E} \rightarrow$
$\text{COMP}(t_2)$ FROM $\text{KNOW}(\{t_1\}^{sym}t_2 \cup I)$;
T FROM $\text{KNOW}(\{t_1\}^{sym}t_2 \cup t_1 \cup I); \mathcal{E}$

$(\mathcal{A}_{pair})$ T FROM $\text{KNOW}(< t_1, t_2 > \cup I); \mathcal{E} \rightarrow$
T FROM $\text{KNOW}(t_1 \cup t_2 \cup < t_1, t_2 > \cup I); \mathcal{E}$

We restrict the application of these rules to rules *applicable* on the current state.

Rules description and applicability:

- $\mathcal{C}_{unif}$: using this rule, the intruder unifies a term he knows (s) with a term he has to compose (t). This rule is *applicable* if, in the current state, neither s nor t are variables;
- $\mathcal{C}_{dec}$: it is applied when the intruder, trying to compose a term $\text{APPLY}(t_1, t_2)$, tries first to compose t_1 and t_2;
- $\mathcal{A}_{pub}$, $\mathcal{A}_{sym}$: these rules are applied when the intruder tries to decompose an encrypted term. In order to decompose this cipher, the intruder has to show he can compose the corresponding key from his current knowledge;
- $\mathcal{A}_{pair}$: this rule is applied when the intruder tries to decompose a message built by the concatenation of two terms t_1 and t_2. No assumptions on his knowledge is needed.

The last four rules are *applicable* only if $\textsc{Apply}(t_1, t_2)$ is not unified with a variable of the current state.

Oracle's rules. The former rules correspond to the Dolev-Yao's intruder model. In our model, the intruder has also the possibility to be helped by interacting with puppet principals. The main difference with regular principals is that they always create the same nonces. We will explain in Section 3 how the rules describing the results of these interactions are built. This construction ensures the property $Var(r) \subseteq \bigcup_{i=1}^{n_r} Var(m_i^r)$ for the rules describing these interactions. They are of the shape

$$\begin{array}{l}(\mathsf{Oracle}(r)) \quad B \mid\mid T, \textsc{comp}(s) \text{ FROM } \textsc{Know}(I); \mathcal{E} \rightarrow \\ \qquad B\sigma \mid\mid T\sigma, \textsc{comp}(m_1^r\sigma), \ldots, \textsc{comp}(m_{n_r}^r\sigma) \text{ FROM } \textsc{Know}(I\sigma); \mathcal{E}\sigma \\ \qquad \text{where } \sigma = mgu(r, s)\end{array}$$

meaning that for composing a term s that unifies with the term r, the intruder has to be able to compose the terms $m_1^r, \ldots, m_{n_r}^r$. The Oracle rules already take into account all the possible decompositions on the added constraints, or the use of another Oracle rule. Thus, we only need to use the $\mathcal{C}_{unif}$ rule to simplify a constraint added by an Oracle rule. The oracle rules are *applicable* if s is not unified with a variable of the current state.

Growth of the knowledge multisets. We have already noted as a knowledge property (Section 1.2) that the multisets of knowledge $(I_i)_{i \in \{1,\ldots,n\}}$ are forming a growing sequence for the inclusion ordering at the time they were added to the constraints set. Then, considering the rules used for the decomposition of intruder's knowledge, one sees that whenever a rule can be applied to I_i, it can also be applied to $I_j,\ j \geq i$. This remark leads to the following result:

Proposition 1. *Let $B \mid\mid \mathcal{E}$ be a state derived. Let T_1 FROM $\textsc{Know}(I_1)$ and T_2 FROM $\textsc{Know}(I_2)$ be two constraints inserted in $\mathcal{E}$ by the* Message *rule, and such that $I_1 \subset I_2$. If T_1 FROM $\textsc{Know}(I_1) \rightarrow^* C_1$ FROM $\textsc{Know}(I_{1,1}); \ldots; C_n$ FROM $\textsc{Know}(I_{1,n}); T_1$ FROM $\textsc{Know}(I_1')$, then there exists a sequence of transitions: T_2 FROM $\textsc{Know}(I_2) \rightarrow^* C_1$ FROM $\textsc{Know}(I_{2,1}); \ldots; C_n$ FROM $\textsc{Know}(I_{2,n}); T_2$ FROM $\textsc{Know}(I_2')$, where $I_1' \subset I_2'$ and $I_{1,i} \subseteq I_{2,i},\ i \in \{1, \ldots, n\}$.*

This proposition permits to make the following hypothesis:

($\mathcal{P}_1$) Let T_1 FROM $\textsc{Know}(I_1)$, $\ldots$, T_n FROM $\textsc{Know}(I_n)$ be the constraints inserted in the constraints set by the Message rule. By Proposition 1, we always assume that if a knowledge decomposition rule is applied on I_i, it is also applied on I_j, for all $j > i$.

Moreover, and for each state $B \mid\mid \mathcal{E}$ accessible from the initial state, the constraints set $\mathcal{E}$ satisfies the next proposition.

Proposition 2. *Let $B \mid\mid \mathcal{E}$ be an accessible state from the initial state of the protocol. For each variable $v \in \mathrm{Var}(I)$, with $\textsc{Know}(I)$ appearing in $\mathcal{E}$ or B, there exists a constraint T' FROM $\textsc{Know}(I')$ in $\mathcal{E}$ such that $v \in \mathrm{Var}(T') \setminus \mathrm{Var}(I')$.*

The proof is done by induction on the constraints resolution system steps. It relies on the fact that all variables are created by the principals in the received messages, and thus appear first in the COMP() part of a constraint.
Remark: Applying further knowledge decomposition steps if necessary, let us assume (by $(\mathcal{P}_1)$) that the knowledge multisets $I_1, \ldots, I_n$ form a growing sequence for inclusion. In this case, we write I_v the smallest knowledge multiset on which there is a constraint $\text{COMP}(t)$, with $v \in Var(t)$. The previous proposition ensures that $v \notin Var(I_v)$. In other words, in the environment, each variable first appears in the COMP() part of a constraint.

1.4 Final State for the Constraints Resolution System

A constraint is *eliminated* when the rule $\mathcal{C}_{unif}$ deletes its last COMP() term. A constraints set $\mathcal{E}$ is *satisfiable* if there exist a ground substitution σ and a sequence τ of transitions that eliminates all the constraints of $\mathcal{E}\sigma$, denoted $\sigma \vdash \mathcal{E}$. A constraints set $\mathcal{E}$ is *simple* if, for all the terms $\text{COMP}(t)$ in $\mathcal{E}$, t is a variable.

Proposition 3. *If $\mathcal{E}$ is simple, $\mathcal{E}$ is satisfiable.*

To prove this proposition, we only need to assume that the intruder knows one constant, his name for instance. Then, considering the substitution σ that maps all variables of $\mathcal{E}$ to this constant, σ is ground and $\sigma \vdash \mathcal{E}$. As a consequence, to show that all the constraints in $\mathcal{E}$ can be resolved, it is sufficient to prove that there is a sequence of transitions leading from $\mathcal{E}$ to $\mathcal{E}'$, with $\mathcal{E}'$ simple. Note that the assumption $(\mathcal{P}_1)$ does not prevent from reaching a simple constraints set.

A state $B \mid\mid \mathcal{E}$ reachable from the initial state and where $\mathcal{E}$ is simple, corresponds to a *possible state of a protocol run.*
Proposition 3 is essential for stating completeness, correctness and termination.

2 Completeness of the Constraints Resolution Rules

Let $B \mid\mid \mathcal{E}$ be a state reached from the initial state $B_0 \mid\mid \emptyset$, after applying some Message rules. We want to prove that if there exists a ground substitution σ such that $\sigma \vdash \mathcal{E}$, there exists a state $B' \mid\mid \mathcal{E}'$, reachable from $B \mid\mid \mathcal{E}$ using only *applicable* transitions, such that $\mathcal{E}'$ is simple. This proof will be obtained by showing that if there exists a sequence of transitions τ such that $B\sigma \mid\mid \mathcal{E}\sigma \rightarrow_\tau^* B''\sigma \mid\mid \emptyset$, then there also exists a sequence τ' of applications of *applicable* constraints simplification rules leading from $\mathcal{E}$ to a simple constraints set $\mathcal{E}'$.

First, we remove or alter the *applicability* condition of the rules. We prove, under these alterations, the reachability of a simple constraints set. Then, we prove that we can deduce, from the constructed transition sequence τ', a sequence of *applicable* transitions.

Alterations. We consider only the sequences of transitions that satisfy the property:

($\mathcal{P}_2$) If, in order to eliminate a constraint $\text{COMP}(t)$, one has the choice between an application of the rule $\mathcal{C}_{unif}$ and of the rule $\mathcal{C}_{dec}$, we choose the sequence with the application of $\mathcal{C}_{dec}$.

We postpone the $\mathcal{C}_{unif}$ and $\mathsf{Oracle}(r)$ applications, and as the $\mathcal{C}_{unif}$ commute, we consider blocks beginning with an application of an Oracle rule, followed by the applications of the $\mathcal{C}_{unif}$ that eliminate the newly introduced constraints.

The decomposition rules cannot be postponed, and we have to consider two cases for building the sequence τ': if the rule is *applicable*, we apply it *as is* in τ'; else, it is applied to a variable. The solution in this last case is to add a constraint describing the shape of the awaited instance of this variable, and the rule is applied in τ', using the new variables (considered as *ghost* terms) introduced in this constraint.

Completeness. The changes done in the decomposition rules now make it possible to follow a sequence of transitions τ from $\mathcal{E}\sigma$, for generating a corresponding sequence of transitions τ' from $\mathcal{E}$. But some transitions of τ may still be postponed and not applied in τ'. In the following, we first show how the postponed rules can be used to reach a simple constraints set $\mathcal{E}'$; then we show that the transitions creating ghost terms can be removed without affecting the reachability of a simple constraints set.

Once there are only postponed transitions left, we repeat the procedure described in Figure 1 until there is no suitable transition left. This procedure ends because there is only a finite number of postponed transitions.

We prove in [6] that it is always possible, when choosing a $\mathcal{C}_{unif}$ rule, to find s' in I such that s' is not a variable and can be unified with t. This is done by induction on the intruder's knowledge sets, and using ($\mathcal{P}_1$) and ($\mathcal{P}_2$).

Once the procedure of Figure 1 ends, either there are no more postponed rules and the set of constraints is empty (and therefore *simple*). Or there are still some postponed rules; the algorithm of Figure 1 implies that we have reached a simple constraints set $\mathcal{E}'$. Thus, we have found a sequence of transitions that reaches a simple constraints set $\mathcal{E}'$ from the satisfiable set $\mathcal{E}$.
This sequence of transitions may contain ghost terms. These terms are not used

1. Choose an *applicable* postponed transition:
   - either $\mathcal{C}_{unif}$, such that:
     (a) this transition is not in an Oracle rule block;
     (b) all the variables v in $Var(s \cup I)$ are in at least one constraint $T, \text{COMP}(v)$ FROM $\text{KNOW}(I_v)$ of $\mathcal{E}$ where $v \notin Var(I_v)$.
   - or $\mathsf{Oracle}(r)$.
2. **Case $\mathcal{C}_{unif}$:** find s' in I such that s' is not a variable and $s'\rho = s\rho = t\rho$; apply the transition by replacing s with s';
   Case $\mathsf{Oracle}(r)$: apply the transition and free the $\mathcal{C}_{unif}$ rules of its block;
3. Apply all the postponed transitions that become applicable.

Fig. 1. Application of postponed transitions

in the knowledge sets, and removing them does not affect the simple property of $\mathcal{E}'$. Transitions creating these terms are of no use and can be removed. Therefore we have built a sequence of applicable transitions leading to a simple constraints set.

Theorem 1 (Completeness). $\forall \mathcal{E},\ \exists \sigma,\ \sigma \vdash \mathcal{E} \ \Rightarrow\ \exists \mathcal{E}'$ *simple*, $\exists \tau',\ \mathcal{E} \rightarrow^{*}_{\tau'} \mathcal{E}'$

3 Building the Oracle Rules

We now define how to build the rules describing the knowledge the intruder may yield by interacting with *puppet* principals. There is a finite number of *different* instances of these principals, but the total number of instances is not bounded. Each puppet principal receives a message and replies to it according to the rule $w_k, M \rightarrow w_{next(k)}, R$. In these rules, variables are used to denote ambiguous values. In this case, when a compound term has an ambiguous value, it is not represented by a single variable, but by a variable at each position of this term. Moreover, the new values created by such rules depend only on the instance of the principal. Thus, there is only a finite number of constants, and variables in messages may only be instantiated by constants or other variables.
In the following, we describe an algorithm building constraints over the set of knowledge I of the intruder, so that if these constraints are satisfied, the intruder may yield a term t after interaction with the puppet principals.

Building the Rules for Receiving/Sending Messages.
In the puppet principals rules, we first eliminate the w term. This cannot be done without other changes, since we do not have, in general, $Var(R) \subseteq Var(M)$. In addition, just removing the w term would imply that the intruder can start to communicate with a principal at a step that is not the first one of the principal. The set of $M \rightarrow R$ rules are transformed using the following algorithm:

```
Let R' = ∅
For each puppet principal instance P
    Let (L_1 → R_1, ... , L_p → R_p)
    be the rules describing the received/sent messages by P.
    For i = 1 to p
        R' := R' ∪ {(⋃_{j=1}^{i} L_j) → R_i}
```

A rule $\bigcup_{j=1}^{i} L_j \rightarrow R_i$ of $\mathcal{R}'$ means that the intruder must compose all the L_j messages (the messages awaited by the principal before step i) for receiving the message R_i (sent at step i by the considered principal). These rules also have the following property: $Var(R_i) \subseteq \bigcup_{j=1}^{i} Var(L_j)$.

Building Constraint Rules over the Intruder Knowledge.
From the rules of $\mathcal{R}'$, with the help of constraints over the intruder's knowledge, we want to describe all the different ways for the intruder to get a subterm of R, for $L \rightarrow R \in \mathcal{R}'$. In this purpose, we build a new set of rules $\mathcal{R}''$ from $\mathcal{R}'$:

Let $\mathcal{R}'' = \mathcal{R}'$
Repeat
To choose $L \to R \in \mathcal{R}''$,
To choose one of the following actions:
– If $L = \text{APPLY}(t_1, t_2) \cup L'$, then $\mathcal{R}'' := \mathcal{R}'' \cup \{t_1 \cup t_2 \cup L' \to R\}$,
– If $R =< t_1, t_2 >$, then $\mathcal{R}'' := \mathcal{R}'' \cup \{L \to t_1,\ L \to t_2\}$,
– If $R = \{t_1\}^{pub} t_2$, then $\mathcal{R}'' := \mathcal{R}'' \cup \{L \cup \text{INV}(t_2) \to t_1\}$,
– If $R = \{t_1\}^{sym} t_2$, then $\mathcal{R}'' := \mathcal{R}'' \cup \{L \cup t_2 \to t_1\}$.

As there is only a finite number of rules in $\mathcal{R}'$, and since a finite number of decompositions can be applied on each rule, $\mathcal{R}''$ is finite.

Searching for all the Constraints over the Intruder Knowledge.
We first initialize $\mathcal{R}'''$ with $\mathcal{R}''' = \mathcal{R}''$. A rule $L \to R \in \mathcal{R}'''$ expresses that the intruder must compose the terms in L in order to be able to know R. Let $l \in L$. He can compose l in two ways:

1. l may be in the intruder's knowledge at the time he started to look for a term;
2. l may be composed using the knowledge given by another rule of $\mathcal{R}'''$. That is, we have to find a rule $L' \to R' \in \mathcal{R}'''$, and a substitution ρ such that $R'\rho = l\rho$. The result is that the intruder may know $R\rho$ if he is able to compose $L'\rho \cup (L \setminus l)\rho$: the rule $L'\rho \cup (L \setminus l)\rho \to R\rho$ is added in $\mathcal{R}'''$.

If we iterate 2 from an initial rule $L \to R$, we find, at every step, a set of terms the intruder has to be able to compose in order to know a term $R\rho$. However, applications of 2 may loop. In order to ensure termination, we add a subsumption rule: suppose the applications of rule 2 results in $L \to R$; if $\exists L' \to R' \in \mathcal{R}'''$, $\exists\rho$ such that $R'\rho = R$ and $L'\rho \subseteq L$, the rule $L \to R$ can be rejected. Moreover, we reject all rules $L \to R$ such that $R \in L$, as they cannot help to obtain R.

There is only a finite number of constants, therefore only a finite number of rules can be generated using 2 and subsumption. They are the **Oracle rules**. The subsumption rule does not remove any useful rule from $\mathcal{R}'''$, since for two rules $L \to R$ and $L' \to R\rho$ in $\mathcal{R}'''$, if $R\rho$ can be unified with a term t, R can be unified with the term t, and if the constraint $\{\text{COMP}(l)\}_{l \in L'}$ FROM KNOW(I) is satisfiable, then the constraint $\{\text{COMP}(l)\}_{l \in L}$ FROM KNOW(I) is also satisfiable if $L\rho \subseteq L'$.

4 Correctness and Termination

The correctness and termination of our system of rules is stated as follows:

Theorem 2 (Correctness). *If $B \mid\mid \mathcal{E} \to^* B' \mid\mid \mathcal{E}'$, with $\mathcal{E}'$ simple, then $\mathcal{E}$ is satisfiable.*

Theorem 3 (Termination). *Starting from an initial state $B_0 \mid\mid \emptyset$, the rules Message, $\mathcal{C}_{unif}$, $\mathcal{C}_{dec}$, Oracle, $\mathcal{A}_{pub}$, $\mathcal{A}_{sym}$ and $\mathcal{A}_{pair}$ can be applied only a finite number of times.*

Correctness. The correctness is proved by induction on the number of steps necessary to reach a simple constraints set $\mathcal{E}'$ from $\mathcal{E}$. At step 0, this theorem corresponds to Proposition 3; the induction is then straightforward.

Termination. For the termination, since only a finite number of Message rules can be applied, this is sufficient to prove the termination of the constraints resolution rules. Second, this is enough to prove the termination of the system without the Oracle's rules, since only $\mathcal{C}_{unif}$ can be applied on the COMP() terms created by these rules.
The application of the constraints resolution rules terminates because at each step, either the number of different variables decreases, or multisets of terms are decreasing w.r.t. the subterm ordering.

5 Experimental results

In order to test the efficiency of our system, we have analyzed a library of authentication protocols. Because of the inherent limitation of our tool, we have focused on protocols that were reported to be flawed in [9, 7].
The introduction of an Oracle partially removes this limitation, as one can indicate that some principals (puppets) may run an unbounded number of sessions in parallel. As termination is ensured, and since the simplifications done on the puppet principals help the intruder, it becomes possible to *validate* a protocol. This validation is weaker than the one described in [17, 4], since we have to provide an execution environment under which flaws are looked for. Moreover, the static analysis done while building the Oracle rules forbids to look for type flaws during the interaction between the intruder and the *puppet* principals.

The study of a protocol is done in two steps. First, a high-level protocol specification is compiled into a set of rewrite rules [12]. This protocol compiler is also used by other teams, for example in the AVISS project [2]. It can also be used to express that some principals (puppets) can be used *in parallel*. In this case, it calculates the Oracle rules. A more complete description is given in [5].

The second tool we use is daTac [19], a generic theorem prover using narrowing. The main reason for its use is that it permits to handle associative-commutative properties, used for example to express commutativity of RSA encryption. Its poor time performance should not conceal that our algorithm permits to visit only a very limited number of different states.

A Novel Attack on Denning Sacco Symmetric Key Protocol. The *Denning Sacco symmetric key protocol* was thought to be secure in several surveys [9, 7]. In this protocol, A wants to communicate with B using a secret key generated by a keyserver S. The sequence of messages is:

$$\begin{array}{lcll} A & \rightarrow & S : & A, B \\ S & \rightarrow & A : & \{B, K_{ab}, T, \{A, K_{ab}, T\}K_{bs}\}K_{as} \\ A & \rightarrow & B : & \{A, K_{ab}, T\}K_{bs} \end{array}$$

where T is a timestamp. The problem is that messages 2 and 3 are of the same global shape. This fact can be exploited in the following sequence of messages:

$$\begin{array}{|l|}\hline I(B) \rightarrow S : B, A \\ S \rightarrow I(B) : \{A, K_{ab}, T, \{B, K_{ab}, T\}K_{as}\}K_{bs} \\ I(A) \rightarrow B : \{A, K_{ab}, T, \{B, K_{ab}, T\}K_{as}\}K_{bs} \quad (\equiv \{A, K_{ab}, T\}K_{bs}) \\ \hline \end{array}$$

After this sequence of messages, B accepts a new value for the symmetric key he shares with A, whereas A is not aware a protocol run took place. This attack relies on a type flaw, and this kind of attack is sometimes considered to be dubious. In this case, one shall note this is very unlikely that B, as he receives the whole message, considers $T, \{B, K_{ab}, T\}K_{as}$ to be a timestamp. But the implementation of this protocol could lead to a real flaw in two cases:

1. in a flawed implementation, B might not check the tail of the last message. Since the last two messages begin with the same data type, there is a real risk of confusion here;
2. if a bloc-ciphering algorithm such as DES is used, the second message may be split into parts, from which the intruder will be able to construct a message acceptable by B, however cautious B is.

Therefore this flaw can occur in the implementation of this protocol. Its detection permits to give guidelines for the practical design of the protocol.

6 Comparison and Conclusion

Although there are still protocols that cannot be expressed in our high-level language, one shall note that our approach, in comparison with other classical results, seems rather efficient, and for two reasons.

Firstly, a look at Table 1 shows that we are able to correctly find out whether a protocol is flawed. Moreover, we were able to find type flaws in addition to other flaws. This is the case, for example, of the *Otway Rees protocol*. In contrast with Lowe's approach, we do not have to specify the type flaw looked for.

Secondly, one shall note that we do not find any "artifact" error. In the case of the *Yahalom protocol*, we do not find any flaw. In this case, the flaw reported in [7] cannot occur in our setting, because the last message of the attack cannot be composed by the Intruder. We do not claim to find *all* type flaws, since some of them rely on the associative property of the pairing. We only consider that pairing is right-associative, which permits us to find type flaws in the *tail* of messages (as in the *Denning-Sacco symmetric key protocol* case).

The tool used by Lowe in [9] is probably one of the best, up to date, for the complete study of security protocols. In contrast with our tool, it does not handle type flaws and is restricted to bounded number of sessions systems.
Blanchet proposed in [4] a tool that also permits model-checking with an unbounded number of principals. On one hand, this method handles nonces better than in our Oracle's rules, and it does not require a scenario. On the other hand, it cannot be used to detect replay attacks, short term secrecy and, in general, everything that is session dependent. For example, in this nonces setting, *every* protocol is subject to a replay attack.

Table 1. Comparison of our approach with others.

Protocol	Our tool	Lowe	Clark&Jacob	Brackin	Time
Protocols using symmetric encryption					
ISO Sym.Key One-Pass Unilat. Auth. Proto.	Attack	Attack	No attack	No attack	2s
ISO Sym. Key Two-Pass Mutual Auth. Proto.	Attack	Attack	No attack	No attack	14s
Andrew Secure RPC Protocol	Attack	Attack	Attack	Attack	14s
Davis Swick Private Key Certificates, Proto. 1	Attack	Attack	Attack	Attack	32s
Davis Swick Private Key Certificates, Proto. 2	Attack	Attack	Attack	Attack	1073s
Davis Swick Private Key Certificates, Proto. 3	Attack	Attack	No attack	No attack	6s
Davis Swick Private Key Certificates, Proto. 4	Attack	Attack	No attack	No attack	80s
Denning Sacco Symmetric Key Protocol	Attack	No attack	No attack	No attack	6s
Needham Schroeder Proto. with Convent. Key	Attack	No attack	Attack	Attack	20s
Otway Rees Key exchange Protocol	Attack	Attack	Attack	No attack	12s
Yahalom Protocol	No attack	No attack	Attack	Attack	
Woo&Lam Auth. Protocol Π_1	Attack	Attack	Attack	No attack	4s
Woo&Lam Auth. Protocol Π_2	Attack	Attack	Attack	No attack	3s
Woo&Lam Auth. Protocol Π_3	Attack	Attack	Attack	No attack	3s
Woo&Lam Auth. Protocol Π	Attack	Attack	Attack	No attack	2s
Woo&Lam Mutual Auth.	Attack	Attack	Attack	No attack	635s
Needham Schroeder Signature Protocol	Attack	Attack	No attack	Attack	18s
Kao Chow Repeated Auth. Protocol	Attack	No attack	Attack	Attack	4s
Kehne Langendorfer Schoenewalder	Attack	Attack	Attack	No attack	4s
Neumann Stubblebine	Attack	Attack	Attack	No attack	3s
Protocols using hash functions					
ISO One-Pass Unilat. Auth. Proto. with CCFs	Attack	Attack	No attack	No attack	3s
ISO Two-Pass Mutual Auth. Proto. with CCFs	Attack	Attack	No attack	No attack	5s
Protocols using public key encryption					
ISO Public Key One-Pass Unilat. Auth. Proto	Attack	Attack	No attack	No attack	4s
ISO Public Key Two-Pass Mutual Auth. Proto	Attack	Attack	No attack	No attack	10s
Needham Schroeder Public Key Protocol	Attack	Attack	Attack	No attack	7s
SPLICE/AS Auth. Protocol	Attack	Attack	Attack	No attack	12s
Hwang&Chen's mod. SPLICE/AS Auth. Proto	Attack	Attack	Attack	No attack	21s
Denning Sacco Key Distrib. with Public Key	Attack	Attack	Attack	No attack	57s
Shamir Rivest Adelman Three Pass Protocol	Attack	Attack	Attack	Unanalyzed	5s
Encrypted Key Exchange Protocol	Attack	Attack	Attack	No attack	6s
TMN Key Exchange Protocol	Attack	Unanalyzed	Unanalyzed	Unanalyzed	80s

The method used in Athena [17] also has the advantage of not requiring the declaration of instances of principals, and can handle an unbounded number of principals. Its drawback are that this method needs atomic keys, which make it inapplicable to protocols like SSL, and it requires that keys used for encryption are known by the receiver. This rules out protocols like SET Card-holder registration, which uses fresh keys in every message. Finally, Athena is not able to handle type flaws.

About these last two systems, we can note that the one described in [4] does not ensure termination because of problems that might arise from the *shape* of messages of the protocol. On the other hand, the model of [17] does not ensure termination because of an unrestricted nonce model. We have ensured termination by restricting the model of nonces and by rejecting type flaws during the building of the Oracle's rules. We feel that the model for the puppet principals can be improved toward either one or the other setting, thus permitting to study a protocol without having to specify instances for those puppet principals.

Our model was developped as part of the AVISS project [2] in the CL part, but is also partly used in the on-the-fly model-checker. Results obtained by both systems show its efficiency.

References

1. R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *CONCUR'2000*, pages 380–394, 2000.
2. A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Vigano, and L. Vigneron. The AVISS Security Protocol Analysis Tool. In *CAV'02*, 2002. Tool presentation.
3. D. Basin. Lazy Infinite-State Analysis of Security Protocols. In R. Baumgart, editor, *Secure Networking — CQRE (Secure)'99*, LNCS 1740, pages 30–42, Heidelberg, Germany, 1999. Springer-Verlag.
4. B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop*, June 2001.
5. Y. Chevalier and L. Vigneron. Towards Efficient Automated Verification of Security Protocols. In *Verification Workshop (VERIFY'01) (in connection with IJCAR'01), Università degli studi di Siena, TR DII 08/01*, pages 19–33, 2001.
6. Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. Research Report 4369, Institut National de Recherche en Informatique et Automatique, Nancy (France), January 2002.
7. J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0, 17 Nov. 1997. URL http://www.cs.york.ac.uk/ jac/papers/drareview.ps.gz.
8. H. Comon, V. Cortier, and J. Mitchell. Tree Automata with one Memory, Set Constraints and Ping-Pong Protocols. In *28th Int. Coll. Automata, Languages, and Programming (ICALP'2001)*, LNCS 2076, pages 682–693. Springer, 2001.
9. B. Donovan, P. Norris, and G. Lowe. Analyzing a Library of Security Protocols using Casper and FDR. In *W. on Formal Methods and Security Protocols*, 1999.
10. N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of Bounded Security Protocols. In *FLOC'99 Workshop on Formal Methods and Sec. Protocols (FMSP'99)*, 1999.
11. J. Goubault-Larrecq. A Method for Automatic Cryptographic Protocol Verification. In *IPDPS 2000 Workshops, Cancun, Mexico*, LNCS 1800, pages 977–984. Springer, May 2000.
12. F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In M. Parigot and A. Voronkov, editors, *Proceedings of LPAR'2000*, LNCS 1955, pages 131–160, Heidelberg, 2000. Springer-Verlag.
13. G. Lowe. Towards a Completeness Result for Model Checking of Security Protocols. *Journal of Computer Security 7*, 1, 1999.
14. J. Millen and V. Shmatikov. Constraint Solving for Bounded-Process Cryptographic Protocol Analysis. In *8th ACM Conference on Computer and Communication Security*, pages 166–175, November 2001.
15. D. Monniaux. Abstracting Cryptographic Protocols with Tree Automata. In *6th Int. Static Analysis Symposium (SAS'99)*, LNCS 1694. Springer-Verlag, 1999.
16. M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 2001.
17. D. Song, S. Berezin, and A. Perrig. Athena, a Novel Approach to Efficient Automatic Security Protocol Analysis. *J. of Computer Security*, 9((1,2)):47–74, 2001.
18. S. Stoller. A Bound on Attacks on Authentication Protocols. Technical Report 526, Indiana University, Computer Science Dept, February 2000.
19. L. Vigneron. Positive Deduction modulo Regular Theories. In Hans Kleine-Büning, editor, *Computer Science Logic*, LNCS 1092, pages 468–485, Berlin, 1995. Springer-Verlag. URL: http://www.loria.fr/equipes/protheo/SOFTWARES/DATAC/.