INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Automated Unbounded Verification of Security Protocols

Yannick Chevalier — Laurent Vigneron

N° 4369

Janvier 2002

THÈME 2

Rapport de recherche

ISSN 0249-6399

Automated Unbounded Verification of Security Protocols *

Yannick Chevalier[†] , Laurent Vigneron[‡]

Thème 2 — Génie logiciel
et calcul symbolique
Projet Protheo

Rapport de recherche n° 4369 — Janvier 2002 — 16 pages

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.*

Key-words: security protocols, verification, lazy strategy, unbounded runs

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

[†] LORIA – Université Henri Poincaré
[‡] LORIA – Université Nancy 2

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Vérification non bornée automatique de protocoles de sécurité

Résumé : Nous présentons un nouveau modèle pour la vérification automatique de protocoles de sécurité, permettant l'utilisation d'un nombre non borné de sessions. Nous démontrons sa correction, sa complétude, ainsi que sa terminaison. Il a été implanté et son efficacité est clairement démontrée de par le nombre de protocoles étudiés avec succès. En particulier, nous présentons un attaque jusque là non rapportée du protocole de Denning-Sacco avec clé symétrique.

Mots-clés : protocoles de sécurité, vérification, stratégie paresseuse, infinité de sessions

Among the methods used for studying security protocols, model-checking has been successful 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 therefore 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 every sessions. Some methods [18, 14] give bounds on the number of principals that it is needed to consider. The drawback is that those bounds are obtained by making 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 by using tree-automata [11, 16, 7] or, in a closer framework, by simplifications either on the nonces creation model [4], or on the complexity of the keys used [17]. The common point of all these models is that the same assumption is done on all principals, making these models less expressive than model-checking. We refer to Section 7 for a more complete comparison of the tools.

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 they can only participate to a bounded number of runs of the protocol, the bound being given before execution. Besides these regular principals, we add *in parallel* a finite number of *different* principals, who are allowed to conduct as many runs of the protocol as the *intruder* wishes them to. The model for these principals is simplified, and these simplifications permit to show the termination of the system as a whole. The intruder uses these simplified 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 give the sketch of the proofs of correctness and completeness of our method in this abstracted model of the principals (Sections 2, 3). The construction of the Oracle's rules is explained in Section 4, and in Section 5, we prove the termination of this system. We then present experimental results on the protocol library given in [6], and comparison with the results of [9] in Section 6. 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 7).

The system presented in this paper is an extension of the "lazy intruder" one given in [5] (see also [15]) by the addition of the Oracle's rules. Other systems using lazy strategies for the intruder (but without Oracle) are presented in [3].

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 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 that tries to find a flaw in the regular instances with the help of those parallel runs.

1.1.1 Regular Principals.

For a principal, there is a finite number of run of the protocol. Considering one run for one principal, its step k 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.

1.1.2 Intruder.

His role is to try to compose the messages m_k of the protocol using his knowledge, written: $\textsc{compose}(m_k)$ FROM $\textsc{Know}(I)$. His knowledge, written $\textsc{Know}(I)$ where I is a set of terms, may be initial one or acquired one. 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

1.2.1 Initial state and transitions.

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

$$\textsc{compose}(m_{1,1}) \cdot \ldots \cdot \textsc{compose}(m_{1,k_1}) \text{ FROM } \textsc{Know}(I_1)$$
$$: \ldots : \textsc{compose}(m_{n,1}) \cdot \ldots \cdot \textsc{compose}(m_{1,k_n}) \text{ FROM } \textsc{Know}(I_n)$$

The *initial state* of the protocol is $\emptyset, B_0$, where $\emptyset$ denotes the empty set of constraints, and B_0 is a ground set containing the initial knowledge of the principals and of the intruder.

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

$$(\mathsf{Message}) \quad \mathcal{E},\ w_k.\textsc{Know}(I).B \to$$
$$\mathcal{E} : \textsc{compose}(m_k) \text{ FROM } \textsc{Know}(I),\ w_{next(k)}.\textsc{Know}(I \cup r_k).B$$

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 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 it may not be able to verify all of the contents of those received messages, it introduces variables in them.

1.2.2 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 composed messages. 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 [8], in which one assumes *a)* you have to know the corresponding key in order to decompose a cipher (perfect encryption hypothesis), *b)* the intruder may compose 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 $\textsc{Apply}(t_1, t_2)$ the result of the creation of a new term from t_1 and t_2.

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

1.3.1 Intruder's model rules.

The following rules model the intruder's possible actions:

$$
\begin{array}{ll}
(\mathcal{C}_{unif}) & T \cdot \textsc{compose}(t) \textsc{ from Know}(s \cup I) : \mathcal{E} \rightarrow \\
 & T\sigma \textsc{ from Know}((s \cup I)\sigma) : \mathcal{E}\sigma \quad (\sigma = mgu(t,s)) \\
(\mathcal{C}_{dec}) & T \cdot \textsc{compose}(\textsc{Apply}(t_1,t_2)) \textsc{ from Know}(I) : \mathcal{E} \rightarrow \\
 & T \cdot \textsc{compose}(t_1) \cdot \textsc{compose}(t_2) \textsc{ from Know}(I) : \mathcal{E} \\
(\mathcal{A}_{pub}) & T \textsc{ from Know}((\{t_1\}^{pub}t_2)^{ndec} \cup I) : \mathcal{E} \rightarrow \\
 & \textsc{compose}(\textsc{inv}(t_2)) \textsc{ from Know}((\{t_1\}^{pub}t_2)^{dec} \cup I) \\
 & : T \textsc{ from Know}((\{t_1\}^{pub}t_2)^{dec} \cup (t_1)^{ndec} \cup I) : \mathcal{E} \\
(\mathcal{A}_{sym}) & T \textsc{ from Know}((\{t_1\}^{sym}t_2)^{ndec} \cup I) : \mathcal{E} \rightarrow \\
 & \textsc{compose}(t_2) \textsc{ from Know}((\{t_1\}^{sym}t_2)^{dec} \cup I) \\
 & : T \textsc{ from Know}((\{t_1\}^{sym}t_2)^{dec} \cup (t_1)^{ndec} \cup I) : \mathcal{E} \\
(\mathcal{A}_{pair}) & T \textsc{ from Know}((< t_1, t_2 >)^{ndec} \cup I) : \mathcal{E} \rightarrow \\
 & T \textsc{ from Know}((t_1)^{ndec} \cup (t_2)^{ndec} \cup (< t_1, t_2 >)^{dec} \cup I) : \mathcal{E}
\end{array}
$$

In these rules, $\_^{dec}$ (*decomposed*) and $\_^{ndec}$ (*not decomposed yet*) are marks that are used to prevent the multiple applications of one rule to the same term. Since they do not interfere with unification, we shall forget about them until the study of termination of this system (Section 5).

Rules description:

- $\mathcal{C}_{unif}$: using this rule, the intruder unifies a term he knows with a term he has to compose;
- $\mathcal{C}_{dec}$: it is applied when the intruder, trying to compose a term $\textsc{Apply}(t_1, t_2)$, 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.

None of these rules is applied to a variable.

1.3.2 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 non-regular principals. The counterpart is that they always create the same nonces. We will explain in Section 4 how the rules describing the results of these interactions are built, and we will show the following property:

$$\mathit{Var}(r) \subseteq \bigcup_{i=1}^{n_r} \mathit{Var}(m_i^r)$$

The rules describing these interactions are of the shape:

$$
\begin{array}{ll}
(\mathsf{Oracle}(r)) & T \cdot \textsc{compose}(s) \textsc{ from Know}(I) : \mathcal{E},\ B \rightarrow \\
 & T\sigma \cdot \textsc{compose}(m_1^r\sigma) \cdot \ldots \cdot \textsc{compose}(m_{n_r}^r\sigma) \textsc{ from Know}(I\sigma) : \mathcal{E}\sigma,\ B\sigma \\
 & \hfill (\sigma = mgu(r,s))
\end{array}
$$

$m_1^r, \ldots, m_{n_r}^r$ are terms the intruder has to be able to compose in order to obtain a term r that can be unified with the term s he tries to compose. 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.

1.3.3 Growth of the knowledge multisets.

We have already noted as a knowledge property (Section 1.2) that the multisets of knowledge $(I_n)_{n\in\{1,\dots,N\}}$ 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 $I_j,\ j \geq i$.

Let T_1 FROM KNOW(I_1) and T_2 FROM KNOW(I_2) be two constraints inserted in the environment by the Message rule, such that $I_1 \subset I_2$. One easily sees that all decomposition rules that can be applied from I_1 can also be applied from I_2, that is if T_1 FROM KNOW$(I_1) \rightarrow^* \mathcal{E}_1 : T_1$ FROM KNOW(I'_1), then we also have T_2 FROM KNOW$(I_2) \rightarrow^* \mathcal{E}_1 : T_2$ FROM KNOW(I'_2), with $I'_1 \subset I'_2$.

Proposition 1 *Let $\mathcal{E}, B$ be a state derived from the initial state, and let T_1 FROM KNOW(I_1) and T_2 FROM 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 KNOW$(I_1) \rightarrow^*$ C_1 FROM KNOW$(I_{1,1}) : \dots : C_n$ FROM KNOW$(I_{1,n}) : T_1$ FROM KNOW(I'_1), then there exists a sequence of transitions: T_2 FROM KNOW$(I_2) \rightarrow^*$ C_1 FROM KNOW$(I_{2,1}) : \dots : C_n$ FROM KNOW$(I_{2,n}) : T_2$ FROM KNOW(I'_2), where $I'_1 \subset I'_2$ and $I_{1,i} \subseteq I_{2,i},\ i \in \{1,\dots,n\}$.*

Proof: The inclusion is true at the time the constraints are inserted into the constraints set. It remains true by induction on the constraints resolution rules, by following each decomposition. □

For the sequel, we consider the following property:

($\mathcal{P}_1$) Let T_1 FROM KNOW$(I_1), \dots, T_n$ FROM 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, whenever $j > i$.

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

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

Proof: First, we note that this property is true for the initial state, since in this state, $\mathcal{E} = \emptyset$.

Let us assume this property is true for each state $\mathcal{E}, B$ accessible from the initial state in less than $n-1$ steps, and let us consider the transition $\mathcal{E}, B \rightarrow \mathcal{E}', B'$. Depending on the rule applied for this transition, let us check that the property is still valid:

- Oracle(r): using the assumptions made on the Oracle(r) rules, we always have:

$$Var(r) \subseteq \bigcup_{i=1}^{n_r} Var(m_i^r)$$

Thus, after the application of the substitution σ, variables that were in s are now distributed among the terms COMPOSE(m_1^r), ... , COMPOSE$(m_{n_r}^r)$;
- $\mathcal{C}_{unif}$: the proposition being true for all variables in I by hypothesis, it will remain true after the application of the substitution;
- $\mathcal{C}_{dec}$, $\mathcal{A}_{pub}$, $\mathcal{A}_{sym}$, $\mathcal{A}_{pair}$: all those decomposition rules preserve the variables in the knowledge multiset, so the property being true for $\mathcal{E}, B$, it still is for $\mathcal{E}', B'$;
- Message: by induction hypothesis, the property is true for all the variables of I. Each variable of $Var(r_k)$ is either in $Var(\mathcal{E})$, or in $Var(m_k) \setminus Var(\mathcal{E})$. In the first case, the induction hypothesis applies. In the second case, the variable cannot be in $Var(I)$ since by induction hypothesis it would also be in $Var(\mathcal{E})$; so the variable is only in COMPOSE(m_k). Hence in this constraint, the variable appears in the COMPOSE() part, and not in the KNOW() one.

We have just proved that this proposition is true by induction on the transitions. □

Remark: Applying further knowledge decomposition steps if necessary, let us assume (by ($\mathcal{P}_1$)) that the knowledge multisets $I_1, \dots, I_n$ form a growing sequence for inclusion. In this case, we write I_v the smallest knowledge multiset on which there is a constraint COMPOSE(t), with $v \in Var(t)$. The previous proposition ensures that $v \notin Var(I_v)$. In other words, in the environment, all variables first appear in the COMPOSE() part of a constraint.

1.4 Final State

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 $\textsc{compose}(t)$ 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, we only have to prove that there is a sequence of transitions that leads from $\mathcal{E}$ to $\mathcal{E}'$, with $\mathcal{E}'$ simple. And the *final state* of the protocol is $\mathcal{E}', B'$ where $\mathcal{E}'$ is simple.
We can also note that the assumption done after Proposition 1 does not prevent from reaching a simple constraints set.
Proposition 3 will be essential for stating the correctness and the completeness of our system.

2 Correctness of the Constraints Resolution Rules

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

Theorem 1 (Correctness) *If $\mathcal{E}, B \rightarrow^* \mathcal{E}', B'$, with $\mathcal{E}'$ simple, then $\mathcal{E}$ is satisfiable.*

This result is proved by induction on the number of steps necessary to reach a simple constraints set, $\mathcal{E}'$, from $\mathcal{E}$: if this number is 0, this theorem corresponds to Proposition 3; then the induction is based on the fact that all the rules are correct.

3 Completeness of the Constraints Resolution Rules

Let $\mathcal{E}, B$ be a state reached from the initial state $\emptyset, B_0$, 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 $\mathcal{E}', B'$, reachable from $\mathcal{E}, B$, such that $\mathcal{E}'$ is simple. This proof will be obtained by showing that if there exists a sequence of transitions τ such that $\mathcal{E}\sigma, B\sigma \rightarrow_\tau^* \emptyset, B''\sigma$, then there also exists a sequence τ' of applications of constraints simplification rules that leads from $\mathcal{E}$ to a simple constraints set $\mathcal{E}'$.

The main difficulty is that, in the next section, we have to give some restrictions on the application of the constraints simplification rules and of the knowledge decomposition rules. But then, we first remove (for the decomposition rules) or alter (for the rule $\mathcal{C}_{unif}$) these restrictions. Then, we show that the added restrictions are of no use to guarantee the existence of a sequence that leads from $\mathcal{E}$ to $\mathcal{E}'$, where $\mathcal{E}'$ is simple.

3.1 Restrictions and their Alteration

3.1.1 Choosing the sequence τ of transitions.

First, and among all the sequences of transitions that lead from $\mathcal{E}\sigma$ to $\emptyset$, we consider only the ones that satisfy the following property:

($\mathcal{P}_2$) If, in order to eliminate a constraint $\textsc{compose}(t)$, 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 $\mathcal{C}_{dec}$.

3.1.2 Alteration of the order of transitions in τ.

By hypothesis, there exists at least one sequence τ that leads from $\mathcal{E}\sigma$ to $\emptyset$. Now, consider an application of $\mathcal{C}_{unif}$ in this sequence. Since $\mathcal{E}\sigma$ is ground, the only effect of this transition is to remove a $\textsc{compose}()$ term. This means that all subsequent rules will be applied on the remaining terms. Therefore the application of the rule $\mathcal{C}_{unif}$ can be "postponed" until we want it to be applied. Of course, all the applications of this rule in τ commute.

So, starting with a sequence τ of transitions satisfying ($\mathcal{P}_2$) that leads from $\mathcal{E}\sigma$ to $\emptyset$, we postpone all the applications of $\mathcal{C}_{unif}$

$$(\mathcal{C}_{unif}) \qquad \begin{array}{l} T \cdot \textsc{compose}(t) \text{ from } \textsc{Know}(s \cup I) : \mathcal{E} \rightarrow \\ T\rho \text{ from } \textsc{Know}((s \cup I)\rho) : \mathcal{E}\rho \quad (\rho = mgu(t, s)) \end{array}$$

until neither t nor s is a variable in $\mathcal{E}$. In the other cases, $\mathcal{C}_{unif}$ is applied in τ'.
For the same reasons, it is also possible to postpone applications of the rules $\mathsf{Oracle}(r)$. But in this case, we need also to ensure that the applications of $\mathcal{C}_{unif}$ on the COMPOSE() terms created by an Oracle rule, and postponed, will be applied *after* that Oracle rule.
In this purpose, and since the applications of the rules $\mathcal{C}_{unif}$ commute one with another, we will consider blocks beginning with an application of an Oracle rule, immediately followed by the applications of the $\mathcal{C}_{unif}$ rules that eliminate the newly introduced constraints.

3.1.3 Application of the decomposition rules.

We want to be able to follow, from $\mathcal{E}$, one sequence of transitions that leads from $\mathcal{E}\sigma$ to $\emptyset$. But we may have to apply a decomposition rule to a variable, which is forbidden. We cannot simply postpone those rules, since another rule in the sequence of transitions τ may be applied on a term created by such a decomposition rule.

The solution is to consider two kinds of applications of a decomposition rule: *1)* the application in a term that is not a variable, this transition can be done in τ'; *2)* the application in a variable, which is handled by the following mechanism.

We introduce projections π_1 and π_2, which have the following property:

$$\pi_i(\textsc{Apply}(t_1, t_2)) = t_i, \; i \in \{1, 2\}$$

We handle a set $\mathcal{L}$ of constraints of the shape: $x = \pi_i(y), \; i \in \{1, 2\}$, x and y variables.

Those constraints play an important role for altering the application of decomposition rules: when one of the rules $\mathcal{C}_{dec}$, $\mathcal{A}_{pub}$, $\mathcal{A}_{sym}$, or $\mathcal{A}_{pair}$ is applied to a variable x (representing for instance a term $\textsc{Apply}(t_1, t_2)$), we create two new variables, x_1 and x_2, we add in $\mathcal{L}$ the constraints $x_1 = \pi_1(x)$ and $x_2 = \pi_2(x)$, and we apply the rule in τ' using x_1 and x_2 instead of t_1 and t_2. The two constraints are eliminated as soon as x is instantiated to a composed term. Moreover, for the rule $\mathcal{C}_{dec}$, we keep the term COMPOSE(x) as long as x is a variable. This does not change the fact whether the constraints set is simple or not.

Marking the "non-existing" terms. If x is in a constraint $x = \pi_i(y), \; i \in \{1, 2\}$ of $\mathcal{L}$, this means y has not yet been instantiated in τ'. We mark these terms x in order to remember that they are not currently derived from $\mathcal{E}$. Note that if a term is marked, it is a variable.

3.2 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 addition, there may be some marked terms in the constraints set $\mathcal{L}$. 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 terms that are marked in $\mathcal{E}'$ can be removed without affecting the reachability of a simple constraints set.

3.2.1 Handling of the postponed transitions.

Once there is 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.

The main difficulty of this algorithm is to prove 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. In order to prove the existence of s', let $T_s \cdot \textsc{compose}(s)$ FROM $\textsc{Know}(I_s)$ be a constraint where $s \notin \mathit{Var}(I_s)$ (Proposition 2). Let i and j be such that the constraint $T_s \cdot \textsc{compose}(s)$ FROM $\textsc{Know}(I_s)$ originated from the insertion of T_i FROM $\textsc{Know}(I_i)$ in the constraints set, and $T.\textsc{compose}(t)$ FROM $\textsc{Know}(s \cup I)$ originated from the insertion of T_j FROM $\textsc{Know}(I_j)$ in the constraints set. Since $s \notin \mathit{Var}(I_i)$ and $s \in \mathit{Var}(I_j)$, we have $j > i$.
If COMPOSE(t) is a constraint introduced during the knowledge decomposition of a subset $I_k, k \leq i$, of I_j, there exists a constraint COMPOSE(t) FROM $\textsc{Know}(I')$ originating from I_k. By Proposition 1, we have $I' \subset s \cup I$.
Else, and *a)* since COMPOSE(s) FROM $\textsc{Know}(I_s)$ cannot be a constraint introduced during the knowledge decomposition of I_i; and *b)* as all decomposition rules have been applied (by $(\mathcal{P}_1)$), we can assume $I_s \subset s \cup I$.

1. Choose a postponed transition:
   - $(\mathcal{C}_{unif})$ $\quad T \cdot \textsc{compose}(t)$ $\textsc{from}$ $\textsc{Know}(s \cup I) : \mathcal{E} \rightarrow$
     $T\rho$ $\textsc{from}$ $\textsc{Know}((s \cup I)\rho) : \mathcal{E}\rho \quad (\rho = mgu(t, s))$
     such that:
     (a) this transition is not in an Oracle rule block;
     (b) t is not a variable (so s is a variable);
     (c) all the variables v in $\mathit{Var}(s \cup I)$ are in at least one constraint $T \cdot \textsc{compose}(v)$ $\textsc{from}$ $\textsc{Know}(I_v)$ of $\mathcal{E}$, and $v \notin \mathit{Var}(I_v)$.
   - $(\mathsf{Oracle}(r))$ $\quad T \cdot \textsc{compose}(s)$ $\textsc{from}$ $\textsc{Know}(I) : \mathcal{E},\ B \rightarrow$
     $T\rho \cdot \textsc{compose}(m_1^r\rho) \cdot \ldots \cdot \textsc{compose}(m_{n_r}^r\rho)$ $\textsc{from}$ $\textsc{Know}(I\rho) : \mathcal{E}\rho,\ B\rho$
     $(\rho = mgu(r, s))$
     such that s is not a variable.
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.

Figure 1: Application of postponed transitions

We note that in both cases, the multisets inclusion and the condition $(\mathcal{P}_2)$ on the transition τ imply that the constraint found must but resolved, in τ, by using $\mathcal{C}_{unif}$. That is, we have in both cases found a subset I'' of I which does not contains s, and which contains a term s'' that can be unified with t. Iterating if necessary, we find a term s' that is not a variable and can be unified with t, as it is requested.

Once the procedure of Figure 1 ends, if there are no more postponed rules, all the transitions of τ have been applied, and the set of constraints is empty, and therefore *simple*. Note that in this case there are no more marked terms.

If there are still some postponed rules, the conditions for choosing transitions in the procedure of Figure 1 imply 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 marked terms. We now have to show that the transitions creating those marked terms are of no use.

3.2.2 Elimination of the marked terms.

The marked terms that are left are variables in the knowledge multiset or in a $\textsc{compose}()$ term. All the variables left in the knowledge multiset have not been used by a $\mathcal{C}_{unif}$ or an Oracle rule in the sequence of transitions from $\mathcal{E}$ to $\mathcal{E}'$. Thus, they can safely be removed from the knowledge multisets.

The remaining $\textsc{compose}(x)$ in $\mathcal{E}'$, where x is a marked variable, can also be removed as this does not affect the *simple* property of $\mathcal{E}'$. The variables in those terms do not appear in any term in the knowledge multisets, since they are unique. Thus, they can safely be removed from the constraints set $\mathcal{E}'$ by not using all the transitions that created them.

3.2.3 Conclusion.

Let $\mathcal{E}$ be a satisfiable set of constraints. We have described how to construct a sequence of transitions from $\mathcal{E}$ to a simple constraints set $\mathcal{E}'$. Therefore, the system is complete.

Theorem 2 $\forall \mathcal{E},\ \exists \sigma,\ \sigma \vdash \mathcal{E} \ \Rightarrow\ \exists \mathcal{E}'$ *simple*, $\exists \tau',\ \mathcal{E} \rightarrow_{\tau'}^{*} \mathcal{E}'$

4 Building the Oracle Rules

We now define how to build the rules describing the knowledge the intruder may yield by interacting with principals *in parallel*. There is a finite number of *different* instances of these principals, but the total number of instances is not bounded.

Each principal in parallel receives a message and replies to it according to the rule $w_k.M \to w_{next(k)}.R$. Like for the rules given in Section 1.1.1, 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 variables at every position of this term. Moreover, the new values created by such a rule 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.
Our goal is to define constraints over the set of knowledge I of the intruder, so that if these constraints are satisfied, the intruder may yield, after interaction with the principals in parallel, a term t.
The following sections describe an algorithm building these constraints.

4.1 Building the Rules for Receiving/Sending Messages

In the parallel 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. To remedy this, let us call $\mathcal{R}$ the set of $M \to R$ rules. These rules are transformed using the following algorithm:

```
Let $\mathcal{R}' = \emptyset$
For each principal instance $P$
    Let $\{L_1 \to R_1, \ldots, L_p \to R_p\} \subseteq \mathcal{R}$
    be the rules describing the received/sent messages by $P$.
    For $i = 1$ to $p$
        $\mathcal{R}' := \mathcal{R}' \cup \{(\bigcup_{j=1}^{i} L_j) \to R_i\}$
```

A rule $\bigcup_{j=1}^{i} L_j \to R_i$ 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). We have: $Var(R_i) \subseteq \bigcup_{j=1}^{i} Var(L_j)$.

4.2 Building Constraint Rules over the Intruder Knowledge

From the rules of $\mathcal{R}'$, with constraints over the intruder's knowledge, we want to describe all the different ways for the intruder to get a subterm of R, for $L \to R \in \mathcal{R}'$. 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 decomposition can be applied on each rule, $\mathcal{R}''$ is finite.

4.3 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 $L \rightarrow R$; if $\exists L' \rightarrow R' \in \mathcal{R}'''$, $\exists \rho$ such that $R'\rho = R$ and $L'\rho \subseteq L$, the rule $L \rightarrow R$ can be rejected. Moreover, we reject all rules $L \rightarrow R$ 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 $L \rightarrow R$ and $L' \rightarrow R\rho$ 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 $\{\text{COMPOSE}(l)\}_{l \in L'}$ FROM $\text{KNOW}(I)$ is satisfiable, then the constraint $\{\text{COMPOSE}(l)\}_{l \in L}$ FROM $\text{KNOW}(I)$ is also satisfiable if $L\rho \subset L'$.

5 Termination

We now prove that our system terminates. First, there is only a finite number of Message rules that can be applied. Thus, it is sufficient to prove the termination of the constraints resolution rules. Second, it is enough to prove the termination of the system without the Oracle's rules, since only $\mathcal{C}_{unif}$ can be applied on the COMPOSE() terms created by these rules.

We have specified that constraints resolution rules shall be applied on variables. Thus, after each application of one of these rules, the number of variables in $\mathcal{E}$ will either remain constant (all rules except maybe $\mathcal{C}_{unif}$), or decrease (if the rule $\mathcal{C}_{unif}$ is applied using a non-trivial substitution). Therefore, it is only possible to apply the rule $\mathcal{C}_{unif}$ a *finite* number of times with a non-trivial substitution.

Consider now the other rules. Let $<$ be the subterm ordering on terms (that is, $t < t'$ if t is a subterm of t'), and let $\prec$ be its extension to multisets of terms. The decomposition rules are decreasing for $\prec$ on the multisets of knowledge. The order $<$ is well-founded as long as no non-trivial substitution is applied, therefore there is only a finite number of knowledge decomposition rules applied between each non-trivial application of $\mathcal{C}_{unif}$.

Finally, the $\mathcal{C}_{dec}$ and $\mathcal{C}_{unif}$ rules with no substitution decrease also the multiset of terms in COMPOSE().

We can sum this up by saying that all rules (except the Oracle's rules) are decreasing for the quadruple $(V, M^{ndec}, M^{dec}, M^{\text{COMPOSE()}})$ ordered lexicographically, where V stands for the number of different variables, M^{ndec} and M^{dec} are the multisets of terms marked with $\_^{ndec}$ and $\_^{dec}$ resp., and $M^{\text{COMPOSE()}}$ is the multiset of terms in all COMPOSE() terms.

This implies that the constraints resolution system using Oracle's rules terminates.

6 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, 6].

The introduction of an Oracle partially removes this limitation, as one can indicate that some principals may run an unbounded number of sessions in parallel. As termination is ensured, and since the simplifications done on the principals in parallel 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 looking for attacks using type flaws during the interaction between the intruder and the principals *in parallel*.

6.1 Tools used

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]. The high-level specification permits to express authenticate goals as defined by Lowe [13], as well as secrecy and short-term secrecy goals. It can also be used to express that some principals can be used *in parallel*. In this case, it calculates the rules for the Oracle. 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. It can be used, for example, for expressing commutativity of RSA encryption. The drawback of this is the poor time performance of our tool, compared to other tools currently used [17, 2, 4]. However, this poor time performance should not conceal that our algorithm permits to visit only a very limited number of different states.

Table 1: Comparison of our approach with others

Protocol	Our tool	Lowe	Clark & Jacob	Brackin	Time
Protocols using symmetric encryption					
ISO Symmetric Key One-Pass Unilateral Auth. Protocol	Attack	Attack	No attack	No attack	2s
ISO Symmetric Key Two-Pass Mutual Auth. Protocol	Attack	Attack	No attack	No attack	14s
Andrew Secure RPC Protocol	Attack	Attack	Attack	Attack	14s
Davis Swick Private Key Certificates, Protocol 1	Attack	Attack	Attack	Attack	32s
Davis Swick Private Key Certificates, Protocol 2	Attack	Attack	Attack	Attack	1073s
Davis Swick Private Key Certificates, Protocol 3	Attack	Attack	No attack	No attack	6s
Davis Swick Private Key Certificates, Protocol 4	Attack	Attack	No attack	No attack	80s
Denning Sacco Symmetric Key Protocol	Attack	No attack	No attack	No attack	6s
Needham Schroeder Protocol with Conventional 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 and Lam Auth. Protocol Π_1	Attack	Attack	Attack	No attack	4s
Woo and Lam Auth. Protocol Π_2	Attack	Attack	Attack	No attack	3s
Woo and Lam Auth. Protocol Π_3	Attack	Attack	Attack	No attack	3s
Woo and Lam Auth. Protocol Π	Attack	Attack	Attack	No attack	2s
Woo and 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 Unilateral Auth. Protocol with CCFs	Attack	Attack	No attack	No attack	3s
ISO Two-Pass Mutual Auth. Protocol with CCFs	Attack	Attack	No attack	No attack	5s
Protocols using public key encryption					
ISO Public Key One-Pass Unilateral Auth. Protocol	Attack	Attack	No attack	No attack	4s
ISO Public Key Two-Pass Mutual Auth. Protocol	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 and Chen's mod. SPLICE/AS Auth. Protocol	Attack	Attack	Attack	No attack	21s
Denning Sacco Key Distribution 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

6.2 A Novel Attack on Denning Sacco Symmetric Key Protocol

The *Denning Sacco symmetric key protocol* was thought to be secure in several surveys [9, 6]. It was conceived as a correction of the *Needham Schroeder symmetric key protocol.* The sequence of messages, in this protocol, is:

$$\begin{array}{lcll} A & \rightarrow & S & : A, B \\ S & \rightarrow & A & : \{B, Kab, T, \{A, Kab, T\}Kbs\}Kas \\ A & \rightarrow & B & : \{A, Kab, T\}Kbs \end{array}$$

where T is a timestamp. In this protocol, A forwards to B a part of the message sent by the server S. The problem, in this protocol, is that messages 2 and 3 are of the same global shape. This fact can be exploited in the following sequence of messages:

$$\begin{array}{lcll} I(B) & \rightarrow & S & : B, A \\ S & \rightarrow & I(B) & : \{A, Kab, T, \{B, Kab, T\}Kas\}Kbs \\ I(A) & \rightarrow & B & : \{A, Kab, T, \{B, Kab, T\}Kas\}Kbs \quad (\equiv \{A, Kab, T\}Kbs) \end{array}$$

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 $T, \{B, Kab, T\}Kas$ to be a timestamp. But the implementation of this protocol could lead to a real flaw in two cases:

1. First, in case there is some padding during the encryption, it is possible that B just does not check what happens after the part of the message he is interested in. Since the two ciphers begin with the same data type, there is a real possibility for B to accept the message, thus leading to the flaw;

2. Second, it may happens that, if a bloc-ciphering algorithm is used (such as DES, for example), the second message may be split into smaller parts, from which the intruder will be able to construct a message acceptable by B, however cautious B is. This will be the case if a bloc ends right after T.

Thus, we believe this type flaw should not be regarded as an artifact, and that it should be considered for an actual implementation of this protocol.

7 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 first look at Table 1 shows that we are able to correctly find out whether a protocol is flawed. But this first look also hides the fact that some protocol may have multiples errors, and in this case, our tool permits to find all of them. This happens on the *Otway Rees key exchange protocol*, where Lowe finds an authentication error, and Clark&Jacob report a type flaw (Brackin does not find any flaw). We are able to find both flaws in this case, thanks to the expressiveness of the goal we use, and to the lazy intruder model, which permits to catch type flaws. The point, here, is that we were able to automatically find a type flaw, whereas another model-checking method was unable to do so.

Secondly, one shall note that we do not find any "artifact" errors. The case arises on the *Yahalom key exchange protocol.* Our approach of the intruder permits to consider only messages that can "really" be exchanged. 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* case).

The tool used by Lowe in [9] is probably one of the best, up to date, for the complete study of security protocols. Its only weaknesses are that it does not handle type flaws and is restricted to finite state space systems.

Blanchet proposed in [4] a tool that also permits model-checking with an unbounded number of principals. On one hand, his 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.
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 messages. 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 one described in [17] does not ensure termination because of a nonce model that is not restricted. 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 principals *in parallel* can be improved toward either one or the other setting, thus permitting to study a protocol without having to specify instances for those principals in parallel.
Our model is used in the AVISS project [2], in our implementation, but also partly in an on-the-fly model-checker by another group. The results obtained by both systems show the efficiency of this model.

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. Bouallag, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Vigano, and L. Vigneron. The AVISS Security Protocol Analysis Tool. Submitted to CAV'02 as Tool presentation.

[3] David Basin. Lazy infinite-state analysis of security protocols. In Rainer Baumgart, editor, *Secure Networking — CQRE (Secure)'99*, LNCS 1740, pages 30–42. Springer-Verlag, Heidelberg, Germany, 1999.

[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 *Proceedings of the Verification Workshop (VERIFY'01) (in connection with IJCAR'01), Università degli studi di Siena, TR DII 08/01*, pages 19–33, 2001.

[6] J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0, 17. Nov. 1997. URL: `http://www.cs.york.ac.uk/$\sim$jac/papers/drareview.ps.gz`.

[7] 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)*, volume 2076 of *Lecture Notes in Computer Science*, pages 682–693. Springer, 2001.

[8] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[9] B. Donovan, P. Norris, and G. Lowe. Analyzing a Library of Security Protocols using Casper and FDR. In *Proceedings of the Workshop on Formal Methods and Security Protocols.* 1999.

[10] N. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proceedings of the FLOC'99 Workshop on Formal Methods and Security Protocols (FMSP'99).* 1999. Available at `http://www.cs.bell-labs.com/who/nch/fmsp99/program.html`.

[11] J. Goubault-Larrecq. A method for automatic cryptographic protocol verification. In *Proc. 15 IPDPS 2000 Workshops*, volume 1800 of *Lecture Notes in Computer Science*, pages 977–984, Cancun, Mexico, 2000. Springer.

[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, La Réunion, France, 2000. Springer-Verlag.

[13] G. Lowe. A Hierarchy of Authentication Specifications. In *Proceedings of the 1997 IEEE Computer Security Symposium on Research in Security and Privacy*, pages 31–43. 1997.

[14] G. Lowe. Towards a Completeness Result for Model Checking of Security Protocols. In *Journal of Computer Security 7*, volume 1. 1999.

[15] 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.

[16] D. Monniaux. Abstracting cryptographic protocols with tree automata. In *Sixth International Static Analysis Symposium (SAS'99)*, volume 1694 of *Lecture Notes in Computer Science*. Springer, 1999.

[17] D. Song, S. Berezin, and A. Perrig. Athena, a Novel Approach to Efficient Automatic Security Protocol Analysis. *Journal of Computer Security*, 9((1,2)):47–74, 2001.

[18] S. Stoller. A bound on attacks on authentication protocols. Indiana University, Computer Science Dept, Technical Report 526, February 2000.

[19] L. Vigneron. Positive Deduction modulo Regular Theories. In Hans Kleine-Büning, editor, *Proceedings of Computer Science Logic*, LNCS 1092, pages 468–485. Springer-Verlag, Berlin, 1995. URL: `www.loria.fr/equipes/protheo/SOFTWARES/DATAC/`.

Contents

1 Model Description 3

1.1 Overview of the Study of a Protocol 3

1.1.1 Regular Principals. 3

1.1.2 Intruder. 4

1.2 Protocol State and User Rules 4

1.2.1 Initial state and transitions. 4

1.2.2 Knowledge properties. 4

1.3 Constraints Resolution Rules 4

1.3.1 Intruder's model rules. 5

1.3.2 Oracle's rules. 5

1.3.3 Growth of the knowledge multisets. 6

1.4 Final State 7

2 Correctness of the Constraints Resolution Rules 7

3 Completeness of the Constraints Resolution Rules 7

3.1 Restrictions and their Alteration 7

3.1.1 Choosing the sequence τ of transitions. 7

3.1.2 Alteration of the order of transitions in τ. 7

3.1.3 Application of the decomposition rules. 8

3.2 Completeness 8

3.2.1 Handling of the postponed transitions. 8

3.2.2 Elimination of the marked terms. 9

3.2.3 Conclusion. 9

4 Building the Oracle Rules 9

4.1 Building the Rules for Receiving/Sending Messages 10

4.2 Building Constraint Rules over the Intruder Knowledge 10

4.3 Searching for all the Constraints over the Intruder Knowledge 10

5 Termination 11

6 Experimental results 11

6.1 Tools used 11

6.2 A Novel Attack on Denning Sacco Symmetric Key Protocol 13

7 Comparison and Conclusion 13

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399