APPENDIX 1.2.4

R. Amadio and W. Charatonik. On name generation and set-based analysis in Dolev-Yao model (extended abstract). In *Proc. CONCUR'02*, volume 2421 of *LNCS*. Springer Verlag, 2002.

On Name Generation and Set-Based Analysis in the Dolev-Yao Model

Roberto M. Amadio[1] and Witold Charatonik[2,3]

[1] Laboratoire d'Informatique Fondamentale, Marseille
amadio@cmi.univ-mrs.fr
[2] Max-Planck-Institut für Informatik, Saarbrücken
[3] Uniwersytet Wrocławski, Wrocław
witold@mpi-sb.mpg.de

Abstract. We study the control reachability problem in the Dolev-Yao model of cryptographic protocols when principals are represented by tail recursive processes with generated names. We propose a conservative approximation of the problem by reduction to a non-standard *collapsed* operational semantics and we introduce checkable syntactic conditions entailing the equivalence of the standard and the collapsed semantics. Then we introduce a conservative and decidable *set-based* analysis of the collapsed operational semantics and we characterize a situation where the analysis is exact.

Keywords: cryptographic protocols, name generation, verification, set constraints.

1 Introduction

We study the control reachability problem for cryptographic protocols in the Dolev-Yao model [DY83] which is nowadays a widely used model abstracting the behaviour of cryptographic functions.

Most cryptographic protocols are programs of finite length without loops and then the control reachability problem can be solved in NPTIME [ALV01, RT01]. However, these finite programs usually can be executed any number of times in different sessions. In every session a participant may generate fresh *names* representing, *e.g.*, nonces or keys. A number of attacks are then possible relying on messages exchanged in previous or parallel sessions. Unfortunately, introducing recursive behaviours in cryptographic protocols leads quickly to an undecidable verification problem.

It has been observed by Durgin *et al.* [DLMS99] that name generation leads to undecidability of control reachability even when the height of the messages considered is *bounded*. When name generation is *not* allowed then the control reachability problem is still undecidable in general, and decidable in certain particular cases allowing a limited use of the pairing construct (also known as *ping-pong* protocols; see [DEK82]). To complete this picture, we show in [AC02] that the control reachability problem is undecidable *without pairs* and *with bounded height* messages when name generation is allowed.

It then appears that in order to obtain a decidable class of protocols we have to further restrict the use of name generation. The approach we explore in this paper is to bound the number of *parallel* sessions generating new names. This entails that there is a bound on the number of *fresh* names 'used' at any time by the principals.

Technically, we will concentrate on *tail-recursive* processes including an operation of *name generation*. In the standard operational semantics, whenever we execute a name generation instruction νc, we associate to c a fresh constant. Thus, during the execution the name c may get assigned constants from a countable set $C_0, C_1, C_2, \ldots$ In this paper, we introduce a non-standard *collapsed* operational semantics for name generation whose intuition is as follows: when a principal starts a new *session* a name generated in the session gets assigned a *new* constant while names generated in previous sessions are collapsed to an *old* constant. Thus a name generator νc operates over the finite set of constants $\{C^{old}, C^{new}\}$.

In our formal model, looping back models starting a new session. Then the collapsed semantics is a partial formalization of the idea that names generated in previous sessions can be confused. We note that the logical *length* of a session can be suitably adapted by unfolding the program. [1] As long as the behaviour of principals does *not* depend on name inequality, the collapsed semantics 'simulates' the standard one, *i.e.*, reachability of a control point in the standard semantics implies reachability of the same control point in the concrete one. Moreover, we introduce checkable syntactic conditions that guarantee the equivalence of the standard and the collapsed semantics (section 2).

Next, we provide a set-based analysis of the collapsed semantics. From an algorithmic point of view, the fact that the collapsed semantics operates over a *finite* signature is exploited to associate to a system of processes Eq a system of set constraints Φ_{Eq} such that the reachability of a control state in Eq implies the nonemptiness of a distinguished variable in the least solution of Φ_{Eq}. It turns out that the set constraints generated are a variant of those studied in [CP97, TDT00] so that the nonemptiness problem can be solved by a suitable adaptation of standard techniques (section 3). In particular, we give a general construction that handles the renaming operators introduced by the collapsed semantics and we point out a 'linear' subclass of definite set constraints that can be solved in PTIME.

In section 4, we characterize a situation –without name generation– where the set based analysis is exact and obtain a new decidable class of cryptographic protocols with a complexity ranging between simple and double exponential time. With name generation, the set-based analysis is conservative but not necessarily exact. In practice, it is difficult to find examples where the loss of precision is

[1] Clearly, if we adopted *iteration* rather than tail recursion, then it would be more difficult to design a sensible collapsed semantics; as shown by Stoller [Sto99] an attack may require exponentially many parallel sessions.

essential; an artificial example is given in section 4. A long version of this paper is available as a research report [AC02]: it includes all the proofs that have to be omitted here for lack of space and it also illustrates how our model can be used to formalize the standard property of *secrecy* as well as a more elusive property of *freshness*.

Related work We restrict our attention to authors who considered fully automatic methods to prove correctness of recursive cryptographic protocols in the Dolev-Yao model.

Monniaux [Mon99] introduces tree automata (which are related to set-constraints) to represent the knowledge of the adversary. In his study, he restricts his attention to protocols without recursion, but the approach has been generalized to handle recursion by the following authors. He already notices the lack of precision of the set based analysis pointed out in example 2.

At about the same time, Weidenbach [Wei99] applies the Spass theorem prover to the analysis of a protocol by Neuman and Stubblebine. He reduces the verification problem to a proof search in monadic Horn clauses (again related to set-constraints). There is no general guarantee of termination for his method but some decidable subclasses are pointed out. Name generation is modelled by skolemisation –looking at the name generator as an existential– but no result is reported on the soundness or completeness of this modelling with respect to a standard operational semantics of name generation.

Later, Genet and Klay [GK00] and Goubault [Gou00] also rely on tree automata to produce a conservative approximation of cryptographic protocols. Genet and Klay consider a rather rough approximation where every action can be performed at any time and point out in their conclusion the need for a refined analysis of the control (a programme we carry on to some extent here). Goubault proposes to approximate a name generator by a superset of all values that it can generate in a run (which is quite different from the collapsed semantics studied here). In these two papers, no definite analysis is given of the complexity and the precision of the approximation schema.

Finally, Comon et al. [CCM01] introduce a special class of tree automata *with memory* that allows to decide in Dexptime secrecy properties for a class of cryptographic protocols strictly containing the ping-pong protocols. This work does not cover the notion of name generation and the class of protocols considered is *incomparable* with the one considered in theorem 4; their class goes beyond regular tree languages but, because of the so called *basicness condition*, only particular input filters are admitted.

2 Model

We begin by recalling the basic assumptions in the Dolev-Yao model (section 2.1). Then we introduce a particular family of tail recursive processes, their operational semantics, and the related control reachability problem (section 2.2). We propose a certain number of syntactic conditions and exhibit related fragments of the model where the control reachability problem is still undecidable (section 2.3). Finally, we define a collapsed semantics simulating the standard one and determine a checkable condition where the collapsed semantics is precise (section 2.4).

2.1 The Dolev-Yao Model

We recall that in the Dolev-Yao model communications are mediated by an adversary that can analyse the messages exchanged and synthesize new ones. To represent the set of messages we assume an infinite set of constants $\mathcal{N}$ and consider terms over the (infinite) signature $\Sigma = \mathcal{N} \cup \{E^2, \langle \_, \_ \rangle^2\}$. Thus we have two binary constructors: E for encryption and $\langle \_, \_ \rangle$ for pairing.

We use the following standard notation: $x, y, \ldots$ for (term) variables; V for the set of variables; $T_\Sigma(V)$ for the collection of finite terms over $\Sigma \cup V$; $t, t', \ldots$ for terms in $T_\Sigma(V)$; $\boldsymbol{t}$ for vectors of terms; $[t/x]$ for the substitution of t for x. We denote with $\mathit{Var}(t)$ the variables occurring in the term t.

The set of *messages* $\mathcal{M}$ is defined as the least set that contains $\mathcal{N}$ and such that: (1) if $t \in \mathcal{M}$ and $t' \in \mathcal{N}$ then $E(t, t') \in \mathcal{M}$ and (2), if $t, t' \in \mathcal{M}$ then $\langle t, t' \rangle \in \mathcal{M}$.

We abbreviate a message $E(\cdots E(t, D_n), \ldots, D_1)$ with $D_1 \cdots D_n t$ thus regarding nested encryptions as *words*. The functions S for *synthesis* and A for *analysis* are closure operators over the power set of messages $\mathcal{M}$ defined as follows:

- $S(T)$ is the least set that contains T and such that: (1) if $t_1, t_2 \in S(T)$ then $\langle t_1, t_2 \rangle \in S(T)$ and (2) if $t_1 \in S(T), t_2 \in T \cap \mathcal{N}$ then $E(t_1, t_2) \in S(T)$.
- $A(T)$ is the least set that contains T and such that: (1) if $\langle t_1, t_2 \rangle \in A(T)$ then $t_i \in A(T)$, $i = 1, 2$ and (2) if $E(t_1, t_2) \in A(T)$, $t_2 \in A(T) \cap \mathcal{N}$ then $t_1 \in A(T)$.

For the sake of simplicity, in this paper we restrict our attention to *symmetric* encryption where encryption and decryption keys coincide and moreover we make the rather standard assumption that keys are atomic names, *i.e.*, a pair or an encrypted message cannot be used as a key. However, our results are not strictly dependent on these hypotheses and we expect that they can been adapted to, *e.g.*, *public* keys and *complex* symmetric keys.

2.2 Tail-Recursive Processes

In a message, it is useful to distinguish between *data* and *key* positions.

Definition 1. *Let u, v be variables or constants and let $t \in T_\Sigma(V)$ be a term. We define two predicates $Occ_{Data}(u, t)$ and $Occ_{Key}(u, t)$ that are satisfied if u occurs in t in* Data *or* Key *position, respectively:*

$$
\begin{aligned}
&(1)\ Occ_{Data}(u, u),\\
&(2)\ Occ_{Data}(u, \langle t_1, t_2 \rangle) \text{ if } Occ_{Data}(u, t_i),\ i = 1 \text{ or } i = 2,\\
&(3)\ Occ_{Data}(u, E(t, v)) \text{ if } Occ_{Data}(u, t).\\
&(4)\ Occ_{Key}(u, \langle t_1, t_2 \rangle)\ \text{ if } Occ_{Key}(u, t_i),\ i = 1 \text{ or } i = 2,\\
&(5)\ Occ_{Key}(u, E(t, v))\ \text{ if } u = v \text{ or } Occ_{Key}(u, t).
\end{aligned}
$$

Remark 1. Note that u may occur in t without occurring in either data or key position. This may happen only when no ground instance of t can produce a message, *i.e.*, when t contains a subterm $E(t_1, t_2)$ and t_2 is neither a constant nor a variable. Note also that we cannot syntactically outlaw such terms since they may be created from correct messages by communication, *e.g.,* $E(t_1, t_2)$ may be created from $E(t_1, x)$ by a communication that assigns the term t_2 to the variable x.

We fix a finite system Eq of k equations:

$$A_i = \nu \boldsymbol{c}_i \, Q_i \text{ where } Q_i \equiv ?t^i_1.!s^i_1 \ldots ?t^i_{l_i}.!s^i_{l_i}.A'_i \text{ for } i = 1, \ldots, k,\ l_i \geq 1 \tag{1}$$

with distinct process identifiers A_i, $i = 1, \ldots, k$ and where A'_i is either A_i or a special state err. The following definitions of configuration and reduction will refer to *this* system. The intuitive semantics is as follows: A_i generates a vector of fresh names $\boldsymbol{c}_i$ (this models the generation of fresh nonces or keys), it engages in an alternating sequence of input-output where it receives messages of the shape t^i_j and emits messages of the shape s^i_j, and finally either it loops back or it reaches an erroneous state. Example 1 shows how to translate a usual protocol description into such a system of equations. In general, we would introduce one equation for each participant and each role it plays in a protocol. If an arbitrary number of participants is allowed then a preliminary phase of abstraction is needed.

We will say that the terms t^i_j are *filters* and define the *filter variables* as

$$FVar(t^i_j) = Var(t^i_j) \setminus \bigcup_{h=1,\ldots,j-1} Var(t^i_h)\ .$$

Similarly, the *key variables* are defined by $KVar(s^i_j) = \{z \in Var(s^i_j) \mid Occ_{Key}(z, s^i_j)\}$.

We suppose that there is always a ground instance of the terms t^i_j, s^i_j that can produce a message. If such a ground instance does not exist, then the thread will be blocked at the occurrence of the term, and an equivalent system is obtained by inserting at a corresponding position a filter which can be never satisfied.

The *filters* t^i_j must correspond to a combination of projections and decryptions. For this purpose, we require that filter variables occur in data position. Namely, if $x \in FVar(t^i_j)$ then x must occur in t^i_j in data position (cf. definition 1). This condition forbids, *e.g.*, to set $t^i_1 = E(x, y)$, which in the operational semantics presented below would allow a principal to decrypt an encrypted message without knowing the key.

Finally, we require that the variables in an output are contained in the variables of the preceding input, *i.e.*, $Var(s^i_j) \subseteq FVar(t^i_j)$. We will see next that there is no loss of generality in this assumption.

We now turn to the formal operational semantics. For every vector of generated names $\nu \boldsymbol{c}_i$, $i = 1, \ldots, k$ in Eq, we reserve the vectors of distinct constants:

$$\boldsymbol{C}^{old}_i, \boldsymbol{C}^{new}_i, \boldsymbol{C}^j_i, j = 0, 1, 2, \ldots$$

We assume that no confusion arises with constants appearing in Eq or with constants related to another thread. We will see that the standard semantics of name generation relies only on the constants $\boldsymbol{C}_i^j, j = 0, 1, 2, \ldots$ while the collapsed semantics presented in section 2.4 will rely on the constants $\boldsymbol{C}_i^{old}, \boldsymbol{C}_i^{new}$. We say that a thread P *relates to* the equation $A_i = \nu \boldsymbol{c}_i\, Q_i$ if P is either A_i or an instance of $?t^i_j.!s^i_j \ldots ?t^i_{l_i}.!s^i_{l_i}.A'_i$, $j \geq 1$.

Definition 2 (standard configuration). *A* standard configuration *is a pair* (R, T) *where:*

- $R \equiv (P_1, n_1) \mid \cdots \mid (P_k, n_k)$ *is the parallel composition of* k *pairs composed of a thread* P_i *relating to the* i^{th} *equation and a counter* n_i*.*
- T *is a finite set of messages representing the knowledge of the adversary,*

and such that the constants $\boldsymbol{C}_i^{old}$, $\boldsymbol{C}_i^{new}$, $\boldsymbol{C}_i^j$ *for* $i = 1, \ldots, k$ *and* $j > n_i$ do not *occur in* (R, T).

We assume that parallel composition is associative and commutative and feel free to write R as $(P_i, n_i) \mid R'$, where R' might be empty.

Definition 3 (standard reduction). *We define a reduction relation on standard configurations as follows:*

$$\begin{array}{lllr}
(\mathsf{unfold}) & ((A_i, n_i) \mid R, T) & \rightarrow ((\sigma_i^{n_i+1} Q_i, n_i + 1) \mid R, T), & (1) \\
(\mathsf{i/o}) & ((?t.!s.P, n_i) \mid R, T) & \rightarrow ((\theta P, n_i) \mid R, T \cup \{\theta s\}), & (2) \\
\textit{where:} & \multicolumn{3}{l}{(1)\ \sigma_i^{n_i+1} = [\boldsymbol{C}_i^{n_i+1}/\boldsymbol{c}_i],\ (2)\ \theta t \in S(A(T)) \textit{ and } \theta s \in \mathcal{M}\ .}
\end{array}$$

The first rule (unfold) expands the recursive definition, instantiates the generated names $\boldsymbol{c}_i$ with fresh constants, and increments the related counter n_i. The second rule (i/o) inputs from the adversary a message θt matching a filter and outputs a message θs. Note that if θs is *not* a message then the reduction cannot take place. We can now state the control reachability problem.

Definition 4. *We fix* $R_0 \equiv (A_1, 0) \mid \cdots \mid (A_k, 0)$ *as initial control,* $T_0 \neq \emptyset$ *as initial knowledge of the adversary, and write* $(R_0, T_0) \xrightarrow{*} \mathsf{err}$ *if* $(R_0, T_0) \xrightarrow{*} ((\mathsf{err}, n) \mid R', T')$ *for some* n, R', T'*. The control reachability problem amounts to determine whether* $(R_0, T) \xrightarrow{*} \mathsf{err}$*.*

2.3 Undecidable Fragments

It is convenient to introduce some *syntactic transformations* that do not affect the control reachability problem. Consider a thread $A = \nu \boldsymbol{c}\, ?t_1.!s_1 \ldots ?t_n.!s_n.A'_i$ and suppose s_i is the first output that depends on filter variables of $t_1, \ldots, t_{i-1}$, say $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}$. If $\boldsymbol{d} \equiv d_1, \ldots, d_j$ and $\boldsymbol{x} \equiv x_1, \ldots, x_j$ then $\langle \boldsymbol{d} \cdot \boldsymbol{x} \rangle$ stands for $\langle d_1 x_1, \ldots, d_j x_j \rangle \equiv \langle E(x_1, d_1), \ldots, E(x_j, d_j) \rangle$. With this convention, we can rewrite the thread as

$$\begin{array}{l}
A = \nu \boldsymbol{c}\, \nu \boldsymbol{d}_1, \ldots, \boldsymbol{d}_{i-1}\, ?t_1.!\langle s_1, \langle \boldsymbol{d}_1 \cdot \boldsymbol{x}_1 \rangle \rangle . \ldots \\
\quad ?t_{i-1}.!\langle s_{i-1}, \langle \boldsymbol{d}_{i-1} \cdot \boldsymbol{x}_{i-1} \rangle \rangle . \\
\quad ?\langle t_i, \langle \boldsymbol{d}_1 \cdot \boldsymbol{y}_1 \rangle, \ldots, \langle \boldsymbol{d}_{i-1} \cdot \boldsymbol{y}_{i-1} \rangle \rangle ![\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{i-1}/\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}] s_i . \ldots \\
\quad ?t_n.!s_n.A'_i
\end{array}$$

where $\boldsymbol{d}_j, \boldsymbol{y}_j$ are fresh. In other terms, we store under the fresh keys $\boldsymbol{d}_j$ the parameters $\boldsymbol{x}_j$ and retrieve them again in the filter just preceding the output s_i. By iterating this transformation, we obtain an equivalent thread satisfying the condition $Var(s_j) \subseteq FVar(t_j)$.

Repeated outputs $?t.!s_1.!s_2$ can be encoded as $?t.!s_1.?x.!s_2$ with x fresh. Repeated inputs $?t_1.?t_2.!s$ can be encoded as $?t_1.!t_0.?t_2.!s$ where t_0 is a term in the initial knowledge. We will also write $?t_1.!s_1 \ldots ?t_n.!s_n.0$ to indicate that the thread stops at point 0. This can be encoded by inserting at the place of 0 a filter that can never be passed. Finally, we may generate names during a session as in $\ldots ?t.!s.\nu c\, ?t'.!s'. \ldots$ which is equivalent to generate them at the very beginning.

Next we introduce four syntactic conditions on the system Eq. We anticipate that the syntactic transformations we have presented above can be performed while satisfying these conditions.

Definition 5 (variable dependency). *We say that the variables x and y are* dependent *in a term t if for some C there exists a subterm $E(t', C)$ of t with two distinct occurrences of x and y (x depends on itself if it occurs twice in t'). Otherwise, we say that x and y are independent in t.*

Definition 6 (syntactic conditions). *The system Eq satisfies the:*

(1) LINEARITY *condition if the filters t^i_j are linear,* i.e., *each variable in $FVar(t^i_j)$ occurs exactly once in t^i_j.*

(2) LOCALITY *condition if the filters t^i_j do* not *depend on previous filter variables,* i.e., *$Var(t^i_j) = FVar(t^i_j)$.*

(3) INDEPENDENCE *condition if for every i/o action $?t.!s$, assuming $\eta : KVar(s) \to \mathcal{N}$ is any assignment and $\langle s_1, \ldots, s_m \rangle \equiv \eta s$ the following holds for $l = 1, \ldots, m$: either (i) $\sharp Var(s_l) \leq 1$, or (ii) s_l is a linear term and the variables $Var(s_l)$ are independent in ηt.*

(4) DATA OR KEY *condition if every generated name occurs in the related thread either in data or key position (but not both, cf. definition 1), and moreover, if at least one generated name occurs in data position then every variable occurring in a filter can only occur in the related thread in data position.*[2]

Conditions (1–3) are partially motivated by the following undecidability result whose proof is based on a rather direct encoding of 2-counter machines. Conditions (1–3) will also play a role in the complexity and precision of the set-based analysis. Moreover, conditions (1–2) together with condition (4) will be used in section 2.4 to characterize the precision of the collapsed semantics.

[2] The restriction that filter variables occur only in data position is needed to avoid that a principal uses generated nonces as keys. The condition could be omitted if we included in the model a *typing* mechanism to distinguish nonces from keys.

Example 1. An example of cryptographic protocol satisfying all the four syntactic conditions in definition 6 is the Andrew Secure RPC Protocol from [CJ97]

$$\begin{array}{l}
(1)\ A \rightarrow B : A, E(Na, Kab)\\
(2)\ B \rightarrow A : E(\langle Na+1, Nb\rangle, Kab)\\
(3)\ A \rightarrow B : E(Nb+1, Kab)\\
(4)\ B \rightarrow A : E(\langle K'ab, N'b\rangle, Kab)
\end{array}$$

We do not have addition in our syntax, but we take $x+1$ as an abbreviation for $\langle x, 1\rangle$ where 1 is a constant symbol. Thus, we can model the protocol with the following two equations (note that the only variables here are x, y, z and w):

$$\begin{array}{lll}
A_1 = \nu n_a & & A_2 = \nu n_b, k'_{ab}, n'_b\\
(1) & !\langle A, E(n_a, K_{ab})\rangle. & (1) \quad ?\langle A, E(x, K_{ab})\rangle.\\
(2) & ?E(\langle n_a+1, y\rangle, K_{ab}). & (2) \quad !E(\langle x+1, n_b\rangle, K_{ab})).\\
(3) & !E(y+1, K_{ab}). & (3) \quad ?E(n_b+1, K_{ab}).\\
(4) & ?E(\langle z, w\rangle, K_{ab}).A_1 & (4) \quad !E(\langle k'_{ab}, n'_b\rangle, K_{ab}).A_2
\end{array}$$

Other examples include a series of Woo and Lam Π protocols in [CJ97, section 6.3.10].

Theorem 1 (undecidability). *There are encodings of 2-counter machines showing that violation of one of the conditions* LINEARITY*,* LOCALITY*,* INDEPENDENCE *is sufficient for the undecidability of the control reachability problem.*

Remark 2. If we assume the DATA OR KEY condition then theorem 1 still holds. Moreover, if we violate the INDEPENDENCE condition then undecidability does not rely on name generation.

2.4 Collapsed Semantics

We introduce the notion of collapsed configuration and reduction mimicking definitions 2 and 3.

Definition 7 (collapsed configuration). *A* collapsed configuration *is a pair* (R, T) *where:*

- *R is the parallel composition of k threads $P_1 \mid \cdots \mid P_k$ with P_i relating to the i^{th} equation,*
- *T is a finite set of messages representing the knowledge of the adversary,*

and such that the constants $\boldsymbol{C}_i^j$ for $i = 1, \ldots, k$ do not *occur in (R, T).*

Definition 8 (collapsed reduction). *We define a reduction relation on collapsed configurations as follows:*

$$\begin{array}{lll}
(\mathsf{unfold}) & (A_i \mid R, T) \ \rightarrow \ (\sigma_i^{new} Q_i \mid \sigma_i^{old} R \ , \sigma_i^{old} T), & \\
& \textit{if } \sigma_i^{new} = [\boldsymbol{C}_i^{new}/\boldsymbol{c}_i] \textit{ and } \sigma_i^{old} = [\boldsymbol{C}_i^{old}/\boldsymbol{C}_i^{new}] & \\
(\mathsf{i/o}) & (?t.!s.P \mid R, T) \ \rightarrow \ (\theta P \mid R, T \cup \{\theta s\}), & \textit{if } \theta t \in S(A(T)) \textit{ and } \theta s \in \mathcal{M} \ .
\end{array}$$

The notion of control reachability for collapsed configurations is an immediate adaptation of definition 4. Without name generation the collapsed semantics coincides with the standard semantics and therefore it follows from the remark 2 that the control reachability problem for the collapsed semantics is also undecidable in general.

We want to relate the control reachability problem for standard and collapsed configurations. Given any standard configuration (R, T) where $R \equiv (P_1, n_1) \mid \cdots \mid (P_k, n_k)$ we define a substitution τ_R as follows:

$$\tau_R(\boldsymbol{C}_i^j) = \begin{cases} \boldsymbol{C}_i^{new} \text{ if } j = n_i \\ \boldsymbol{C}_i^{old} \text{ if } j < n_i \ . \end{cases}$$

We extend the definition of τ_R to standard configurations as follows:

$$\tau_R(R, T) = (\tau_R P_1 \mid \cdots \mid \tau_R P_k, \tau_R T) \ .$$

We can prove the following proposition by case analysis on the reduction rules.

Proposition 1 (simulation). *Let (R, T) be a standard configuration.*

(1) *If $(R, T) \to (R_1, T_1)$ then $\tau_R(R, T) \to \tau_{R_1}(R_1, T_1)$.*

(2) *If $((A_1, 0) \mid \cdots \mid (A_k, 0), T_0) \xrightarrow{*} \mathsf{err}$ then $(A_1 \mid \cdots \mid A_k, T_0) \xrightarrow{*} \mathsf{err}$.*

Next, we analyse the impact of the syntactic conditions on the precision of the collapsed semantics.

Proposition 2. *There are examples showing that the violation of one of the conditions* LINEARITY, LOCALITY, DATA OR KEY *is sufficient to compromise the precision of the collapsed semantics (even when condition* INDEPENDENCE *is satisfied).*

On the other hand, if the three conditions hold then the control reachability problem in the collapsed semantics is equivalent to the control reachability problem in the standard one.

Theorem 2 (precision of collapsed semantics). *Suppose the system Eq satisfies conditions* LINEARITY, LOCALITY, *and* DATA OR KEY. *Then $((A_1, 0) \mid \cdots \mid (A_k, 0), T_0) \xrightarrow{*} \mathsf{err}$ iff $(A_1 \mid \cdots \mid A_k, T_0) \xrightarrow{*} \mathsf{err}$.*

Another way to obtain the precision of the collapsed semantics, which was suggested to us by Y. Lakhnech, is to restrict our attention to tail-recursive processes that *publish* at the end of the session the names generated at its beginning. We say that the system Eq satisfies the condition PUBLISH if in the system (1) the Q_i's have the shape $?t_1^i.!s_1^i \ldots ?x.!\boldsymbol{c}_i.A_i'$.

Proposition 3. *Suppose the system Eq satisfies conditions* LINEARITY, LOCALITY, *and* PUBLISH. *Then the collapsed semantics is precise in the sense of theorem 2.*

The condition PUBLISH allows to get rid of the restrictive condition DATA OR KEY. On the other hand, this condition puts the burden on the protocol which has to resist attacks coming from the publication of 'old' names.

3 Set Based Analysis

In this section we perform an analysis based on *set constraints* of the control reachability problem in the collapsed semantics. In section 3.1 we introduce a particular family of set constraints tailored to our needs, in section 3.2 we show how to generate them, and in section 3.3 we explain how to solve them.

3.1 A Family of Set Constraints

We will use a class of set constraints very close to *definite* set constraints with membership expressions [HJ90, CP97, TDT00], but there are few differences. First, we do not allow any expressions except variables on the right-hand side of inclusion, which is quite usual in set based program analysis [HJ94]; therefore, we are not interested in testing satisfiability but in computing the least solution (such constraints are always satisfiable—a trivial solution is a valuation assigning the set of all terms to each variable). Second, we use conditional inclusions of the form $\mathsf{if}\ \mathsf{ne}(S)\ \mathsf{then}\ S' \subseteq X$. This is not an essential extension since the emptiness test is an inherent part of every set constraint solving algorithm. Moreover, in a setting with more complicated expressions on the right-hand side of inclusion, such conditional inclusion is equivalent to $f(S, S') \subseteq f(S, X)$. Third, we use membership expressions as in [TDT00]. A not surprising, but probably new observation here is that if the formulas in the set comprehension part of these expressions are all linear then set constraints can be solved in polynomial time (see theorem 3(2)). Finally, we use a novel operation of renaming needed in the representation of the unfolding rule of the collapsed semantics (definition 8). To our knowledge, this operation was not present in any previous work on set constraints.

Syntax of Set Constraints We assume that a finite signature $\Sigma = \{f, \ldots\}$ of function symbols is given. Every function symbol has a fixed arity; symbols of arity 0 are also called constants and denoted with $c, \ldots$ We will use a set of *individual variables* $V = \{x, y, \ldots\}$ ranging over terms from T_Σ and a set of *set variables* $\Xi = \{X, Y, \ldots\}$ ranging over sets of terms 2^{T_Σ}.

Set expressions are given by the grammar:

$$S \ ::= \ X \mid f(S_1, \ldots, S_n) \mid \{x \mid t \in S\} \mid \sigma S,$$

where X ranges over Ξ and f over n-ary function symbols from Σ. The meta-variable t ranges over *linear* terms in $T_\Sigma(V)$ and may contain individual variables possibly different from x. The hypothesis that terms are linear can be removed at the price of an increase in the complexity of the solving algorithm (see [CP97, TDT00]).

The renaming operator σ is a substitution of the form $[\boldsymbol{c'}/\boldsymbol{c}\ ]$ that replaces all occurrences of constant symbols from $\boldsymbol{c}$ with a respective symbol in $\boldsymbol{c'}$. To ensure termination of the solving algorithm we assume that all renamings here are idempotent ($\sigma\sigma = \sigma$) and commutative ($\sigma\sigma' = \sigma'\sigma$) and come from a finite set *Sub*, so that a composition of renamings can be canonically represented by a subset of *Sub* taking conventionally the identity if the subset is empty. The assumption that the renamings operate on constant symbols is not essential—an extension to other function symbols is straightforward.

Set constraints are:

$$\Phi \ ::= \ S \subseteq X \mid \mathsf{if} \ \bigwedge_i \mathsf{ne}(S_i) \ \mathsf{then} \ S \subseteq X \mid \Phi_1 \wedge \Phi_2 \ .$$

As is usual, we identify a conjunction of constraints with the set of all conjuncts. We will require that for every inclusion $S \subseteq X$ in all subexpressions $\{x \mid t \in S'\}$ of S, the variable x occurs in the term t (this requirement is not essential, but it allows to avoid special rules in table 2 for handling the set of all terms). We will not require this in subexpressions of $\mathsf{ne}(S)$.

Semantics of Set Constraints A *valuation* $\alpha : \Xi \to 2^{T_\Sigma}$ is a mapping assigning sets of ground terms to set variables. The semantics of set expressions relative to a valuation α is defined recursively as follows.

$$\begin{aligned}
[\![X]\!]_\alpha &= \alpha(X) \\
[\![f(S_1,\dots,S_n)]\!]_\alpha &= \{f(t_1,\dots,t_n) \mid t_1 \in [\![S_1]\!]_\alpha,\dots,t_n \in [\![S_n]\!]_\alpha\} \\
[\![\{x \mid t \in S\}]\!]_\alpha &= \{t' \in T_\Sigma \mid \{\boldsymbol{y}\} = \mathit{Var}(t)\backslash\{x\} \text{ and } \exists\, \boldsymbol{s}\ \ [t'/x, \boldsymbol{s}/\boldsymbol{y}]t \in [\![S]\!]_\alpha\} \\
[\![\sigma S]\!]_\alpha &= \{\sigma t \mid t \in [\![S]\!]_\alpha\} \ .
\end{aligned}$$

We note that if x does not occur in t and there is an instance of t in $[\![S]\!]_\alpha$ then $[\![\{x \mid t \in S\}]\!]_\alpha$ is the set of all terms T_Σ. Similarly, if x does not occur in t and there is no instance of t in $[\![S]\!]_\alpha$ then $[\![\{x \mid t \in S\}]\!]_\alpha$ is the empty set.

A valuation α is a *solution* of a constraint Φ if for all conjuncts $S \subseteq X$ in Φ we have $[\![S]\!]_\alpha \subseteq \alpha(X)$, and for all conjuncts $\mathsf{if} \ \wedge_i \mathsf{ne}(\boldsymbol{S}_i) \ \mathsf{then} \ S' \subseteq X$ we have $[\![S']\!]_\alpha \subseteq \alpha(X)$ whenever $[\![S_i]\!]_\alpha \neq \emptyset$ for all i.

Obviously every constraint Φ is satisfiable—a trivial solution is a valuation assigning T_Σ to each variable (under this valuation each inclusion just expresses that a given set is a subset of the universe). In the following we want to find the least solution of a given constraint Φ. This *least solution* may be defined inductively by:

$$\begin{aligned}
\alpha_0(X) &= \emptyset, \\
\alpha_{i+1}(X) &= \bigcup \{[\![S]\!]_{\alpha_i} \mid S \subseteq X \in \Phi \text{ or } (\mathsf{if}\ \mathsf{ne}(\wedge_j S'_j)\ \mathsf{then}\ S \subseteq X \in \Phi, [\![S'_j]\!]_{\alpha_i} \neq \emptyset)\}, \\
\alpha(X) &= \textstyle\bigcup_{i \in \mathbb{N}} \alpha_i(X) \ .
\end{aligned}$$

3.2 Constraints Generation

Here we show how to generate set constraints from a system *Eq* satisfying the conditions LINEARITY, LOCALITY, and INDEPENDENCE. We prove that the generated constraints give a conservative approximation of the protocols. [3]

[3] The method can be generalized to protocols not satisfying the conditions above. To this end, it seems natural to rely on *non-linear* constraints in order to limit the loss of precision. In this case the constraints can be solved in DEXPTIME.

Under the syntactic conditions above a *control state* is a process R that can be decomposed in $P_1 \mid \cdots \mid P_k$ so that P_i is either A_i or err or $\sigma_i^{new}(?t^i_j.!s^i_j \ldots ?t^i_{l_i}.!s^i_{l_i}).A'_i$. For a system of k threads each comprising n i/o alternations there are at most $(n+2)^k$ control states and for a reachable collapsed configuration (R, T), R is always a control state.

For every control state R we introduce a set variable T_R. Intuitively it will represent (an approximation of) the knowledge of the adversary at this control point, so for any reachable configuration (R, T) the variable T_R will contain the set $S(A(T))$. Moreover, we introduce a set variable T_{err} that will be empty if an erroneous configuration is not reachable.

Table 1 gives the rules to generate constraints. The rules are self-explanatory: (init) is for the initial configuration, (err) for determining T_{err}, (A_{1-3}) and (S_{1-2}) for analysis and synthesis, respectively. The rule (unfold) is for the unfolding step. Here we note that every renaming occurring in the (unfold) constraint is of the form $[\boldsymbol{C}_i^{old}/\boldsymbol{C}_i^{new}]$, so for two different renamings their domains are disjoint, and every domain is disjoint from any range. This gives idempotency and commutativity as required. Finally, the ($\mathsf{i/o}_{1-2}$) rules cover the (i/o) reduction. In these rules, we get rid of the key variables in the output by considering all their possible instances (note that in the collapsed semantics the set $\mathcal{N}$ is finite). This is conceptually simple but may lead to inefficiency. In practice, one can introduce a limited form of intersection and write set expressions such as $\{x_i \mid t \in X\} \cap \mathcal{N}$.

Table 1. Constraints generated

(init)	$t \subseteq T_{R_0}$	(1)
(err)	$T_{\mathsf{err} \mid R} \subseteq T_{\mathsf{err}}$	(2)
(A_1)	$\{x \mid \langle x, y\rangle \in T_R\} \subseteq T_R$	(3)
(A_2)	$\{y \mid \langle x, y\rangle \in T_R\} \subseteq T_R$	(3)
(A_3)	$\mathsf{if\ ne}(\{y \mid C \in T_R\})\ \mathsf{then}\ \{x \mid E(x, C) \in T_R\} \subseteq T_R$	(3)
(S_1)	$\langle T_R, T_R\rangle \subseteq T_R$	(3)
(S_2)	$\mathsf{if\ ne}(\{y \mid C \in T_R\})\ \mathsf{then}\ E(T_R, C) \subseteq T_R$	(3)
(unfold)	$\sigma_i^{old} T_{A_i \mid R} \subseteq T_{\sigma_i^{new} Q_i \mid R}$	(4)
($\mathsf{i/o}_1$)	$\mathsf{if\ ne}(\{y \mid \eta t \in T_R\})\ \mathsf{then}\ T_R \subseteq T_{R'}$	(5)
($\mathsf{i/o}_2$)	$\mathsf{if\ ne}(\{y \mid \eta t \in T_R\})\ \mathsf{then}\ [\boldsymbol{S}/\boldsymbol{x}](\eta s) \subseteq T_{R'}$	(5)

(1) (R_0, T_0) initial configuration, $t \in T_0$.
(2) $\mathsf{err} \mid R$ control state.
(3) For all R control state and $C \in \mathcal{N}$.
(4) $A_i \mid R$ control state.
(5) $R \equiv ?t.!s.R_1 \mid R_2$ control state, $R' \equiv R_1 \mid R_2$,
$\{x_1, \ldots, x_n\} = \mathit{Var}(s) \backslash \mathit{KVar}(s)$,
$\eta : \mathit{KVar}(s) \to \mathcal{N}$, $S_i \equiv \{x_i \mid \eta t \in T_R\}$.

The set constraints provide a conservative approximation of the collapsed semantics. The proof proceeds by induction on the length of the reduction $(R_0, T_0) \xrightarrow{*} (R, T)$. We will see in example 2 that due to the constraints generated by rule $(\mathsf{i}/\mathsf{o}_2)$ the analysis is not exact.

Proposition 4. *Let α be a solution of the constraints generated according to table 1. Then:*

(1) *If $(R_0, T_0) \xrightarrow{*} (R, T)$ then $S(A(T)) \subseteq \alpha(T_R)$.*

(2) *If $(R_0, T_0) \xrightarrow{*} \mathsf{err}$ then $\alpha(T_{\mathsf{err}}) \neq \emptyset$.*

3.3 Constraints Solving

We say that a constraint Φ is in a *shallow* form if in every expression of the form $f(S_1, \ldots, S_n)$, $\{x \mid t \in S\}$ or σS occurring in Φ the expressions $S, S_1, \ldots, S_n$ are set variables. Any set constraint can be transformed into an equivalent one in shallow form by replacing every occurrence of a nested subexpression S with a fresh set variable X_S and adding a conjunct $S \subseteq X_S$. For the rest of this section we assume that all constraints are in shallow form.

Given a constraint Φ in shallow form over a set of variables Ξ and a set of renamings *Sub* we construct a constraint Φ' over a set of variables $\Xi' = \{(Y, u) \mid Y \in \Xi, u \subseteq \mathit{Sub}\}$ and an *empty* set of renamings. Thus if in Φ we have n variables and k renaming we have $n \cdot 2^k$ variables in Φ'. If $X = (Y, u) \in \Xi'$ and $\sigma \in \mathit{Sub}$ then X^σ stands for $(Y, u \cup \{\sigma\})$. Then Φ' is obtained from Φ by first replacing all set variables Y by $(Y, \emptyset)$ and then all set expressions σX by X^σ. By this construction we get rid of the set expressions σS and we will see that the least solution of Φ' when restricted to the variables $(Y, \emptyset)$ gives the least solution of Φ.

We say that a constraint Φ is in a *solved* form if all its conjuncts are of the form $f(X_1, \ldots, X_n) \subseteq X$. A constraint in a solved form can be seen as a transition table of a tree automaton whose states are set variables. The least solution of such a constraint is then a valuation that assigns to a variable X the language recognized by this automaton with X as a final state.

Our constraints solving algorithm applies the rules in table 2 starting from Φ'. Each consequence is an inclusion between two terms from a set of bounded size; this is used to show the termination and to derive the upper bound. Then given the initial constraint Φ' we infer all consequences of Φ' under the rules and then simply remove all inclusions that are not in solved form.

Definition 9 (closed and solved form). *For a constraint Φ we denote by Φ^C the least set of constraints that contains Φ and is closed under all rules in table 2, and by Φ^S the restriction of Φ^C to the constraints in a solved form $f(X_1, \ldots, X_n) \subseteq X$.*

Theorem 3. *Let Φ be a linear constraint over Ξ, Sub and let Φ' be the associated constraint over $\Xi' = \Xi \times 2^{\mathit{Sub}}$.*

Table 2. Saturation rules for linear constraints

1. $\bigwedge_i \mathsf{ne}(X_i) \rightarrow \mathsf{ne}(f(X_1,\ldots,X_n))$
2. $\bigwedge_i \mathsf{ne}(\{x \mid t_i \in X_i\}),\ f(X_1,\ldots,X_n) \subseteq X \rightarrow \mathsf{ne}(\{x \mid f(t_1,\ldots,t_n) \in X\})$
   (provided $f(t_1,\ldots,t_n)$ occurs in Φ)
3. $\mathsf{ne}(S), S \subseteq X \rightarrow \mathsf{ne}(X)$
4. $\mathsf{ne}(X) \rightarrow \mathsf{ne}(\{x \mid y \in X\})$ (y can be x)
5. $S \subseteq X,\ X \subseteq Y \rightarrow S \subseteq Y$
6. $\{x \mid f(t_1,\ldots,t_i[x],\ldots,t_n) \in X\} \subseteq Y,\ f(X_1,\ldots,X_n) \subseteq X \rightarrow$
   $\mathsf{if}\ \bigwedge_{j\neq i} \mathsf{ne}(\{x \mid t_j \in X_j\})\ \mathsf{then}\ \{x \mid t_i[x] \in X_i\} \subseteq Y$
7. $\{x \mid x \in X\} \subseteq Y \rightarrow X \subseteq Y$
8. $\bigwedge_i \mathsf{ne}(S_i),\ \mathsf{if}\ \bigwedge_i \mathsf{ne}(S_i)\ \mathsf{then}\ S \subseteq X \rightarrow S \subseteq X$
9. $f(X_1,\ldots,X_n) \subseteq X \rightarrow \sigma(f)(X_1^\sigma,\ldots,X_n^\sigma) \subseteq X^\sigma$

(1) *The least solution of Φ'^S restricted to the variables $(Y,\emptyset)$ of Ξ' is the least solution of Φ.*

(2) *The least solution of Φ can be computed in time $\mathsf{poly}(n \cdot 2^k)$ where n is the size of Φ and k is the number of renaming operations in Φ.*

4 Precision of the Set-Based Analysis

The use of the collapsed semantics and the related set-based analysis can be applied to all protocols from [CJ97]. However, most of them violate at least one of the conditions above (example 1 being an exception) and this leads to a loss of precision either in the collapsed semantics or in set-based analysis. It turns out that in general the set based analysis is *not* precise even under the four syntactic conditions given in definition 6.

Example 2. Consider the initial knowledge $T_0 = \{ABD, ACD\}$ and the system

$$A_1 = ?Ax.!x.?By.?Cz.\mathsf{err}\ .$$

The constraints generated do not express that after the first filter is passed either BD or CD are known but not both. Consequently, the following two filters are passed and the erroneous state is reached.

An approach to the characterization of the set-based analysis is to look at *iterated* threads *without* name generation. [4] A thread is iterated if a new copy of the thread is spawned as soon as the first input output action is performed. A system Eq of k iterated threads is then formalized by k equations:[5]

$$A_i = ?t_1^i.!s_1^i.(A_i \mid (?t_2^i.!s_2^i \ldots ?t_{l_i}^i.!s_{l_i}^i.U.)) \tag{2}$$

[4] *With* name generation a simple variant of the example 2 shows that precision is lost.

[5] This formalization is equivalent to the standard notion of replication in π-calculus.

where U can be either the terminated state 0 or the erroneous state err. Since we do not have generated names, the transformation given in section 2.3 does not apply and we just require that $Var(s^i_j)$ is a subset of $\bigcup_{l\leq j} FVar(t^i_l)$ rather than of $FVar(t^i_j)$.

It turns out that these programs can be *flattened* so that each thread includes just *one alternation of input and output* and it is thus expressed by a tail recursive definition $A = ?t.!s.A$. Thus the following theorem 4 is also a result about the precision of the set based analysis for tail-recursive definitions without name generation and with one alternation of input and output.

Definition 10. *The system of iterated threads (2) satisfies, respectively, the* LINEARITY *and* INDEPENDENCE *conditions if for all equations $i = 1, \ldots, k$ the term $\langle t^i_1, \ldots, t^i_{l_i}\rangle$ is linear and the i/o action $?\langle t^i_1, \ldots, t^i_{l_i}\rangle.!\langle s^i_1, \ldots, s^i_{l_i}\rangle$ is independent in the sense of definition 6(3).*

Theorem 4. *Let n be the size of the system of iterated threads (2), c be the number of different keys, and k be the number of key variables. Suppose the system satisfies the* INDEPENDENCE *condition 10. Then the control reachability problem is* DEXPTIME*-hard and decidable in time $\mathsf{exp}(n \cdot c^k)$. Moreover, under the additional* LINEARITY *condition it is decidable in time $\mathsf{poly}(n \cdot c^k)$.*

We expect that the factor c^k can be reduced, but the exponential blowup in the non-linear case is unavoidable as we can reduce the satisfaction of unary definite set constraints (see [CPT00]) to control reachability for the class of iterated systems considered even when neither pairs nor key variables are used (see [AC02] for details).

The upper bound relies on the flattening transformation mentioned above. By this transformation every input-output action can be repeated in every reachable configuration. This property coupled with the INDEPENDENCE condition allows to match the constraint generated by the rule $(\mathsf{i/o}_2)$.

As a particular instance of this result, one can obtain yet another polynomial time decision procedure for ping-pong protocols which satisfy LINEARITY and do not contain variables in key position (see [DEK82] and [ALV01] for another decision procedure based on prefix-rewriting). Of course, the DEXPTIME lower bound implies that the class of decidable protocols considered in the theorem 4 above is strictly more expressive than the class of ping-pong protocols.

Acknowledgement

The first author is partly supported by ACI VERNAM and IST PROFUNDIS.

References

[ALV01] R. Amadio, D. Lugiez, and V. Vanackere. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science* (to appear). Also RR 4147, INRIA. 499, 513

[AC02] R. Amadio, W. Charatonik. On name generation and set-based analysis in the Dolev-Yao model. RR-INRIA 4379, January 2002. 499, 501, 513

[AM01] R. Amadio, C. Meyssonnier. On the decidability of fragments of the asynchronous π-calculus. Journal of Nordic Computing (to appear). Also RR-INRIA 4241.

[CP97] W. Charatonik and A. Podelski. Set constraints with intersection. In *Proc. 12th IEEE LICS*, 1997. 500, 508

[CPT00] W. Charatonik, A. Podelski, and J.-M. Talbot. Paths vs. trees in set-based program analysis. In *Proc. 27th Annual ACM POPL*, 2000. 513

[CCM01] H. Comon, V. Cortier, and J. Mitchell. Tree automata with one memory, set constraints, and ping-pong protocols. In *Proc. ICALP*, Springer Lecture Notes in Comp. Sci. 2076, 2001. 501

[CJ97] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. Available at `http://www-users.cs.york.ac.uk/~jac/papers/drareview.ps.gz`, 1997. 506, 512

[DEK82] D. Dolev, S. Even, and R. Karp. On the security of ping-pong protocols. *Information and Control*, 55:57–68, 1982. 499, 513

[DLMS99] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. Formal methods and security protocols, FLOC Workshop, Trento*, 1999. 499

[DY83] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Trans. on Information Theory*, 29(2):198–208, 1983. 499

[GK00] T. Genet and F. Klay. Rewriting for cryptographic protocol verification. In *Proc. CADE*, Springer Lecture Notes in Comp. Sci. 1831, 2000. 501

[Gou00] J. Goubault. A method for automatic cryptographic protocol verification. In *Proc. FMPPTA, Springer-Verlag*, 2000. 501

[HJ90] N. Heintze and J. Jaffar. A decision procedure for a class of set constraints (extended abstract). In *Proc. 5th IEEE LICS*, 1990. 508

[HJ94] N. Heintze and J. Jaffar. Set constraints and set-based analysis. In *Proc. Workshop on Principles and Practice of Constraint Programming*, Springer Lecture Notes in Comp. Sci. 874, 1994. 508

[Mon99] D. Monniaux. Abstracting cryptographic protocols with tree automata. In *Proc. Static Analysis Symposium*, Springer Lect. Notes in Comp. Sci., 1999. 501

[RT01] M. Rusinowitch and M. Turuani Protocol insecurity with finite number of sessions is NP-complete. RR INRIA 4134, March 2001. 499

[Sto99] S. Stoller. A bound on attacks on authentication protocols. TR 526, Indiana University, CS Dept., july 1999. 500

[TDT00] J.-M. Talbot, Ph. Devienne, and S. Tison. Generalized definite set constraints. *Constraints: An International Journal*, 5(1-2):161–202, January 2000. 500, 508

[Wei99] C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *Proc. CADE 99*. Springer Lect. Notes in Comp. Sci. (LNAI) 1632, 1999. 501