Finite Approximations of Terms up to Rewriting

Roberto Zunino and Pierpaolo Degano

Dipartimento di Informatica, Università di Pisa, Italy
{zunino,degano}@di.unipi.it

Abstract. A technique is presented for computing a finite over-approximation of a language of terms, modulo rewriting. The language is represented by an arbitrary term grammar. The approximation is a term grammar as well, and it already embodies rewriting, i.e. its language includes all the original terms *and* all their possible rewritings. We propose this technique to statically verify cryptographic protocols, expressed in process calculi involving non-free term algebras. As an example, we establish the forward secrecy of a Diffie-Hellman–based protocol.

1 Introduction

In the last years, various process calculi have been used to specify cryptographic protocols [15,3,5], making it possible to build automatic tools for reasoning on them. Among these, we mention dynamic analysers [16,13] and static analysers [7,6,8,5,11,1,18].

Each of the above calculi fixes once and for all a specific set of cryptographic primitives, and relies on having a *single* representation for each value. So, destructing composed terms is only possible through pattern matching, which is a feature of these calculi, and *not* of their term algebras which are free. Exploiting the freeness of the term algebra, the control flow analysis (CFA) of calculi for cryptographic protocols [17] succeeds in finitely representing (approximations of) the set of terms that can be dynamically bound to variables occurring in the protocol specification. This is done for the Spi–calculus [3], and related calculi by using *tree grammars*. Roughly, the computed grammar has one non-terminal for each "point" in the specification, and the productions follow the flow of data from each point towards another one. The number of non-terminals is determined by the protocol specification, only.

Unfortunately, not all cryptographic primitives admit a single representation, e.g. XOR. When using these primitives, the term algebra is no longer free, so the above calculi are not fully adequate. The applied pi calculus [2] instead has been designed to allow for non-free term algebras: its users can define their own primitives, and can specify an equivalence relation over terms. The resulting calculus may be used for the specification of a broader class of protocols, more directly than using, e.g. Spi. However, the presence of a non-free term algebra makes it difficult to define a static analysis of the applied pi calculus. For instance, a CFA for it has been defined in [19], but only for free algebras.

In this paper, we present a static analysis technique for approximating term languages modulo a given equivalence, and making it easier to reason about cryptographic protocols specified in the full applied pi calculus. The equivalence defining the term algebra can be specified through *any* rewriting system $\mathcal{R}$. Indeed, we do not put any restrictions on $\mathcal{R}$: for our purposes, it needs neither to be confluent nor terminating. Also here, we use grammars for the finite representation of languages. Unlike the standard CFAs [17], the set of non-terminals is *not* fixed: non-terminals are instead generated during the analysis on a need-by-need basis. Actually, the number of generated non-terminals is a user-tunable parameter of our analysis. Increasing this number can produce a better approximation, at the expense of a (polynomially) longer computation time.

More in detail, we assume to have a grammar $\mathcal{G}$ (and its derivation relation $\rightarrow_{\mathcal{G}}$). Here, we do not discuss how to derive $\mathcal{G}$ from a given specification. This can be done following the standard CFA approach, with minor enhancements. Our goal is over-approximating the reachability defined by $\rightarrow_{\mathcal{G}}^{*}\rightarrow_{\mathcal{R}}^{*}$, where $\rightarrow_{\mathcal{R}}$ denotes the rewriting relation. Indeed, the reachable terms form the languages of $\mathcal{G}$ up to rewritings in $\mathcal{R}$. It is convenient to approximate instead the larger relation $(\rightarrow_{\mathcal{G}}^{*}\rightarrow_{\mathcal{R}}^{*})^{*}$ which allows for interleaving derivations with rewritings. To do that, we look for a new grammar $\mathcal{G}'$ such that

$$\rightarrow_{\mathcal{G}'}^{*}= (\rightarrow_{\mathcal{G}'}^{*}\rightarrow_{\mathcal{R}}^{*})^{*} \supseteq (\rightarrow_{\mathcal{G}}^{*}\rightarrow_{\mathcal{R}}^{*})^{*} \supseteq\rightarrow_{\mathcal{G}}^{*}\rightarrow_{\mathcal{R}}^{*}$$

We call such a $\mathcal{G}'$ *fully-exposing* because its language is *already closed* under rewritings in $\mathcal{R}$. We define a (polynomial-time) algorithm to compute such a $\mathcal{G}'$. We also have a running prototype, that has been used to validate our ideas. In particular we established forward secrecy for a Diffie-Hellman–based protocol, that uses exponentials.

To the best of our knowledge, the only approach similar to ours is [4]. However, they only consider certain equational theories, e.g. without associativity, and define a semi-algorithm to obtain rewriting rules with "partial normal forms." They then use ProVerif to check processes equivalent, thus establishing security properties.

We plan to exploit the results presented here, and build a CFA for the full applied pi calculus, with no constraints on the term algebras. Leaving this for future research, in this work we instead focus on the foundational, calculus-neutral issues.

Background and notation are in Section 2. Section 3 establishes the semantic properties we exploit throughout the paper. An algorithm for left-linear rewritings is defined and discussed in Section 4, while in Section 5 we relax the left-linearity assumption, adapting our algorithm to any rewriting system. Finally, Section 6 reports on our experiments with our tool.

2 Preliminaries

Throughout the paper, we use the following denumerable sets: the set $\mathcal{L}$ of *labels* ($@l$, $@a$, $@b$, $\dots$), the set $\mathcal{X}$ of *variables* (X, Y,$\dots$), the set $\mathcal{F}$ of *function symbols* (f, g, $+$, 1, $\dots$).

Definition 1 *The set of* terms $\mathcal{T}$ *is inductively defined by*

$$\begin{array}{lll} T ::= & X & \textit{variable} \\ & @l & \textit{label} \\ & \mathsf{f}(\cdots, T, \cdots) & \textit{application} \end{array}$$

By definition, we have $\mathcal{L} \cup \mathcal{X} \subseteq \mathcal{T}$. Moreover, we assume that each function symbol f has a fixed associated arity. When the arity is zero (as for the function symbol 1), we simply write f instead of $\mathsf{f}()$.

The *size* of a term is the number of the applications occurring in it. The *depth* of a term is the maximum number of *nested* applications. For instance, g, $\mathsf{f}(X)$, and $\mathsf{f}(@l)$ have depth and size 1, while $\mathsf{f}(\mathsf{g}, \mathsf{h})$ has depth 2 and size 3, and X and $@l$ have depth and size zero. The variables occurring in a term T are denoted by $\mathsf{vars}(T)$.

Also, we adopt the following conventions:

- a *ground* term ($\in \mathcal{T}_{\mathsf{gr}}$) is a term with no occurring variables;
- a *simple* term ($\in \mathcal{T}_{\mathsf{sm}}$) is a term with no occurring labels;
- a *pure* term ($\in \mathcal{T}_{\mathsf{pr}}$) is a term which is both ground and simple;
- a *plain* term ($\in \mathcal{T}_{\mathsf{pl}}$) is a ground term of depth at most one.

Similarly, we shall use the sets $\mathcal{C}_{\mathsf{gr}}, \mathcal{C}_{\mathsf{sm}}, \mathcal{C}_{\mathsf{pr}}$ to denote the sets of ground, simple, and pure *contexts* $C[\bullet]$, respectively. That is, $C[\bullet] \in \mathcal{C}_{\mathsf{type}}$ iff $C[T] \in \mathcal{T}_{\mathsf{type}}$, where T is an arbitrary pure term, and type ranges over $\mathsf{gr}, \mathsf{sm}, \mathsf{pr}$. $\mathcal{C}$ is the set of all contexts.

Below, we give the definitions for rewriting systems and grammars.

Definition 2 *A* term rewriting system $\mathcal{R}$ *is a finite set of rewriting rules* $\mathcal{R}$ *having the form* $L \Rightarrow R$*, where* $L, R \in \mathcal{T}_{\mathsf{sm}}$ *and* $\mathsf{vars}(R) \subseteq \mathsf{vars}(L)$.

Example The following rules model the usual rules for pairs.

$$\mathcal{R} = \{\mathsf{fst}(\mathsf{cons}(X, Y)) \Rightarrow X, \ \ \mathsf{snd}(\mathsf{cons}(X, Y)) \Rightarrow Y\}$$

Definition 3 *Given a term rewriting system* $\mathcal{R}$*, we define the relation* $\rightarrow_{\mathcal{R}} \subseteq \mathcal{T}_{\mathsf{gr}} \times \mathcal{T}_{\mathsf{gr}}$ *to be the minimum relation closed under ground contexts such that for each* $(L \Rightarrow R) \in \mathcal{R}$*, and for each substitution* $\sigma : \mathcal{X} \rightarrow \mathcal{T}_{\mathsf{gr}}$*, we have* $\sigma L \rightarrow_{\mathcal{R}} \sigma R$.

Example Using the above $\mathcal{R}$, we have, for any $T_i \in \mathcal{T}_{\mathsf{gr}}$

$$\mathsf{fst}(\mathsf{fst}(\mathsf{cons}(\mathsf{cons}(T_1, T_2), T_3))) \rightarrow_{\mathcal{R}} \mathsf{fst}(\mathsf{cons}(T_1, T_2)) \rightarrow_{\mathcal{R}} T_1$$

Definition 4 *A* term grammar $\mathcal{G}$ *is a finite set of productions having the form* $@l : T$*, where* $T \in \mathcal{T}_{\mathsf{pl}}$*. This is a context–free grammar where the non–terminals are the labels. We write* $\mathcal{L}_{\mathcal{G}}$ *for the set of labels occurring in* $\mathcal{G}$*. We sometimes write several productions in the compact form* $@l : T_1, T_2, T_3$.

Note that we require $T \in \mathcal{T}_{\mathsf{pl}}$ instead of $T \in \mathcal{T}_{\mathsf{gr}}$. This is not really a restriction, since every term grammar having productions where $T \in \mathcal{T}_{\mathsf{gr}}$ can be transformed into its equivalent *Chomsky normal form*, which only uses plain terms, as the following example shows.

Example The grammar

$$@a : \mathsf{f}(@a), 1 \qquad @l : \mathsf{cons}(\mathsf{cons}(@a, @a), @a)$$

has the following Chomsky normal form, where $@l_1$ is a fresh label:

$$@a : \mathsf{f}(@a), 1 \qquad @l : \mathsf{cons}(@l_1, @a) \qquad @l_1 : \mathsf{cons}(@a, @a)$$

Definition 5 *Given $\mathcal{G}$ we define the relation $\rightarrow_{\mathcal{G}} \subseteq \mathcal{T}_{\mathsf{gr}} \times \mathcal{T}_{\mathsf{gr}}$ as the minimum relation closed under ground contexts such that, for each $@l : T \ \in \mathcal{G}$, we have $@l \rightarrow_{\mathcal{G}} T$.*

Definition 6 *Let $\rightarrow_{\mathcal{R},\mathcal{G}}$ be $\rightarrow_{\mathcal{R}} \cup \rightarrow_{\mathcal{G}}$. We write $[\![T]\!]_{\mathcal{G}}$ ($[\![T]\!]_{\mathcal{R},\mathcal{G}}$) for the set of* pure *terms reachable through $\rightarrow_{\mathcal{G}}$ ($\rightarrow_{\mathcal{R},\mathcal{G}}$) from the term $T \in \mathcal{T}_{\mathsf{gr}}$:*

$$[\![T]\!]_{\mathcal{G}} = \{T' \mid T \rightarrow^*_{\mathcal{G}} T' \in \mathcal{T}_{\mathsf{pr}}\} \qquad [\![T]\!]_{\mathcal{R},\mathcal{G}} = \{T' \mid T \rightarrow^*_{\mathcal{R},\mathcal{G}} T' \in \mathcal{T}_{\mathsf{pr}}\}$$

Of course, $[\![T]\!]_{\mathcal{G}}$ is the language generated by T, and trivially $[\![T]\!]_{\mathcal{G}} \subseteq [\![T]\!]_{\mathcal{R},\mathcal{G}}$.

It might be interesting to note that, in general, the relations $\rightarrow^*_{\mathcal{G}}$ and $\rightarrow^*_{\mathcal{R}}$ do not commute, even if the target term is restricted to be in $\mathcal{T}_{\mathsf{pr}}$. Formally:

$$\exists \mathcal{R}, \mathcal{G} \ . \ \rightarrow^*_{\mathcal{R}}\rightarrow^*_{\mathcal{G}} \cap (\mathcal{T}_{\mathsf{gr}} \times \mathcal{T}_{\mathsf{pr}}) \not\subseteq \rightarrow^*_{\mathcal{G}}\rightarrow^*_{\mathcal{R}}$$
$$\exists \mathcal{R}, \mathcal{G} \ . \ \rightarrow^*_{\mathcal{G}}\rightarrow^*_{\mathcal{R}} \cap (\mathcal{T}_{\mathsf{gr}} \times \mathcal{T}_{\mathsf{pr}}) \not\subseteq \rightarrow^*_{\mathcal{R}}\rightarrow^*_{\mathcal{G}}$$

A counterexample for the first statement is

$$\mathcal{R} = \{\mathsf{f}(X) \Rightarrow \mathsf{g}(X, X)\} \qquad \mathcal{G} = \{@l : \mathsf{a}, \mathsf{b}\}$$
$$T_b = \mathsf{f}(@l) \rightarrow_{\mathcal{R}} \mathsf{g}(@l, @l) \rightarrow_{\mathcal{G}} \mathsf{g}(\mathsf{a}, @l) \rightarrow_{\mathcal{G}} \mathsf{g}(\mathsf{a}, \mathsf{b}) = T_e$$

Indeed, we do not have $T_b \rightarrow^*_{\mathcal{G}}\rightarrow^*_{\mathcal{R}} T_e$. For the second statement, take

$$\mathcal{R} = \{\mathsf{f}(X) \Rightarrow \mathsf{a}\} \qquad \mathcal{G} = \{@l : f(@m) \ \ ; \ \ @m : \mathsf{b}\}$$
$$T_b = @l \rightarrow_{\mathcal{G}} \mathsf{f}(@m) \rightarrow_{\mathcal{R}} \mathsf{a} = T_e$$

and note that we do not have $T_b \rightarrow^*_{\mathcal{R}}\rightarrow^*_{\mathcal{G}} T_e$. Therefore, the language up to rewriting $[\![@a]\!]_{\mathcal{G}}/\mathcal{R}$ may be smaller than the set $[\![@a]\!]_{\mathcal{R},\mathcal{G}}$ we focus on. We will return to this point in Section 5.1.

The following definition characterizes those grammars the languages of which include also the terms up to rewriting.

Definition 7 *A grammar $\mathcal{G}$ is said to be* fully-exposing *(w.r.t $\mathcal{R}$) iff for every $@l \in \mathcal{L}_{\mathcal{G}}$, we have $[\![@l]\!]_{\mathcal{G}} = [\![@l]\!]_{\mathcal{R},\mathcal{G}}$.*

2.1 Matching

Beneficial to studying $\rightarrow_{\mathcal{R},\mathcal{G}}$ is understanding which terms T, with $@l \rightarrow^*_{\mathcal{G}} T$, can be rewritten by a rule $L \Rightarrow R$, i.e. which such terms T *match* L. To this aim, we define the result of a *match* of the label $@l$ with the pattern L as the set of the bindings $\sigma : \mathcal{X} \to \mathcal{T}_{\mathsf{gr}}$ for which $@l \to^*_{\mathcal{G}} \sigma L$. Note that the number of such bindings may be infinite, in general.

A very interesting subset of the bindings resulting from a match is the one formed by *label* bindings, i.e. those $\sigma : \mathcal{X} \to \mathcal{L}$. Indeed, there are only a finite number of label bindings, since the variables occurring in L are finite, and so are the labels occurring in $\mathcal{G}$. The set of the label bindings that match against L can be computed in a natural way under the following assumption.

Left Linearity We assume each rule $L \Rightarrow R \in \mathcal{R}$ be *left–linear*, i.e. each variable occurs at most once in L.

Intuitively, the above assumption means that each variable in L can be matched independently. To compute label bindings, just run a top–down parser, expanding $@l$ by applying all the productions in a non deterministic fashion, and discarding the mismatching branches. Note that we only need to search as deep as the size of the pattern L: for the most common rewriting rule sets this is 2 or 3. Therefore, even though we adopt an exponential-time parser for this, the impact on the performance of our tool will still be acceptable. The number of label bindings is at most $(\#\mathcal{L}_{\mathcal{G}})^k$ where k is the number of variables in L. If we consider a given $\mathcal{R}$, then $k = \mathcal{O}(1)$, hence the number of bindings is polynomial on the size of $\mathcal{G}$.

Example Given the grammar $\mathcal{G}$

$$\begin{array}{ll} @a : 1, 2, 3 & @b : +(@a, @a) \\ @l : \mathsf{cons}(@a, @l), \mathsf{fst}(@m) & @m : \mathsf{cons}(@l, @b), @l \end{array}$$

the match of $@l$ with the pattern $L = \mathsf{fst}(\mathsf{cons}(X, Y))$ produces the bindings $\sigma_1 = \{X \mapsto @l, Y \mapsto @b\}, \sigma_2 = \{X \mapsto @a, Y \mapsto @l\}$.

Assuming left linearity is rather convenient, but it prevents us from using many well-known rewriting rules, such as $\mathsf{xor}(X, X) \Rightarrow 0$. We will return to this hypothesis, and relax it, in Section 5.

3 Semantics

A nice direct consequence of having $\mathcal{G}$ in Chomsky normal form is that each *sub*-term occurring in $[\![@l]\!]_{\mathcal{G}}$ also belongs to the language $[\![@m]\!]_{\mathcal{G}}$ for some $@m \in \mathcal{L}_{\mathcal{G}}$, as established by the lemma below.

Lemma 1 (Label-in-the-middle). *If $@l \to^*_{\mathcal{G}} C[T]$ for some $@l \in \mathcal{L}_{\mathcal{G}}$, $C[\bullet] \in \mathcal{C}_{\mathsf{gr}}$ and $T \in \mathcal{T}_{\mathsf{gr}}$, then for some label $@m \in \mathcal{L}_{\mathcal{G}}$ we have $@l \to^*_{\mathcal{G}} C[@m]$ and $@m \to^*_{\mathcal{G}} T$.*

Intuitively, the "label-in-the-middle" lemma implies that $\mathcal{G}$ provides us with a convenient abstraction (namely, the label $@m$) for many terms that are affected by rewriting. Indeed, whenever a rewriting $\sigma L \to_{\mathcal{R}} \sigma R$ occurs and $\sigma L \in [\![@l]\!]_{\mathcal{G}}$, we know that there is some label $@m_i$ in $\mathcal{G}$ that generates σX_i, for each variable X_i of L. This suggests us a way for proving some properties involving the whole, potentially infinite, set of bindings $\sigma : \mathcal{X} \to \mathcal{T}_{\mathsf{gr}}$. Proving a property Φ for all σ might reduce to checking Φ on the set $\mathcal{B}$ of label bindings $\sigma : \mathcal{X} \to \mathcal{L}$. This turns out to be convenient when only variables in $\mathcal{R}$ and labels in $\mathcal{G}$ are used, because $\mathcal{B}$ is finite.

Quite interestingly, this is the case when proving that a grammar is fully-exposing. The next lemma gives a convenient criterion for this, thus paving the way towards an algorithmic treatment of grammars and rewritings.

Lemma 2 (Label Bindings Criterion). *If $\mathcal{G}$ satisfies*

$$\forall (L \Rightarrow R) \in \mathcal{R}.\ \forall \sigma : \mathcal{X} \to \mathcal{L}\ .\ \forall @l \in \mathcal{L}_{\mathcal{G}}.\ @l \to^*_{\mathcal{G}} \sigma L \implies @l \to^*_{\mathcal{G}} \sigma R$$

then $\mathcal{G}$ is fully-exposing.

3.1 Grammar Transformations

We now define a way to modify $\mathcal{G}$ preserving its semantics, in the sense made precise by the following preorder.

Definition 8 *Given $\mathcal{R}$ and $\mathcal{L}' \subseteq \mathcal{L}$, we define a preorder $\sqsubseteq^{\mathcal{L}'}_{\mathcal{R}}$ over the set of grammars as follows:*

$$\mathcal{G} \sqsubseteq^{\mathcal{L}'}_{\mathcal{R}} \mathcal{G}' \iff \forall @l \in \mathcal{L}'\ .\ [\![@l]\!]_{\mathcal{R},\mathcal{G}} \subseteq [\![@l]\!]_{\mathcal{R},\mathcal{G}'}$$

We also write $\mathcal{G} \equiv^{\mathcal{L}'}_{\mathcal{R}} \mathcal{G}'$ when $\mathcal{G} \sqsubseteq^{\mathcal{L}'}_{\mathcal{R}} \mathcal{G}' \wedge \mathcal{G}' \sqsubseteq^{\mathcal{L}'}_{\mathcal{R}} \mathcal{G}$.

Using standard terminology of grammars, labels in $\mathcal{L}'$ are the starting non-terminals for a set of languages, while all other labels only serve as auxiliary non-terminals. The preorder considers the languages of non-terminals in $\mathcal{L}'$, and disregards the other labels.

To modify $\mathcal{G}$ into a grammar $\mathcal{G}'$ such that $\mathcal{G} \equiv^{\mathcal{L}'}_{\mathcal{R}} \mathcal{G}'$, each label in $\mathcal{L}_{\mathcal{G}} \cap \mathcal{L}'$ needs to be in $\mathcal{G}'$, as well. Also, we may rename or remove labels in $\mathcal{L}_{\mathcal{G}} \setminus \mathcal{L}'$, as long as the languages generated by $\mathcal{L}_{\mathcal{G}} \cap \mathcal{L}'$ in $\mathcal{G}'$ are larger than (or equal to) those generated in $\mathcal{G}$. Below, we define three transformations that preserve our preorder.

$$\textsf{Augment} \qquad \frac{}{\mathcal{G} \sqsubseteq^{\mathcal{L}'}_{\mathcal{R}} \mathcal{G} \cup (@l : T)}$$

$$\textsf{Transitivity} \qquad \frac{(@l_1 : @l_2) \in \mathcal{G} \quad (@l_2 : T) \in \mathcal{G}}{\mathcal{G} \equiv^{\mathcal{L}'}_{\mathcal{R}} \mathcal{G} \cup (@l_1 : T)}$$

$$\textsf{Join} \qquad \frac{\forall T\ .\ (@l_1 : T) \in \mathcal{G} \iff (@l_2 : T) \in \mathcal{G}}{\mathcal{G} \equiv^{\mathcal{L}'}_{\mathcal{R}} \mathcal{G}\{@l_2/@l_1\}} \ @l_1 \notin \mathcal{L}'$$

Rule Augment simply adds a production to $\mathcal{G}$, clearly preserving our preorder. Rule Transitivity short-cuts two productions, yielding a completely equivalent grammar. Note that both rules Augment and Transitivity increase the size of $\mathcal{G}$.

Rule Join is more peculiar. It replaces one label $@l_1$ with another label $@l_2$, under the assumption they have the same productions (and therefore generate the same language). This rule may decrease the size of $\mathcal{G}$, both because all the productions having the form $@l_1 : T$ are removed, and because pairs of productions such as $@m : \mathsf{f}(@l_1), \mathsf{f}(@l_2)$ collapse into the single production $@m : \mathsf{f}(@l_2)$. The languages generated by the labels are unaffected, except for $[\![@l_1]\!]_{\mathcal{R},\mathcal{G}}$ that is not of interest, as $@l_1 \notin \mathcal{L}'$.

4 Algorithm

Below, we give an algorithm that approximates a pair $\mathcal{G}, \mathcal{R}$ with a fully-exposing grammar $\mathcal{G}'$. In other words, it computes a $\mathcal{G}'$ such that $\mathcal{G} \sqsubseteq_{\mathcal{R}}^{\mathcal{L}_\mathcal{G}} \mathcal{G}'$.

```
MAIN ALGORITHM Inputs: $\mathcal{G}, \mathcal{R}$, maxLabels, maxJoin

1. Start with $\mathcal{G} = \{@l_i : T_i\}$, where $0 \leq i < s$ and $T_i \in \mathcal{T}_{\mathsf{pl}}$.
2. nextLabel $\leftarrow s$, numJoin $\leftarrow 0$
3. Close $\mathcal{G}$ under Transitivity.
4. If numJoin < maxJoin:
      Close $\mathcal{G}$ under Join. Increment numJoin for each joined label.
5. For each label $@l \in \mathcal{L}_\mathcal{G}$:
      For each rule $(L \Rightarrow R) \in \mathcal{R}$:
         For each binding $\sigma \in \mathcal{X} \to \mathcal{L}$ such that $@l \to_\mathcal{G}^* \sigma L$:
            augment$(@l, \sigma R)$.
6. Repeat from step 3, until $\mathcal{G}$ reaches a fixed point.

SUBROUTINE augment$(@l, T)$

1. If $T \in \mathcal{T}_{\mathsf{pl}}$, then $\mathcal{G} \leftarrow \mathcal{G} \cup (@l : T)$ and return.
2. We have $T = \mathsf{f}(T_1, \dots, T_n)$.
   For each $i \in \{1 \dots n\}$:
      $@res_i \leftarrow$ place$(T_i)$.
3. $\mathcal{G} \leftarrow \mathcal{G} \cup (@l : \mathsf{f}(@res_1, \dots, @res_n))$.

SUBROUTINE place$(T)$ Returns a label.

1. If $T \in \mathcal{L}$, then return $T$.
2. We have $T = \mathsf{f}(T_1, \dots, T_n)$.
   For each $i \in \{1 \dots n\}$:
      $@res_i \leftarrow$ place$(T_i)$.
3. Let $T \leftarrow \mathsf{f}(@res_1, \dots, @res_n)$.
4. If $(@l : T) \in \mathcal{G}$ for some $@l$, choose one such $@l$ in any arbitrary way. Then
   return $@l$.
5. If $\#\mathcal{L}_\mathcal{G} <$ maxLabels, then $@l \leftarrow @l_{\mathsf{nextLabel}}$, and increment nextLabel. Other-
   wise, choose arbitrarily any $@l \in \mathcal{L}_\mathcal{G}$.
6. Let $\mathcal{G} \leftarrow \mathcal{G} \cup (@l : T)$, and return $@l$.
```

Fig. 1. The main algorithm

At the beginning, we choose an arbitrary bound $\mathsf{maxLabels}$ on the number of distinct labels that can be used in constructing $\mathcal{G}'$. During the execution of our algorithm this bound can be reached, and still a production for a ground term

T has to be added to $\mathcal{G}$. If the term T is not plain, a Chomsky normalization is in order, that requires a fresh, unavailable label. Rule Augment comes to our rescue: we can safely reuse any of the labels already employed. Of course, this causes a loss of precision in the language of the selected label, yet the result is sound w.r.t. $\sqsubseteq_{\mathcal{R}}^{\mathcal{L}_{\mathcal{G}}}$. This is done in step 5 of the subroutine place of the algorithm.

We stress that the result is sound for *any* maxLabels bound. In fact, having just one label is enough to produce a sound approximation: the language consisting of *all* pure terms is generated by all the productions $@o : \mathsf{f}(\dots, @o, \dots)$ which only involve one label. Of course, one wants a more precise result than this, and therefore runs the algorithm with more resources.

We also put a bound maxJoin on the number of Join operations we allow during the algorithm run. The bounds maxJoin and maxLabels help in ensuring termination of the algorithm; its correctness is established below.

Theorem 1. *The algorithm is correct, i.e. given $\mathcal{R}, \mathcal{G}$, it always computes a fully-exposing $\mathcal{G}'$ such that $\mathcal{G} \sqsubseteq_{\mathcal{R}}^{\mathcal{L}_{\mathcal{G}}} \mathcal{G}'$. Also, $\mathcal{G}'$ is closed by* Transitivity.

5 Relaxing the Left Linearity Assumption

Unfortunately, many useful rewriting rules are not left–linear, e.g.

$$\mathcal{R} = \left\{ \mathsf{xor}(X, X) \Rightarrow 0 \qquad \mathsf{dec}(\mathsf{enc}(M, K), K) \Rightarrow M \right\}$$

In order to apply our technique to the above rules, we could start by approximating the rule set $\mathcal{R}$ with another left–linear rule set $\mathcal{R}_l$ for which $\rightarrow_{\mathcal{R}} \subseteq \rightarrow_{\mathcal{R}_l}$. Such a $\mathcal{R}_l$ could be

$$\mathcal{R}_l = \left\{ \mathsf{xor}(X_1, X_2) \Rightarrow 0 \qquad \mathsf{dec}(\mathsf{enc}(M, K_1), K_2) \Rightarrow M \right\}$$

While this would lead to a correct approximation of $\rightarrow^*_{\mathcal{R},\mathcal{G}}$, the precision of result would be very scarce. In the example above $\mathcal{R}_l$ allows us to decrypt any message, even if garbage is used instead of the proper key. Such an approximation of $\mathcal{R}$ will unlikely lead to the proof of any interesting security property.

Instead, we extend rewriting rules to have a set of equalities (which we write as $\widetilde{X} = \widetilde{Y}$) between variables as a side condition. We thus rewrite $\mathcal{R}$ as

$$\mathcal{R}' = \left\{ \begin{array}{ll} \mathsf{xor}(X_1, X_2) & {}_{X_1 = X_2} \Rightarrow 0 \\ \mathsf{dec}(\mathsf{enc}(M, K_1), K_2) & {}_{K_1 = K_2} \Rightarrow M \end{array} \right\}$$

Now the $\rightarrow_{\mathcal{R}'}$ rewritings are only allowed when the side condition is met; for instance, in $\mathcal{R}'$, X_1 and X_2 must actually match the same ground terms, as well as K_1 and K_2. With this extension, we can reuse our technique with minor changes, only: indeed, a revised version of Lemma 2 still holds:

Lemma 3 (Label Bindings Criterion). *Given $\mathcal{G}, \mathcal{R}$, let*

$$\simeq \ = \{ \langle @l_1, @l_2 \rangle \mid \exists T \in \mathcal{T}_{\mathsf{gr}}.\ @l_1 \rightarrow^*_{\mathcal{G}} T \wedge @l_2 \rightarrow^*_{\mathcal{G}} T \} \qquad (1)$$

If $\mathcal{G}$ satisfies

$$\left.\begin{array}{r}\forall (L_{\widetilde{X}=\widetilde{Y}} \Rightarrow R) \in \mathcal{R} \\ \forall \sigma : \mathcal{X} \to \mathcal{L} \\ \forall @l \in \mathcal{L}_{\mathcal{G}}\end{array}\right\} @l \to^*_{\mathcal{G}} \sigma L \wedge \forall i.\ \sigma \widetilde{X}_i \simeq \sigma \widetilde{Y}_i \implies @l \to^*_{\mathcal{G}} \sigma R \qquad (2)$$

then $\mathcal{G}$ is fully-exposing.

Note that properties (1) and (2) depend on both $\mathcal{G}$ and $\simeq$. Thus, we must adapt our algorithm to reach a dual fixed point, for $\mathcal{G}$ and $\simeq$.

To give an intuition of the new algorithm, consider the above $\mathcal{R}'$, and in particular $L = \mathsf{xor}(X_1, X_2)$. During the execution of step 5 of the (old) algorithm, assume to have a $\simeq$ satisfying (1) for the current $\mathcal{G}$. We look for some label binding σ such that $@l \to^*_{\mathcal{G}} \sigma L$. Now, before calling the subroutine $\mathsf{augment}(@l, \sigma R = 0)$ that would add the rewritten term to $\mathcal{G}$, we check $\sigma X_1 \simeq \sigma X_2$, i.e. whether the side condition holds. If the check succeeds, then we call $\mathsf{augment}(@l, 0)$; otherwise, we do nothing. When $\mathcal{G}$ changes, we must ensure that the relation $\simeq$ still satisfies (1). This forces us to update the relation $\simeq$ at each main loop iteration. The algorithm then becomes:

```
REVISED ALGORITHM Inputs: $\mathcal{G}, \mathcal{R}, \mathsf{maxLabels}, \mathsf{maxJoin}$
1. Start with $\mathcal{G} = \{@l_i : T_i\}$, where $0 \le i < s$ and $T_i \in \mathcal{T}_{\mathsf{pl}}$.
2. $\mathsf{nextLabel} \leftarrow s$, $\mathsf{numJoin} \leftarrow 0$
3. Close $\mathcal{G}$ under Transitivity.
4. If $\mathsf{numJoin} < \mathsf{maxJoin}$:
      Close $\mathcal{G}$ under Join. Increment numJoin for each joined label.
5. $\simeq \leftarrow \mathsf{approxIntersection}(\mathcal{G})$.
6. For each label $@l \in \mathcal{L}_{\mathcal{G}}$:
      For each rule $(L_{\widetilde{X}=\widetilde{Y}} \Rightarrow R) \in \mathcal{R}$:
         For each binding $\sigma \in \mathcal{X} \to \mathcal{L}$ such that $@l \to^*_{\mathcal{G}} \sigma L$:
            If $\forall i.\ \sigma \widetilde{X}_i \simeq \sigma \widetilde{Y}_i$:
               $\mathsf{augment}(@l, \sigma R)$.
7. Repeat from step 3, until $\mathcal{G}$ reaches a fixed point.
SUBROUTINE $\mathsf{approxIntersection}(\mathcal{G})$ Returns $\simeq$.
1. $\simeq \leftarrow \{\langle @l, @l\rangle | @l \in \mathcal{L}_{\mathcal{G}}\}$
2. If $(@a : \mathsf{f}(@a_1, \ldots, @a_n)), (@b : \mathsf{f}(@b_1, \ldots, @b_n)) \in \mathcal{G}$,
   and $\forall i.\ @a_i \simeq @b_i$:
3.    $\simeq \leftarrow \simeq \cup \{\langle @a, @b\rangle\}$
4. Repeat from step 2 until $\simeq$ reaches a fixed point.
5. return $\simeq$.
```

The correctness of the revised algorithm is established below.

Theorem 2. *Given $\mathcal{R}, \mathcal{G}$, the revised algorithm always outputs a fully-exposing $\mathcal{G}'$ such that $\mathcal{G} \sqsubseteq^{\mathcal{L}_{\mathcal{G}}}_{\mathcal{R}} \mathcal{G}'$. Also, $\mathcal{G}'$ is closed under* Transitivity.

Time Complexity From the complexity point of view, the revised algorithm is quite naïve, and would benefit from the use of more adequate data structures and more efficient subroutines. Yet, the time complexity is polynomial on the size of $\mathcal{G}$. However, we believe there is still room for improvement.

Theorem 3. *Let n be $\mathsf{maxLabels} + \mathsf{maxJoin}$, $\mathsf{maxArity}$ be the maximum arity of the function symbols occurring in $\mathcal{R}, \mathcal{G}$, and $\mathsf{maxLSize} = \max\{\mathsf{size}(L) | (L_{X=Y} \Rightarrow R) \in \mathcal{R}\}$. The revised algorithm runs in worst–case time complexity $\mathcal{O}(n^k \log n \cdot f(\mathcal{R}))$, where $k = \max(5 + 3 \cdot \mathsf{maxArity}, 2 + \mathsf{maxArity} \cdot (1 + \mathsf{maxLSize}))$, and $f(\mathcal{R})$ only depends on $\mathcal{R}$.*

5.1 Improving Precision

In order to compute a sound approximation of a language up to rewriting $[\![@a]\!]_{\mathcal{G}}/\mathcal{R}$, our algorithm approximates the set $[\![@a]\!]_{\mathcal{R},\mathcal{G}} \supseteq [\![@a]\!]_{\mathcal{G}}/\mathcal{R}$. This may lead to a loss of precision. For instance, consider $\mathcal{R} = \{\mathsf{f}(X) \Rightarrow 1\}$ and $\mathcal{G} = \{@a : \mathsf{f}(@a)\}$. Here $@a$ generates an empty language, thus the language up-to-rewriting $[\![@a]\!]_{\mathcal{G}}/\mathcal{R}$ is also empty. However, $1 \in [\![@a]\!]_{\mathcal{R},\mathcal{G}}$ since we have $@a \to_{\mathcal{G}} \mathsf{f}(@a) \to_{\mathcal{R}} 1$. Therefore, a correct (w.r.t. $\sqsubseteq_{\mathcal{R}}^{\mathcal{L}_{\mathcal{G}}}$) fully-exposing approximation $\mathcal{G}'$ must satisfy $@a \to_{\mathcal{G}'}^{*} 1$.

We can compute a more precise result (i.e. closer to $[\![@a]\!]_{\mathcal{G}}/\mathcal{R}$) by restricting rewriting so that it is only applicable to *inhabited* terms T, i.e. such that $[\![T]\!]_{\mathcal{G}} \neq \emptyset$. Our revised algorithm can be easily adapted to exploit this restriction. For instance, the rewriting of $T_1 = \mathsf{g}(@a, @b, @c)$ with $\mathsf{g}(X, Y, Z)_{Y=Z} \Rightarrow 1$ only happens if $T_1 \to_{\mathcal{G}}^{*} T_2 = \mathsf{g}(@a, T', T') \to_{\mathcal{R}} 1$ for some *ground* T'. When rewriting is restricted, we have that T_2 is inhabited, and so are $@a$ and T'. This also implies that rewriting occurs only if for some *pure* T'' we have $T_1 \to_{\mathcal{G}}^{*} \mathsf{g}(@a, T'', T'') \to_{\mathcal{R}} 1$. Therefore, we modify our algorithm so that it rewrites T_1 only when (i) $[\![@b]\!]_{\mathcal{G}} \cap [\![@c]\!]_{\mathcal{G}} \neq \emptyset$ and (ii) $[\![@a]\!]_{\mathcal{G}} \neq \emptyset$. More generally, we can require (i) above for each side condition and (ii) for each label matching a variable. Checking (i) is tackled by changing the definition of the $\simeq$ relation so that we have $@l_1 \simeq @l_2$ iff $[\![@l_1]\!]_{\mathcal{G}} \cap [\![@l_2]\!]_{\mathcal{G}} \neq \emptyset$: step 1 of the subroutine $\mathsf{approxIntersection}$ is rewritten as "$\simeq\ \leftarrow \emptyset$". Checking (ii) is reduced to checking (i) by adding trivial equations $X = X$ for all variables in each rule, thus requiring $[\![\sigma X]\!]_{\mathcal{G}} \neq \emptyset$.

The modified algorithm is correct. Technically, correctness follows from the obvious adaptation of Lemma 3, where the new definition of $\simeq$ uses pure terms $T \in \mathcal{T}_{\mathsf{pr}}$ and the notion of "fully-exposing" involves the restricted rewriting.

6 Implementation

We implemented the revised algorithm, with the improvements discussed in Section 5.1. Our implementation follows the presented algorithm rather faithfully, and therefore we do not discuss its details here: we only mention that we use some simple heuristics for subroutine place, when we have to choose among many suitable labels.

We applied our tool to several rewriting systems, including those for pairs $(\mathsf{cons}, \mathsf{fst}, \mathsf{snd})$, encryptions $(\mathsf{enc}, \mathsf{dec})$, and also those for xor and exponentials

$(\mathsf{exp}, *, \mathsf{inv}, 1)$ that do not admit normal forms. These experiments often produced approximations precise enough to show the security properties we were looking for. For instance, we successfully checked a one-time pad example and the wide-mouthed frog [10] protocol. A flawed version of the WEP protocol [9] did *not* pass the test, as expected. The tool showed its limitations when applied to an exotic definition of even and odd naturals: in that case the tool over-approximated the sets yielding the larger $\mathbb{N}$.

Below, we report on applying our tool to the following protocol, based on the Diffie-Hellman key exchange.

1. $A \to \mathsf{all} : \mathsf{g}$
2. $A \to B \ : \{\mathsf{g}^{\mathsf{a}}\}_{\mathsf{k1}}$
3. $B \to A \ : \{\mathsf{g}^{\mathsf{b}}\}_{\mathsf{k2}}$
4. $A \to B \ : \{M\}_{\mathsf{g}^{\mathsf{ab}}}$
5. $\dots$
6. $A \to \mathsf{all} : \mathsf{k1}, \mathsf{k2}$

Initially, the principals A and B share two long term secret keys $\mathsf{k1}, \mathsf{k2}$, and agree on a public finite field $\mathsf{GF}[p]$ (where p is a large prime), and public generator g of $\mathsf{GF}[p]^*$. In the second step, principal A generates a nonce a and sends B the result of $\mathsf{g}^{\mathsf{a}} (\bmod p)$, encrypted with the $\mathsf{k1}$ key. In the third step, B does the same, with its own nonce b and key $\mathsf{k2}$. Since both principals know the long term keys, they can compute $(\mathsf{g}^{\mathsf{b}})^{\mathsf{a}} = \mathsf{g}^{\mathsf{ab}} = (\mathsf{g}^{\mathsf{a}})^{\mathsf{b}} \pmod p$ and use this value as a session key to exchange the message M in the fourth step.

We study the robustness of this protocol against the active Dolev–Yao [12] adversary (such an adversary has full control over the public network, can reroute, discard or forge messages; further, he can apply any operation in the term algebra to terms learnt before). In particular, we are interested in the *forward secrecy* of the message M. That is, we want M to be kept secret even though later on the long term keys $\mathsf{k1}, \mathsf{k2}$ are disclosed (last step). The rewriting rules for encryption, multiplication, exponentiation, and inversion are taken from [14]:

$$\mathcal{R} = \left\{ \begin{array}{ll} \mathsf{dec}(\mathsf{enc}(X, K), K) \Rightarrow X & \\ *(1, X) \Rightarrow X & \mathsf{exp}(\mathsf{inv}(X), Y) \Rightarrow \mathsf{inv}(\mathsf{exp}(X, Y)) \\ *(X, Y) \Rightarrow *(Y, X) & *(X, *(Y, Z)) \Rightarrow *(*(X, Y), Z) \\ \mathsf{exp}(X, 1) \Rightarrow X & \mathsf{exp}(1, X) \Rightarrow 1 \\ \mathsf{inv}(\mathsf{inv}(X)) \Rightarrow X & \mathsf{exp}(\mathsf{exp}(X, Y), Z) \Rightarrow \mathsf{exp}(X, *(Y, Z)) \\ \mathsf{inv}(1) \Rightarrow 1 & \mathsf{exp}(*(Y, Z), X) \Rightarrow *(\mathsf{exp}(Y, X), \mathsf{exp}(Z, X)) \\ *(X, \mathsf{inv}(X)) \Rightarrow 1 & \mathsf{inv}(*(X, Y)) \Rightarrow *(\mathsf{inv}(X), \mathsf{inv}(Y)) \end{array} \right\}$$

Note that the term algebra $\mathcal{A}$ defined by the above rewriting rules is not the same algebra of $\mathsf{GF}[p]^*$. In fact, $\mathcal{A}$ satisfies more equations than the ones that hold in $\mathsf{GF}[p]^*$. For instance, operations in $\mathcal{A}$ do not specify which modulus is being used; e.g., inversion modulo n is simply written as $\mathsf{inv}(X)$ rather than $\mathsf{inv}(X, n)$. Therefore, we have *(a)* $*(X, \mathsf{inv}(X)) = 1$ and *(b)* $\mathsf{exp}(Y, *(X, \mathsf{inv}(X))) = Y$. However, *(a)* holds in $\mathsf{GF}[p]^*$ only if $\mathsf{inv}(X)$ is performed modulo p, while *(b)* holds only if $\mathsf{inv}(X)$ is performed modulo $\varphi(p) = p - 1$ (where φ is the Euler function). In spite of $\mathcal{A}$ being not equal to the $\mathsf{GF}[p]^*$ algebra, it *soundly* approximates it, i.e. all the equations that hold in $\mathsf{GF}[p]^*$ do hold in $\mathcal{A}$.

We write a grammar $\mathcal{G}$ that approximates the adversary knowledge $@k$ *before* the disclosure of the long term keys $\mathsf{k1}, \mathsf{k2}$. Initially, the adversary knows 1 and g and can generate an unlimited number of nonces.

$$@nonce : \mathsf{seed}, \mathsf{next}(@nonce) \qquad\qquad @k : 1, \mathsf{g}, @nonce.$$

The adversary then learns the messages as they are sent through the network. We thus add the following productions to $\mathcal{G}$.

$$@k : \mathsf{enc}(\mathsf{exp}(\mathsf{g}, \mathsf{a}), \mathsf{k1}), \mathsf{enc}(\mathsf{exp}(\mathsf{g}, \mathsf{b}), \mathsf{k2}), \mathsf{enc}(\mathsf{m}, \mathsf{exp}(\mathsf{dec}(@k, \mathsf{k2}), \mathsf{a}))$$

The last term models the actual operations performed by A in order to compute g^{ab}, i.e. taking some value from the network ($@k$), decrypting it with $\mathsf{k2}$, and raising it to the nonce a. Note that our adversary is active, and may provide A with any term it knows as a basis for the further computations of A. Further, we add the following productions to make $@k$ closed under every operation the adversary may do:

$$@k : \mathsf{enc}(@k, @k), \mathsf{dec}(@k, @k), \mathsf{inv}(@k), *(@k, @k), \mathsf{exp}(@k, @k)$$

We then proceed to define the adversary knowledge *after* the long term keys $\mathsf{k1}, \mathsf{k2}$ have been disclosed: it is sufficient to add $\mathsf{k1}$ and $\mathsf{k2}$ to the language of $@k$ and to close it under the adversary operations to obtain the new language $@l$

$$@l : @k, \mathsf{k1}, \mathsf{k2}, \mathsf{enc}(@l, @l), \mathsf{dec}(@l, @l), \mathsf{inv}(@l), *(@l, @l), \mathsf{exp}(@l, @l)$$

Checking the forward secrecy property boils down to showing that $\mathsf{m} \notin [\![@l]\!]_{\mathcal{R},\mathcal{G}}$. We ran our implementation on $\mathcal{R}, \mathcal{G}$: it took about one minute[1] to generate $\mathcal{G}'$, having 185 productions. Within $\mathcal{G}'$ there is no production $@l : \mathsf{m}$, and therefore, by Theorem 2, we have $\mathsf{m} \notin [\![@l]\!]_{\mathcal{R},\mathcal{G}}$, thus establishing forward secrecy.

Our tool was also applied to a variant of this protocol, closer to the original Diffie-Hellman scenario. It uses instead digital signatures to carry the exponents g^{a} and g^{b}. We modeled the adversary roughly as done above, and run the tool on the usual rewriting rules for signatures. The tool succeeded in producing a $\mathcal{G}'$ (245 productions) proving the secrecy of message m.

References

1. M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 5(46):18–36, 1999.
2. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL'01), pages 104-115.*, 2001.
3. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The Spi calculus. *Journal of Information and Computation*, 148(1):1–70, 1999.

[1] Ref.: PowerPC G4 1.25GHz, 768MB RAM, $\mathsf{maxLabels} = \#\mathcal{L}_{\mathcal{G}} + 6$, $\mathsf{maxJoin} = +\infty$

4. B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005)*, 2005.
5. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. R. Nielson. Automatic validation of protocol narration. *To appear in Journal of Computer Science.* Abridged version in Proc. of CSFW 16, pages 126–140, 2003.
6. C. Bodei, P. Degano, F. Nielson, and H. R. Nielson. Static analysis for the π-calculus with application to security. *Journal of Information and Computation*, 168(1):68–92, 2001.
7. C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Static analysis for secrecy and non-interference in networks of processes. *Lecture Notes in Computer Science*, 2127, 2001.
8. C. Bodei, P. Degano, H.R. Nielson, and F. Nielson. Flow logic for Dolev-Yao secrecy in cryptographic processes. *Future Generation Computer Systems*, 18(6), 2002.
9. Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications: The insecurity of 802.11. http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html, 2001.
10. M. Burrows, M. Abadi, and R.M. Needham. A logic of authentication. In *Proc. of the Royal Society of London A*, volume 426, pages 233–271, 1989.
11. L. Cardelli and A.D. Gordon. Types for mobile ambients. In *Proc. of POPL'99*, pages 79–92. ACM Press, 1999.
12. D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, 1983.
13. J. K. Millen. The Interrogator: A tool for cryptographic protocol security. In *Proceedings of the 1984 Symposium on Security and Privacy*, pages 134–141, 1990.
14. J. K. Millen and V. Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *Computer Security Foundations Workshop*, 2003.
15. R. Milner. *Communicating and Mobile Systems: the π-Calculus.* Cambridge University Press, 1999.
16. J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murϕ. In *Proceedings of the 1997 Conference on Security and Privacy*, pages 141–153, 1997.
17. F. Nielson, H. R. Nielson, and H. Seidl. Cryptographic analysis in cubic time. *Electronic Notes in Theoretical Computer Science*, 62, 2002.
18. F. Pottier. A simple view of type-secure information flow in the π-calculus. In *CSFW-15*, pages 320–330, 2002.
19. R. Zunino. Control flow analysis for the applied π–calculus. In *Proceedings of the MEFISTO Project 2003*, volume ENTCS 99, pages 87–110, 2004.

A Proofs

Proof for Lemma 1 (Label-in-the-middle)

If $@l \to_{\mathcal{G}}^* C[T]$ for some $@l \in \mathcal{L}_{\mathcal{G}}$, $C[\bullet] \in \mathcal{C}_{\mathsf{gr}}$ and $T \in \mathcal{T}_{\mathsf{gr}}$, then for some label $@m \in \mathcal{L}_{\mathcal{G}}$ we have $@l \to_{\mathcal{G}}^* C[@m]$ and $@m \to_{\mathcal{G}}^* T$.

Proof. Easy induction on the number of derivations in $\to_{\mathcal{G}}^*$. □

Proof for Lemma 2

If $\mathcal{G}$ satisfies

$$\forall (L \Rightarrow R) \in \mathcal{R}.\ \forall \sigma : \mathcal{X} \to \mathcal{L}\ .\ \forall @l \in \mathcal{L}_{\mathcal{G}}.\ @l \to_{\mathcal{G}}^* \sigma L \implies @l \to_{\mathcal{G}}^* \sigma R \tag{3}$$

then $\mathcal{G}$ is fully-exposing.

Proof. First, property (3) suffices to ensure the following apparently stronger property, which involves *any* binding σ:

$$\forall (L \Rightarrow R) \in \mathcal{R}.\ \forall \sigma : \mathcal{X} \to \mathcal{T}_{\mathsf{gr}}\ .\ \forall @l \in \mathcal{L}_{\mathcal{G}}.\ @l \to_{\mathcal{G}}^* \sigma L \implies @l \to_{\mathcal{G}}^* \sigma R \tag{4}$$

This is because, whenever $\sigma X \notin \mathcal{L}$, we can apply the "label-in-the-middle" lemma, and find a label which is responsible for σX, i.e. a label $@m \in \mathcal{L}_{\mathcal{G}}$ such that $@l \to_{\mathcal{G}}^* C[@m] \to_{\mathcal{G}}^* C[\sigma X] = \sigma L$. Exploiting the left–linearity of $\mathcal{R}$, we can then update the binding to $\sigma' = \sigma[X \mapsto @m]$ and still have $@l \to_{\mathcal{G}}^* \sigma' L$. We repeat this process for any variable $X_1, \ldots, X_n$ occurring in L. In this way we obtain $\sigma'' \in \mathcal{X} \to \mathcal{L}$ such that $@l \to_{\mathcal{G}}^* C[@m_1, \ldots, @m_n] = \sigma'' L$ and, for each i, $@m_i \to_{\mathcal{G}}^* \sigma X_i$. By hypothesis (3), this implies

$$@l \to_{\mathcal{G}}^* \sigma'' R = C'[\ldots, @m_{i_k}, \ldots] \to_{\mathcal{G}}^* C'[\ldots, \sigma X_{i_k}, \ldots] = \sigma R.$$

and therefore $@l \to_{\mathcal{G}}^* \sigma R$, completing the proof for (4).

We now proceed and show $[\![@l]\!]_{\mathcal{R},\mathcal{G}} = [\![@l]\!]_{\mathcal{G}}$, for each $@l \in \mathcal{L}_{\mathcal{G}}$. Since $[\![T]\!]_{\mathcal{G}} \subseteq [\![T]\!]_{\mathcal{R},\mathcal{G}}$ for any $T \in \mathcal{T}_{\mathsf{gr}}$, we only need to show $[\![@l]\!]_{\mathcal{R},\mathcal{G}} \subseteq [\![@l]\!]_{\mathcal{G}}$.

This proof is by contradiction: assume that some T exists such that $@l \to_{\mathcal{G}}^* \to_{\mathcal{R}} T$ but for which $@l \not\to_{\mathcal{G}}^* T$. This means that, for some rule $(L \Rightarrow R) \in \mathcal{R}$, we have

$$@l \to_{\mathcal{G}}^* C[\sigma L] \to_{\mathcal{R}} C[\sigma R] = T$$

By the "label-in-the-middle" lemma, for some label $@m \in \mathcal{L}_{\mathcal{G}}$, $@l \to_{\mathcal{G}}^* C[@m]$ and $@m \to_{\mathcal{G}}^* \sigma L$, which by (4) implies $@m \to_{\mathcal{G}}^* \sigma R$. This shows that

$$@l \to_{\mathcal{G}}^* C[@m] \to_{\mathcal{G}}^* C[\sigma R] = T$$

which contradicts $@l \not\to_{\mathcal{G}}^* T$. □

Proof for Theorem 1

The main algorithm is correct, i.e. given $\mathcal{R}, \mathcal{G}$, it always computes a fully-exposing $\mathcal{G}'$ such that $\mathcal{G} \sqsubseteq_{\mathcal{R}}^{\mathcal{L}_{\mathcal{G}}} \mathcal{G}'$. Also, $\mathcal{G}'$ is closed by $\mathsf{Transitivity}$.

Proof. We start by showing the termination of the algorithm. First, we show that the size of $\mathcal{G}$ is bounded because:

- the productions in $\mathcal{G}$ only carry *plain* terms, which have bounded depth;
- no new function symbols are introduced;
- the function symbols have *fixed* arity;
- the number of labels in $\mathcal{G}$ is bounded by $\mathsf{maxLabels}$.

Termination is proved by contradiction, assuming our algorithm loops indefinitely in steps $3 \ldots 6$. Since we never exit the loop, and the number of Join operations is bounded by $\mathsf{maxJoin}$, eventually only rules $\mathsf{Augment}$ and $\mathsf{Transitivity}$ have to be applied. These rules make the size of $\mathcal{G}$ to indefinitely grow, so contradicting the first fact proved above.

Having established termination, by Lemma 2 the resulting grammar $\mathcal{G}'$ is fully-exposing. Moreover, since the algorithm only applies the grammar transformations of Section 3.1, we have $\mathcal{G} \sqsubseteq_{\mathcal{R}}^{\mathcal{L}_{\mathcal{G}}} \mathcal{G}'$. □

Proof for Lemma 3

Given $\mathcal{G}, \mathcal{R}$, let

$$\simeq \; = \{\langle @l_1, @l_2\rangle \mid \exists T \in \mathcal{T}_{\mathsf{gr}}.\ @l_1 \to^*_{\mathcal{G}} T \wedge @l_2 \to^*_{\mathcal{G}} T\}$$

If $\mathcal{G}$ satisfies

$$\left.\begin{array}{r} \forall (L_{\widetilde{X}=\widetilde{Y}} \Rightarrow R) \in \mathcal{R} \\ \forall \sigma : \mathcal{X} \to \mathcal{L} \\ \forall @l \in \mathcal{L}_{\mathcal{G}} \end{array}\right\} @l \to^*_{\mathcal{G}} \sigma L \wedge \forall i.\ \sigma \widetilde{X}_i \simeq \sigma \widetilde{Y}_i \implies @l \to^*_{\mathcal{G}} \sigma R$$

then $\mathcal{G}$ is fully-exposing.

Proof. Analogous to Lemma 2. □

Proof for Theorem 2

Given $\mathcal{R}, \mathcal{G}$, the revised algorithm always outputs a fully-exposing $\mathcal{G}'$ such that $\mathcal{G} \sqsubseteq^{\mathcal{L}_{\mathcal{G}}}_{\mathcal{R}} \mathcal{G}'$. Also, $\mathcal{G}'$ is closed under Transitivity.

Proof. Termination is established as for Theorem 1: we only note that $\simeq \subseteq \mathcal{L}_{\mathcal{G}} \times \mathcal{L}_{\mathcal{G}}$ implies that subroutine approxIntersection terminates.

A simple inductive argument shows the last $\simeq$ computed satisfies (1) w.r.t. $\mathcal{G}'$. By Lemma 3, $\mathcal{G}'$ is thus fully-exposing. Finally, we have that $\mathcal{G} \sqsubseteq^{\mathcal{L}_{\mathcal{G}}}_{\mathcal{R}} \mathcal{G}'$ since the algorithm only applies the $\sqsubseteq^{\mathcal{L}_{\mathcal{G}}}_{\mathcal{R}}$–preserving transformations of Section 3.1. □