INSTITUT FÜR INFORMATIK

UND PRAKTISCHE MATHEMATIK

Automata-based Analysis of Recursive Cryptographic Protocols

Ralf Küsters and Thomas Wilke

Bericht Nr. 0311
September 2003

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

KIEL

Institut für Informatik und Praktische Mathematik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

Automata-based Analysis of Recursive Cryptographic Protocols

Ralf Küsters and Thomas Wilke

Bericht Nr. 0311
September 2003

e-mail:
kuesters@theory.stanford.edu,wilke@ti.informatik.uni-kiel.de

Automata-based Analysis of Recursive Cryptographic Protocols

Ralf Küsters
Stanford University
kuesters@theory.stanford.edu

Thomas Wilke
Christian-Albrechts-Universität zu Kiel
wilke@ti.informatik.uni-kiel.de

Abstract

Cryptographic protocols can be divided into (1) protocols where the protocol steps are simple from a computational point of view and can thus be modeled by simple means, for instance, single rewrite rules—we call these protocols *non-looping*—and (2) protocols, such as group protocols, where the protocol steps are complex and typically involve an iterative or recursive computation—we call them *recursive*. While many results on the decidability of security are known for non-looping protocols, only little is known for recursive protocols. In this paper, we prove decidability of security (w.r.t. the standard Dolev-Yao intruder) for a core class of recursive protocols and undecidability for several extensions. The key ingredient of our protocol model are specifically designed tree transducers which work over infinite signatures and have the ability to generate new constants (which allow us to mimic key generation). The decidability result is based on an automata-theoretic construction which involves a new notion of regularity, designed to work well with the infinite signatures we use.

1 Introduction

In most cryptographic protocols, principals are described by a fixed sequence of what we call *receive-send actions*. When performing such an action, a principal receives a message from the environment and, after some internal computation, reacts by returning a message to the environment. Research on automatic protocol analysis [24, 2, 4, 19] has concentrated on protocols where a receive-send action can basically be described by a single rewrite rule of the form $t \to t'$: When receiving a message m, the message $\sigma(t')$ is returned as output provided that σ is the matcher for t and m, i.e., $\sigma(t) = m$. In other words, an input message is processed by applying the rewrite rule *once* on the top-level. We call receive-send actions of this kind and protocols based on such receive-send actions *non-looping*. It has been proved that for non-looping protocols when analyzed w.r.t. a finite number of receive-send actions and the standard Dolev-Yao intruder where the message size is not bounded, security (more precisely, secrecy) is decidable even when principals can perform equality tests on arbitrary messages [24, 2, 4, 19], complex keys are allowed [24, 4, 19], and the free term algebra assumption is relaxed by algebraic properties of XOR and Diffie-Hellman Exponentiation [7, 10, 8].

The main question we are concerned with in this paper is in how far security is decidable for protocols where receive-send actions are complex and typically involve an iterative or recursive computation; we call such receive-send actions and protocols containing such actions *recursive*. The answer to this question is not at all obvious since protocol models for non-looping protocols do not capture recursive protocols and there are almost no decidability results for recursive protocols (see the related work).

To illustrate the kind of receive-send actions performed in recursive protocols, let us consider the key distribution server S of the Recursive Authentication (RA) Protocol [6] (see also Section 7). In this protocol, the server S needs to perform the following *recursive* receive-send action: The server S first receives an a priori unbounded sequence of requests of pairs of principals who want to share session keys. Then, S generates sessions keys, and finally sends a sequence of certificates (corresponding to the requests) containing the session keys. Receive-send actions of this kind are typical for group protocols, but also occur in protocols such as the Internet Key Exchange protocol (IKE)—see [18] for a description of some recursive protocols. As pointed out by Meadows [18] and illustrated in [26, 13], modeling recursion is security relevant.

A natural way to describe recursive receive-send actions is by tree transducers, which extend the class of transductions expressible by single rewrite rules (with linear left-hand side). More precisely, to study decidability, in Section 2 we introduce non-deterministic top-down tree transducers (TTACs) with look-ahead and epsilon transitions which work on a signature containing an *infinite* set of what we call *anonymous constants* (ACs), over which the TTACs has only very limited control. TTACs can generate new (anonymous) constants, a feature often needed to model recursive receive-send actions; in the RA protocol for instance, the key distirbution server needs to generate (an a priori unbounded number of) session keys.

The main result of this paper is that i) security (for a finite number of receive-send actions, atomic keys, and the standard Dolev-Yao intruder where the message size is not bounded) is decidable if receive-send actions are modeled by TTACs (Section 5), and that ii) certain features of models for non-looping protocols cannot be added without losing decidability: As soon as TTACs are equipped with the ability to perform equality tests between arbitrary messages, complex keys are allowed, or the free term algebra assumption is relaxed by adding XOR or Diffie-Hellman Exponentiation security is undecidable (Section 6).

The undecidability results are obtained by reductions from Post's Correspondence Problem. The decidability result is obtained in two steps. First, we show that TTACs are powerful enough to simulate the intruder. This allows us to describe attacks as the composition of transducers. We can then reduce the security problem to the iterated pre-image word problem for a composition of TTACs, which we show to be decidable (Section 3): Given a term t, a "regular set" R of terms, and a sequence of TTACs, the *iterated pre-image word problem* asks whether on input t the composition of TTACs can produce an output in R. Here, "regular set" means the set of terms recognizable by a new kind of tree automata, *tree automata over signatures with anonymous constants* (TAACs), which can compare anonymous constants for equality.

See the technical report (enclosed as appendix) for detailed proofs of all results presented here.

Related work. Recursive protocols have been analyzed manually [23] and semi-automatically using theorem provers or special purpose tools [22, 5, 17].

Decidability for recursive protocols has initially been investigated in a previous paper [15]. However, there are significant differences to the present paper. First, in [15] *word* transducers are employed, which are much less powerful than tree transducers. Therefore, tree transducers provide a much clearer picture of the differences between recursive and non-looping protocols, and also allow to trace a tighter boundary of decidability. Second, generating new constants (e.g., session keys) has not been considered in [15]. Third, TTAC-based models of (recursive) protocols are in general much more precise than models based on word transducers because session keys can be generated and nonces need not necessarily be typed (in Section 7 this is illustrated for the RA protocol). Fourth, the proof techniques

employed are different. In [15], a quite involved and technical pumping argument was used to obtain decidability since word transducers are not powerful enough to simulate the intruder. In the current paper, the characterization of attacks in terms of the composition of transducers allows for a much more elegant proof and anonymous constants present a completely new challenge.

Motivated by the analysis of cryptographic protocols, in the present work we study automata and transducers over infinite signatures (alphabets). This is an import topic in automata theory which is relevant also in other areas such as type checking and type inference for XML queries with data values (see, e.g., [1]), although the settings studied here and in the context of XML are quite different. For XML queries it has been pointed out that equality tests between data values (which correspond to our anonymous constants) often lead to undecidability. Word automata over infinite alphabets have also been investigated (see, e.g., [20]).

Structure of the paper. In Section 2, we introduce tree automata (TAACs) and transducers (TTACs) over signatures with anonymous constants and prove basic properties. Section 3 provides the definition of the iterated pre-image word problem and the proof that this problem is decidable for TTACs. Our tree transducer-based protocol model is presented in Section 4. In Section 5 we show that security in this model is decidable. The purpose of Section 6 is i) to briefly discuss the relationship between our model and models for non-looping protocols, and ii) to prove the mentioned undecidability results. Section 7 contains formal TTAC-based models of the Recursive Authentication Protocol and the Needham Schroeder Public Key Authentication Protocol. We conclude in Section 8.

Basic definitions and notation. A *symbol* is an object with an *arity* assigned to it. A symbol of arity 0 is called a *constant (symbol).* A *signature* is a set of symbols. When Σ denotes a signature, then Σ_n denotes the set of symbols from Σ with arity n.

The set of terms over a signature Σ is denoted T_Σ. For a set C of constant symbols disjoint from a signature Σ, $T_\Sigma(C) = T_{\Sigma \cup C}$.

We fix an infinite supply X of variables among which we find x_0, x_1, x_2, ... For $n \in \mathbf{N}$, we write T^n_Σ for the set of all terms in $T_\Sigma(\{x_0, \ldots, x_{n-1}\})$. A term $t \in T^n_\Sigma$ is *linear* if every x_i with $i < n$ occurs exactly once in t. When $t \in T^n_\Sigma$ and $t_0, \ldots, t_{n-1}$ are arbitrary terms, we write $t[t_0, \ldots, t_{n-1}]$ for the term which is obtained from t by simultaneously substituting t_i for x_i, for every $i < n$. A substitution over Σ is a function $\sigma \colon T_\Sigma(X) \to T_\Sigma(X)$ such that for each term t, $\sigma(t)$ is obtained from t by simultaneously substituting $\sigma(x)$ for x for every $x \in X$.

By $\mathbf{N}^*$ we denote the set of finite strings over the non-negative integers $\mathbf{N}$. The empty string is ε. With $\leq$ we denote the prefix ordering on $\mathbf{N}^*$, i.e., for $v, w \in \mathbf{N}^*$, $v \leq w$ iff there exists $v' \in \mathbf{N}^*$ such that $vv' = w$, where vv' denotes the concatenation of v and v'. In this case, v is called a *prefix* of w. A set $S \subseteq \mathbf{N}^*$ is called *prefix closed* if with $v \in S$ every prefix of v belongs to S.

We use the notions "term" and "tree" interchangeably since a term t can be seen as a tree. Formally, a tree is a mapping from a non-empty, finite, and prefix closed set $S \subseteq \mathbf{N}^*$ into Σ such that if $t(\pi) \in \Sigma_n$ for some $n \geq 0$ and $\pi \in S$, then $\{i \mid \pi i \in S \text{ and } i \geq 0\} = \{0 \ldots, n-1\}$, and if $t(\pi)$ is a variable, then $\{i \mid \pi i \in S \text{ and } i \geq 0\} = \emptyset$. We call S the *set of positions* of t and denote this set by $\mathcal{P}(t)$.

For a term t and $\pi \in \mathcal{P}(t)$, $t|_\pi$ shall denote the *subterm* of t at position π, i.e., $\mathcal{P}(t|_\pi) := \{\pi' \mid \pi\pi' \in \mathcal{P}(t)\}$ and $t|_\pi(\pi') := t(\pi\pi')$ for every $\pi' \in \mathcal{P}(t|_\pi)$.

A subset τ of $T_\Sigma \times T_\Sigma$ is called a *transduction* over Σ. For a term t, we define $\tau(t) = \{t' \mid (t, t') \in \tau\}$. If τ and τ' are transductions over Σ, then their *composition* $\tau \circ \tau'$ defines the transduction

$\{(t, t') \mid \exists t'' \in T_\Sigma \text{ s.t. } (t, t'') \in \tau' \wedge (t'', t') \in \tau\}$, i.e., the composition is read from right to left. Given a transduction τ over Σ and a set $R \subseteq T_\Sigma$, the *pre-image* of R under τ is the set $\tau^{-1}(R) = \{t \mid \exists t' \in R \text{ s.t. } (t, t') \in \tau\}$.

2 Tree Automata and Transducers with Anonymous Constants

In this section we describe the models of tree automata and transducers that we use, completely independent of the application we have in mind, as they are of general interest.

2.1 Signatures and Anonymous Constants

A pair (Σ, C) consisting of a finite signature Σ and an arbitrary *infinite* set C of constant symbols disjoint from Σ is called a *signature with anonymous constants;* the elements of Σ and C are referred to as *regular symbols* and *anonymous constants,* respectively. With such a signature, we associate the signature $\Sigma \cup C$, denoted Σ^C. That is, when we speak of a *term over* (Σ, C) we mean a term over Σ^C. In what follows, let $occ_C(t)$ ($occ_C(\mathcal{S})$) denote the set of elements from C that occur in the term t (the set of terms $\mathcal{S}$).

2.2 Tree Automata over Signatures with Anonymous Constants

Our tree automata are non-deterministic bottom-up tree automata that accept trees over signatures with anonymous constants; they have full control over the regular symbols but only very limited control over the anonymous constants. For instance, it will be the case that with every tree such an automaton accepts it accepts every tree which is obtained from this one just by permuting—consistently renaming—the anonymous constants.

Our tree automata have two distinguished states, q^d and q^s, which are used as initial states for the anonymous constants: In every run the automaton non-deterministically assigns q^d and q^s to the anonymous constants in an arbitrary fashion under the restriction that at most one anonymous constant, which is then called the *selected constant,* gets assigned q^s and all the others get assigned q^d, the *default value.* (For an exact definition see below.)

Formally, a *tree automaton (TAAC) over a signature with anonymous constants* (Σ, C) is a tuple

$$\mathbf{A} = (Q, q^d, q^s, \Delta, F) \tag{1}$$

where Q is a non-empty finite *set of states,* $q^d \in Q$ is the *default state,* $q^s \in Q$ is the *selecting state,* Δ is a finite *set of transitions* as specified below, and $F \subseteq Q$ is a *set of final states.* The latter can be omitted; in this case, we speak of a *semi TAAC.*

There are two types of transitions: A *consuming transition* is of the form $f(q_0, \ldots, q_{n-1}) \to q$ where $f \in \Sigma_n$, $q, q_0, \ldots, q_{n-1} \in Q$; an *epsilon transition* is of the form $q' \to q$ where $q', q \in Q$.

We call a TAAC *deterministic* if it does not contain epsilon transitions and if for every $f \in \Sigma_n$ and $q_0, \ldots, q_{n-1} \in Q$ there exists at most one $q \in Q$ such that $f(q_0, \ldots, q_{n-1}) \to q \in \Delta$.

Each TAAC over a signature with anonymous constants (Σ, C) defines a set of trees from $T_\Sigma(C)$. To describe this set, we view the set Q as a set of constants and define for each term $t \in T_\Sigma(C \cup Q)$ the set $[t]_\mathbf{A}$ of states which the automaton reaches after having read the term t.

We first give an inductive definition for $t \in T_\Sigma(Q)$. In this case, $[t]_\mathbf{A}$ is the smallest set satisfying the following rules:

- if $t \in Q$, then $t \in [t]_{\mathbf{A}}$,
- if $t = f(t_0, \ldots, t_{n-1})$ and there exist $q_0, \ldots, q_{n-1}$ such that $f(q_0, \ldots, q_{n-1}) \to q \in \Delta$ and $q_i \in [t_i]_{\mathbf{A}}$ for every $i < n$, then $q \in [t]_{\mathbf{A}}$,
- if $q \in [t]_{\mathbf{A}}$ and $q \to q' \in \Delta$, then $q' \in [t]_{\mathbf{A}}$.

A *permitted* substitution σ is a function $\sigma\colon C \to \{q^d, q^s\}$ where at most one element of C gets assigned q^s. Now, for an arbitrary $t \in T_\Sigma(C \cup Q)$, we set

$$[t]_{\mathbf{A}} = \bigcup_{\sigma \text{ permitted}} [\sigma(t)]_{\mathbf{A}} \ . \tag{2}$$

The *tree language recognized* by $\mathbf{A}$ is the language

$$T(\mathbf{A}) = \{t \in T_\Sigma(C) \mid F \cap [t]_{\mathbf{A}} \neq \emptyset\} \ . \tag{3}$$

We say a tree language over Σ^C is *TAAC recognizable* over (Σ, C) if it is recognized by some TAAC over (Σ, C).

Before we provide some examples of (not) TAAC recognizable languages, we introduce a notation which we will use later. Given a term $t \in T^n_{\Sigma^C}$ and sets $S_0, \ldots, S_{n-1} \subseteq Q$, we write

$$[t[S_0, \ldots, S_{n-1}]]_{\mathbf{A}} = \bigcup_{q_i \in S_i} [t[q_0, \ldots, q_{n-1}]]_{\mathbf{A}}.$$

Example 1 *Assume $\Sigma_2 = \{f\}$ and $\Sigma_i = \emptyset$ for every $i \neq 2$. Let $T_= = \{f(c, c) \mid c \in C\}$. This language is recognized by a TAAC with only three states, say q_0, q_1, and q_2. We choose $q^d = q_0$ and $q^s = q_1$, $F = \{q_2\}$ and have only one transition, namely $f(q_1, q_1) \to q_2$.*

Example 2 *Fix any signature (Σ, C) with anonymous constants. For every $i \leq 3$, let $T_{\geq i}$ be the tree language over Σ^C which contains a tree t iff in t at least i pairwise distinct anonymous constants occur. Then $T_{\geq i}$ is TAAC recognizable over (Σ, C) for $i \leq 2$, but not for $i = 3$.*

Example 3 *Fix any signature (Σ, C) with anonymous constants. For every i, let T_i be the tree language over Σ^C which contains a tree t iff in t there are at least i occurrences of anonymous constants. Then T_i is TAAC recognizable over (Σ, C) for every i.*

We will also use a weak form of TAACs where the default and the selecting state are required to be identical, that is, where $q^d = q^s$ holds. This means there is actually no selecting state. These automata will be called *weak TAAC (WTAAC)*. They are really weaker because it is easy to see that, for instance, $T_=$ is not WTAAC recognizable over (Σ, C).

We conclude this section by summarizing basic properties of TAACs and WTAACs. We start with a simple observation, which can be proved using a straightforward powerset construction.

Lemma 4 *Every TAAC is equivalent to (recognizes the same tree language as) a deterministic TAAC. The same holds true for WTAAC.*

In the following lemma we consider closure properties of TAACs and WTAACs. As we will see, the behavior of TAACs is quite different to that of tree automata over finite signatures.

Lemma 5 *Let (Σ, C) be a signature with anonymous constants. Then, the following is true.*

1. *The set of tree languages over (Σ, C) recognized by WTAACs over (Σ, C) is closed under union, intersection, and complement.*
2. *The set of tree languages over (Σ, C) recognized by TAACs over (Σ, C) is closed under union.*
3. *The set of tree languages over (Σ, C) recognized by TAACs over (Σ, C) is closed under complement iff $\Sigma = \Sigma_0 \cup \Sigma_1$. The same holds true for intersection.*

PROOF. Statement 1. This statement can be proved similar to the case of bottom-up tree automata over finite signatures. Closure under complement follows from the fact that every WTAAC can be turned into an equivalent deterministic automata. For closure under intersection, one constructs the product automaton of the two given WTAACs. The default state is the tuple consisting of the default states of the two WTAACs. For closure under union, one takes the union of the two automata. More precisely, one first modifies the automata such that the default states only occur on the left-hand side of transitions. Then, one renames the states in both automata such that the state spaces are disjoint, except that the default states are named the same. Now, one can take the union of the automata, i.e., the union of the state spaces, the set of transitions, and the set of final states.

Statement 2. To prove this statement one constructs the union of TAACs similar to the union of WTAACs.

Statement 3. It is easy to see that if $\Sigma = \Sigma_0 \cup \Sigma_1$, then a TAAC over (Σ, C) is equivalent to some WTAAC over (Σ, C). With Statement 1., closure under complement and intersection follows immediately.

We now show that TAACs are not closed under complement and intersection in case the only symbol in Σ is a binary symbol, say f. It is straighforward to extend this to any signature with at least one symbol of arity ≥ 2. In what follows, let $\#_c(t)$ be the number of occurrences of c in t.

For every $n \geq 1$, we define the tree language

$$L_n = \{f(t, t') \in T_{\Sigma^C} \mid \text{there exists } c \in C \text{ such that } \#_c(t) \neq \#_c(t') \bmod n\}.$$

It is easy to see that L_n is TAAC recognizable for every $n \geq 1$. However, we show that the complement $\overline{L}_2 = T_{\Sigma^C} \setminus L_2$ of L_2 is not TAAC recognizable. Obviously, $\overline{L}_2 = C \cup L_2^{=}$ where $L_2^{=} = \{f(t, t') \mid \#_c(t) = \#_c(t') \bmod n \text{ for every } c \in C\}$. Since C is TAAC recognizable and because of Statement 2., it suffices to show that $L_2^{=}$ is not TAAC recognizable. By contradiction, assume that $L_2^{=}$ is recognized by some TAAC $\mathbf{A}$ as in (1). Then, this automaton would accept the term $t = f(f(c_0, c_1), f(c_0, c_1))$ for two distinct anonymous constants c_0 and c_1. Thus, there exists a permitted subsitution σ such that $[\sigma(t)]_{\mathbf{A}} \cap F \neq \emptyset$. If $\sigma(c_0) = \sigma(c_1) = q^d$, then $\mathbf{A}$ would also accept $t' = f(f(c_0, c_1), f(c_0, c_2))$ where c_2 is a constant different from c_0 and c_1, a contradiction. We may therefore assume that $\sigma(c_0) = q^s$ (the case $\sigma(c_1) = q^s$ is symmetric). But then, $\mathbf{A}$ would again accept t', a contradiction. Consequently, there does not exist a TAAC which accepts $L_2^{=}$.

For the intersection, we consider the languages L_2 and L_3, which as mentioned are TAAC recognizable, and show that their intersection $L_2 \cap L_3$ is not TAAC recognizable. Assume that there exists a TAAC $\mathbf{A}$ as in (1) recognizing $L_2 \cap L_3$. For $c \in C$ and $n \geq 2$, let $c^n = f(c, f(c, \cdots f(c, c)))$ such that $\#_c(c^n) = n$. Let c_0 and c_1 be two distinct anonymous constants. Obviously, $t = f(f(c_0^3, c_1^2),$

$f(c_0^6, c_1^4)) \in L_2 \cap L_3$. Hence, there exists a permitted substitution σ such that $[\sigma(t)]_{\mathbf{A}} \cap F \neq \emptyset$. By considering different cases for σ, similar as above, one shows that different variants of t are recognized by $\mathbf{A}$ although they do not belong to $L_2 \cap L_3$, which leads to a contradiction. □

We finally note that for TAACs the word and emptiness problem are decidable. For the word problem—which asks whether given a term and a TAAC, the TAAC recognizes the term—this is obvious. For the emptiness problem—which asks whether given a TAAC, the language recognized by the TAAC is empty—one can show by the usual pumping argument that if a TAAC recognizes a tree then also a tree of depth bounded by the number of states of the TAAC. As an immediate consequence, one obtains a bound on the number of different anonymous constants to be considered. Together this implies decidability of the emptiness problem.

Lemma 6 *The word and the emptiness problem are decidable for TAACs (and thus, WTAACs).*

2.3 Tree Transducers over Signatures with Anonymous Constants

Tree tranducers come in many different flavors. Our model is designed in such a way that (1) the pre-image of a TAAC recognizable language is TAAC recognizable again and (2) we can (easily) model the cryptographic protocols and the adversary we want to. These two goals are opposed to each other: to achieve (1), the model needs to be weak, to achieve (2), it needs to be strong. An important aspect of (2) is that it will be necessary that an *unbounded* number of anonymous constants may be introduced by a tree transducer, but only in a very weak fashion.

Our model is a top-down tree transducer, that is, a given tree is transformed into a new tree according to certain rewrite rules, which are applied from the root of the tree to its leaves. There are serveral specific features: a WTAAC look-ahead; generation of new (!) anonymous constants; a register for one anonymous constant. In addition, our tree transducers may be non-determinstic and may contain epsilon transitions.

We need some more notation. We fix a signature (Σ, C) with anonymous constants and a finite set S of states, whose elements we view as binary symbols. We assume that we are given a set $V = \{v_R, v_N\}$ of two variables for anonymous constants: v_R represents the aforementioned register, v_N refers to a newly generated anonymous constant.

A *state term* is of the form $s(z, t)$ for $s \in S$, $z \in V \cup C \cup \{*\}$, and $t \in T_\Sigma(C \cup X)$. The term t is then called the *core term* of this term. If z belongs to some set $D \subseteq V \cup C \cup \{*\}$, then we say $s(z, t)$ is a *D-state term*.

Intuitively, a state term of the form $s(*, t)$ or $s(c, t)$ with $c \in C$ is part of a configuration of a transducer and means that the transducer is about to read t starting in state s where the register does not store a value or stores the anonymous constant *c*, respectively. To describe transitions we use state terms of the form $s(v_R, t)$, $s(v_N, t)$, and again $s(*, t)$, but not $s(c, t)$ (see below).

Formally, a *tree transducer (TTAC) over a signature with anonymous constants* (Σ, C) is a tuple

$$\mathbf{T} = (S, I, \mathbf{A}, \Gamma) \tag{4}$$

where

- S is a finite set of *states,*
- $I \subseteq S$ is a set of *inital states,*

- $\mathbf{A}$ is a WTAAC over (Σ, C), and
- Γ is a finite set of transitions as described below.

A *transition* is of the form

$$s(z,t) \to^q t'[v_R, v_N, t'_0, \ldots, t'_{r-1}] \tag{5}$$

where

- $q \in Q$ is the look-ahead,
- $s(z,t)$ is an $\{v_R, *\}$-state term (recall that this means that $z = v_R$ or $z = *$) with $t \in T_\Sigma^n$ and t linear,
- $t' \in T_\Sigma^{r+2}$ (not necessarily linear), where v_R does not occur in $t'[v_R, v_N, t'_0, \ldots, t'_{r-1}]$ if $z = *$, and
- each t'_i is either a variable x_j with $j < n$ or a $\{z, v_N, *\}$-state term with the core term being a subterm of t.

When v_N occurs in $t'[v_R, v_N, t_0, \ldots, t_{r-1}]$, then the transition is called *generative* and *non-generative* otherwise. Sometimes we omit the look-ahead q when we write transitions. This is equivalent to assuming that the look-ahead is some state q in which $\mathbf{A}$ accepts every term. If $\mathbf{A}$ does not contain such a state, then $\mathbf{A}$ can be extended accordingly. For $q \in Q$, we denote by $\mathbf{T}(q)$ the transducer $\mathbf{T}$ with q as its only inital state.

The computation the TTAC carries out is described by a sequence of rewrite steps. The corresponding rewrite relation $\vdash_U$ is defined w.r.t. a subset $U \subseteq C$ of anonymous constants to ensure that whenever the TTAC generates a new constant this constant does not belong to U. Later U will be the set of anyomous constants in the input term, which then guarantees that the anonymous constants generated by the TTAC are different from those occurring in the input.

To define $\vdash_U$, suppose we are given a term $u_0 = u_1[s(c, u_2)]$ where $u_2 = t[t_0, \ldots, t_{n-1}] \in T_{\Sigma C}$ and a transition τ as in (5) with $z = v_R$. Let σ be the substitution defined by $\sigma(x_i) = t_i$. Then, if $q \in [u_2]_{\mathbf{A}}$,

$$u_0 \vdash_U u_1[t'[c, c', \sigma(t'_0), \ldots, \sigma(t'_{r-1})]]$$

for every $c' \in C \setminus (occ_C(u_0) \cup U)$. Observe that if τ is non-generative, c' and U are irrelevant. Also note that the newly generated anonymous constant does not occur in U and in the output term computed so far. The rewrite step in case $u_0 = u_1[s(*, u_2)]$ is defined in the same way, where it is required that $z = *$.

A sequence $s(*,t) \vdash_U t_1 \vdash_U t_2 \vdash_U \cdots \vdash_U t'$ with t and t' terms over (Σ, C) is called a *computation.*

Let $\vdash_U^*$ denote the reflexive transitive closure of $\vdash_U$. We write $t \vdash^* t'$ as a short form for $t \vdash_{occ_C(t)} t'$. The relation on $T_\Sigma(C)$ defined by the TTAC is

$$\tau_{\mathbf{T}} = \{(t, t') \in T_{\Sigma C} \times T_{\Sigma C} \mid \exists s (s \in I \wedge s(*,t) \vdash^* t')\} \ . \tag{6}$$

We say that a transduction τ on (Σ, C) is *TTAC realizable* if there exists a TTAC $\mathbf{T}$ such that $\tau_{\mathbf{T}} = \tau$. We call two TTACs *equivalent* if they realize the same transduction.

Let us look at a reasonably complex example.

Example 7 *Let $\Sigma_0 = \{d\}$, $\Sigma_1 = \{f\}$, $\Sigma_2 = \{g\}$ and C an infinite set of anonymous constants. Consider the transduction τ on (Σ, C) where $(t, t') \in \tau$ if t does not contain f and t' is obtained from t by replacing every maximal subterm which does not contain anonymous constants by a term of the form $g(f(\dots f(c)\dots), f(\dots f(c)\dots))$ for a new anonymous constant c, where the arguments of g may be of different depth. We show that τ is TTAC realizable.*

Let $\mathbf{A}$ be the semi TAAC with states q_C, q_R, q_M, q_f where q_C is the default state and the transitions are:

$$\begin{aligned} d &\to q_R \ , \\ g(q_R, q_R) &\to q_R \ , \\ q_C &\to q_M \ , \\ g(q_R, q_M) &\to q_M \ , \\ g(q_M, q_R) &\to q_M \ , \\ g(q_M, q_M) &\to q_M \ , \\ d &\to q_f \ , \\ q_C &\to q_f \ , \\ g(q_f, q_f) &\to q_f \ . \end{aligned}$$

Then $q_f \in [t]_{\mathbf{A}}$ iff f does not occur in t; $q_R \in [t]_{\mathbf{A}}$ iff t does not contain f nor anonymous constants; and $q_M \in [t]_{\mathbf{A}}$ iff t does not contain f but an anonymous constant.

Now it is easy to construct the desired TTAC. We choose s_I to be its initial state and use the following transitions:

$$\begin{aligned} s_I(*, x_0) &\to^{q_f} s_0(*, x_0) \ , \\ s_0(*, x_0) &\to^{q_C} x_0 \ , \\ s_0(*, g(x_0, x_1)) &\to^{q_M} g(s_0(*, x_0), s_0(*, x_1)) \ , \\ s_0(*, x_0) &\to^{q_R} g(s_f(v_N, x_0), s_f(v_N, x_0)) \ , \\ s_f(v_R, x_0) &\to f(s_f(v_R, x_0)) \ , \\ s_f(v_R, x_0) &\to v_R \ . \end{aligned}$$

It is well-known [12] that the set of transductions realized by non-deterministic top-down tree transducers over finite signatures is not closed under composition. It is easy to see that this also holds true for TTACs.

Lemma 8 *The set of TTAC realizable transductions is not closed under composition.*

3 The Iterated Pre-image Word Problem

The objective of this section is to prove that the iterated pre-image word problem is decidable. This problem is defined as follows:

ITERATEDPREIMAGE. Given a term t over (Σ, C), a TAAC $\mathbf{B}$ over (Σ, C), and a sequence of TTACs $\mathbf{T_0}, \dots, \mathbf{T_{l-1}}$ over (Σ, C) with $\tau = \tau_{T_0} \circ \dots \circ \tau_{T_{l-1}}$, decide whether $t \in \tau^{-1}(T(\mathbf{B}))$.

The key for proving decidability of this problem is:

Theorem 9 *The pre-image of a TAAC recognizable tree language under a TTAC realizable transduction is a TAAC recognizable tree language. Moreover, an appropriate TAAC can be constructed effectively.*

Using this theorem and Lemma 6 (decidability of the word problem), we obtain:

Corollary 10 ITERATEDPREIMAGE *is decidable.*

The proof of Theorem 9 is carried out in two steps. We first show how TTACs can be turned into what we call simple TTACs. We then construct a TAAC recognizing the pre-image of a TAAC recognizable tree language under a simple TTAC.

3.1 Simple TTACs

We say that a transition of the form (5) is *simple* if

1. t is a variable—in this case we call the transition *epsilon transition*—or of the form $f(x_0, \ldots, x_{n-1})$ for some $f \in \Sigma_n$—in this case we call the transition Σ-*transition*—, and
2. for every $i < r$, t'_i is either a variable x_j or a state term of the form $s(z', x_j)$ for some j.

We call a TTAC *simple* if it only contains simple transitions.

Before we show that every TTAC can be turned into an equivalent simple TTAC, we observe:

Lemma 11 *For every linear term $t \in T^n_{\Sigma C}$, there exists a WTAAC $\mathbf{A_t}$ over (Σ, C) and a state, say q_t, in $\mathbf{A_t}$ such that q_t is the only final state of $\mathbf{A_t}$ and $T(\mathbf{A_t}) = \{t' \mid$ there exists a substitution σ such that $\sigma(t) = t'\}$.*

Lemma 12 *Every TTAC is equivalent to (i.e., induces the same transduction as) a simple TTAC, which can be constructed in polynomial time.*

PROOF. Let $\mathbf{T}$ be a TTAC as in (4) and let Γ contain a non-simple transition T of the form

$$s(z, t) \to^q t'[v_R, v_N, t'_0, \ldots, t'_{r-1}, x_{i_0}, \ldots, x_{i_{l-1}}] \tag{7}$$

where the t'_i are state terms.

We show how (7) can be turned into a set of simple transitions; by iterating this argument all non-simple transitions of $\mathbf{T}$ can be replaced by simple transitions. The idea is as follows: We first extend $\mathbf{A}$ by $\mathbf{A_t}$ (Lemma 11) to be able to check whether the input term matches with t. This is done in an epsilon transition with q_t as look-ahead. Another epsilon transition is used to check the look-ahead q. Now, we add transitions that allow to navigate from the input term to the x_{i_j} (i.e., the position of the input term corresponding to x_{i_j}) and transitions that allow to navigate to the core terms of the t'_i.

The automaton $\mathbf{A}$ is extended by $\mathbf{A_t}$ by taking the union of these automata as explained in the proof of Lemma 5.

To navigate to the different positions, we add to the states of $\mathbf{T}$ the set of states $\{p_{\pi,q} \mid \pi \in \mathcal{P}(t), q \in S\} \cup \{p'_\pi \mid \pi \in \mathcal{P}(t)\}$. The states $p_{\pi,q}$ are used to navigate to the core terms of the t'_i and with p'_π we navigate to the variables x_{i_j}. In the former case, the state is subscribed with q since once

the core term of t'_i is reached, the computation needs to continue in the state stipulated by t'_i. In the latter case, one merely needs to copy (the term substituted for) x_{i_j} into the output, and therefore, does not need to remember a state.

The transitions for navigating to the core terms of the t'_i are as follows: For every $f \in \Sigma_n$, $q \in Q$, $i \geq 0$, $z \in \{v_R, *\}$, and π with $i\pi \in \mathcal{P}(t)$ we add the following transition:

$$p_{i\pi,q}(z, f(x_0, \ldots, x_{n-1})) \to p_{\pi,q}(z, x_i),$$

where we set $p_{\varepsilon,q} = q$.

The transitions for navigating to the x_{i_j} are as follows: For every $f \in \Sigma_n$, $i \geq 0$, and π with $i\pi \in \mathcal{P}(t)$ we add the following transition:

$$p'_{i\pi}(*, f(x_0, \ldots, x_{n-1})) \to p'_{\pi}(*, x_i),$$

where $p'_{\pi}(*, x_i)$ is replaced by x_i in case $\pi = \varepsilon$.

Finally, we add the following two transitions to $\mathbf{T}$. The first one checks whether the input term matches t by using q_t as look-ahead. The second one checks the look-ahead q and initializes the navigation process.

The first transition is the following epsilon transition:

$$s(z, x) \to^{q_t} s'(z, x).$$

where s' is a new state.

For the second transition, we need some notation. Assume that $t'_i = s_i(z_i, t''_i)$ and that π_i is the position of t''_i in t, i.e., $t|_{\pi_i} = t''_i$; in case there are different positions with this property, we simply pick one. Also, let π'_j be the position of x_{i_j} in t. Now, the transition is the following epsilon transition:

$$s'(z, x) \to^{q} t'[v_R, v_N, p_{\pi_0,s_0}(z_0, x), \ldots, p_{\pi_{r-1},s_{r-1}}(z_{r-1}, x), p'_{\pi'_0}(*, x), \ldots, p'_{\pi'_{l-1}}(*, x)]$$

It is easy to see that the transducer obtained in this way is equivalent to $\mathbf{T}$. □

3.2 Construction of the TAAC Recognizing the Pre-image

In what follows, let

$$\mathbf{I} = (Q_I, q^d, q^s, \Delta_I, F_I)$$

be a TAAC over (Σ, C) and

$$\mathbf{T} = (Q_T, I_T, \mathbf{A}, \Gamma_T)$$

be a TTAC over (Σ, C) with

$$\mathbf{A} = (Q_A, q^d_A, q^d_A, \Delta_A)$$

as its look-ahead. We need to construct a TAAC

$$\mathbf{P} = (Q_P, q^d_P, q^s_P, \Delta_P, F_P)$$

over (Σ, C) such that

$$T(\mathbf{P}) = \tau^{-1}_{\mathbf{T}}(T(\mathbf{I})).$$

Due to Lemma 12, we may assume that $\mathbf{T}$ is simple. Thus, Γ_T consists of Σ-transitions of the form

$$q(z, f(x_0, \ldots, x_{n-1})) \to^{q_A} t'[v_R, \ldots, v_R, v_N, \ldots, v_N, t'_0, \ldots, t'_{r-1}, x_{i_0}, \ldots, x_{i_{l-1}}] \tag{8}$$

where t' is linear, t'_i is a $\{v_R, *\}$-state term of the form $q_i(z_i, x_{j_i})$, and v_R may only occur on the right-hand side of (8) if $z = v_R$, and epsilon transitions of the form

$$q(z, x) \to^{q_A} t'[v_R, \ldots, v_R, v_N, \ldots, v_N, t'_0, \ldots, t'_{r-1}, x, \ldots, x] \tag{9}$$

where t' is linear, t'_i is a $\{v_R, *\}$-state term of the form $q_i(z_i, x)$, and v_R may only occur on the right-hand side of (9) if $z = v_R$. Note that assuming t' to be linear is w.l.o.g.

Roughly speaking, the idea behind the construction of $\mathbf{P}$ is that in a run of $\mathbf{P}$ on $t \in T_{\Sigma^C}$, $\mathbf{P}$ simulates the runs of $\mathbf{I}$ on all possible outputs t' of $\mathbf{T}$ on input t simultaneously. The runs of $\mathbf{I}$ on the terms t' can, however, not be simulated as a whole but only in small pieces, namely, on every right-hand side of transitions of $\mathbf{T}$ at a time. The problem is that runs of $\mathbf{I}$ require a global condition, namely that the default and selecting states q^d and q^s of $\mathbf{I}$ are assigned to constants in a consistent way—by a permitted substitution. To capture this global condition we use that runs of $\mathbf{P}$ also meet such a global condition. More precisely, the permitted substitution in a run of $\mathbf{P}$ on t will determine the permitted substitutions that are considered in the runs of $\mathbf{I}$ on the trees t'. We distinguish the following cases:

1. If in a run of $\mathbf{P}$ on t one constant c is assigned to q^s_P, then this (and only this) constant will be assigned to q^s in the runs of $\mathbf{I}$ on the trees t'; in particular, all anonymous constants generated by $\mathbf{T}$ are assigned to q^d.

2. If in a run of $\mathbf{P}$ on t, all constants occurring in t are assigned to q^d_P, then these constants are assigned to q^d in the runs of $\mathbf{I}$ on the trees t' as well. The constants newly generated by $\mathbf{T}$ may or may not be assigned to q^s. One has to do some book keeping to guarantee that at most one of the new constants is assigned to q^s. The automaton $\mathbf{P}$ will simulate all runs of $\mathbf{I}$ on the trees t' w.r.t. to all permitted substitutions where at most one newly generated constant is assigned to q^s simultaneously.

We now provide the formal definition of $\mathbf{P}$. In Section 3.3, we show that $\mathbf{P}$ in fact recognizes the pre-image.

State Space of $\mathbf{P}$. The state space of $\mathbf{P}$ is defined to be

$$2^{Q_I} \times 2^{Q_A} \times \{\mathsf{yes}, \mathsf{no}\} \times 2^{Q_T \times \{q^d, q^s, *\} \times Q_I} \times 2^{Q_T \times \{q^d, *\} \times Q_I}.$$

We need the following notation. Let $b = (S, L, \alpha, M_d, M_s)$ be a state of $\mathbf{P}$. We define:

- $\mathsf{Iset}(b) = S$,
- $\mathsf{LA}(b) = L$,
- $\mathsf{seen}(b) = \alpha$,
- $D_{(q,s)}(b) = \{a \mid (q, s, a) \in M_d\}$ for every $q \in Q_T$ and $s \in \{q^d, q^s, *\}$, and

- $S_{(q,s)}(b) = \{a \mid (q,s,a) \in M_s\}$ for every $q \in Q_T$ and $s \in \{q^d, *\}$.

Let us explain the intuitive meaning of the different components of a state. The first component S collects the possible states reachable by $\mathbf{I}$ on the input tree. The second component L is used to store the possible values of the look-ahead of $\mathbf{T}$. The case $\alpha = \mathsf{yes}$ corresponds to 1. above and $\alpha = \mathsf{no}$ corresponds to 2. In $D_{(q,s)}(b)$ we collect all states reachable in a run of $\mathbf{I}$ on some output tree t' obtained by running $\mathbf{T}$ on t starting in state q where the register is $*$ (in case $s = *$) or a constant c (in case $s \in \{q^d, q^s\}$) not occuring in t. The runs of $\mathbf{I}$ on some t' are simulated w.r.t. a permitted subsitution that maps all new constants to the default state q^d—the capital D in $D_{(q,s)}(b)$ being reminiscent of this—and coincides with the permitted substitution used in the run of $\mathbf{P}$ on all constants occurring in t. The value of $s \in \{q^d, q^s\}$ determines what state the constant c in the register is assigned to. The interpretation of $S_{(q,s)}(b)$ is similar: Here, we assume that all constants in t are assigned to q^d and in the runs of $\mathbf{I}$ all permitted substitutions are considered which map all constants in t to q^d and at most one new constant to the selecting state q^s—the capital S in $S_{(q,s)}(b)$ being reminiscent of this. The case where the register is assigned to q^s does not need to be considered. The intuition behind the components of the states of $\mathbf{P}$ is formally captured in Lemma 15.

We now introduce some more notation. Given a transition as in (9), we write $t^{\varepsilon}_{q',q'',b}$ as abbreviation for the term

$$t'[q', \ldots, q', q'', \ldots, q'', D_{(q_0,s_0)}(b), \ldots, D_{(q_{r-1},s_{r-1})}(b), \mathsf{Iset}(b), \ldots, \mathsf{Iset}(b)]$$

and $t^{\varepsilon,k}_{q',q'',b}$ as abbreviation for the term

$$t'[q', \ldots, q', q'', \ldots, q'', D_{(q_0,s_0)}(b), \ldots, S_{(q_k,s_k)}(b), \ldots, D_{(q_{r-1},s_{r-1})}(b), \mathsf{Iset}(b), \ldots, \mathsf{Iset}(b)]$$

where in both cases $s_i = *$ if $z_i = *$, $s_i = q'$ if $z_i = v_R$, and $s_i = q''$ if $z_i = v_N$.

Given a transition as in (8), we write $t^{\varepsilon}_{q',q''}$ as abbreviation for the term

$$t'[q', \ldots, q', q'', \ldots, q'', D_{(q_0,s_0)}(b_{j_0}), \ldots, D_{(q_{r-1},s_{r-1})}(b_{j_{r-1}}), \mathsf{Iset}(b_{i_0}), \ldots, \mathsf{Iset}(b_{i_{l-1}})]$$

and $t^{\varepsilon,k}_{q',q''}$ as abbreviation for the term

$t'[q', \ldots, q', q'', \ldots, q'', D_{(q_0,s_0)}(b_{j_0}), \ldots, S_{(q_k,s_k)}(b_{j_k}), \ldots, D_{(q_{r-1},s_{r-1})}(b_{j_{r-1}}), \mathsf{Iset}(b_{i_0}),$
$\ldots, \mathsf{Iset}(b_{i_{l-1}})]$

where again in both cases $s_i = *$ if $z_i = *$, $s_i = q'$ if $z_i = v_R$, and $s_i = q''$ if $z_i = v_N$.

We need to define the epsilon closure of states. If $b \in Q_P$, its *epsilon closure* $\overline{b}$ is defined inductively as follows: Let $b_0 = b$.

- $\mathsf{Iset}(b_{i+1}) = \mathsf{Iset}(b)$, $\mathsf{LA}(b_{i+1}) = \mathsf{LA}(b)$, $\mathsf{seen}(b_{i+1}) = \mathsf{seen}(b)$,
- for every $q \in Q_T$ and $s \in \{q^d, q^s, *\}$, let $D_{(q,s)}(b_{i+1}) = D_{(q,s)}(b_i) \cup \{a \mid$ there exists an epsilon transition as in (9) such that $q_A \in \mathsf{LA}(b_i)$, ($z = *$ iff $s = *$), and $a \in [t^{\varepsilon}_{s,q^d,b_i}]_{\mathbf{I}}\}$, and
- for every $q \in Q_T$ and $s \in \{q^d, *\}$, let $S_{(q,s)}(b_{i+1}) = S_{(q,s)}(b_i) \cup \{a \mid$ there exists an epsilon transition as in (9) such that $q_A \in \mathsf{LA}(b_i)$, ($z = *$ iff $s = *$), and $a \in [t^{\varepsilon}_{s,q^s,b_i}]_{\mathbf{I}}$ or $a \in [t^{\varepsilon,k}_{s,q^d,b_i}]_{\mathbf{I}}$ for some $k\}$,

We define $\overline{b} = b_j$ for some $j \geq 0$ such that $b_j = b_{j+1}$.

The Default and Selecting States of $\mathbf{P}$. The default state q_P^d of $\mathbf{P}$ is defined as the epsilon closure $\overline{b^d}$ of the following state b^d:

- $\mathsf{Iset}(b^d) = [q^d]_{\mathbf{I}}$.
- $\mathsf{LA}(b^d) = [q_A^d]_{\mathbf{A}}$.
- $\mathsf{seen}(b^d) = \mathsf{no}$.
- $D_{(q,s)}(b^d) = \emptyset$ for every $q \in Q_T$ and $s \in \{q^d, q^s, *\}$.
- $S_{(q,s)}(b^d) = \emptyset$ for every $q \in Q_T$ and $s \in \{q^d, *\}$.

The selecting state q_P^s of $\mathbf{P}$ is defined as the epsilon closure $\overline{b^s}$ of the following state b^s:

- $\mathsf{Iset}(b^s) = [q^s]_{\mathbf{I}}$.
- $\mathsf{LA}(b^s) = [q_A^d]_{\mathbf{A}}$.
- $\mathsf{seen}(b^s) = \mathsf{yes}$.
- $D_{(q,s)}(b^s) = \emptyset$ for every $q \in Q_T$ and $s \in \{q^d, q^s, *\}$.
- $S_{(q,s)}(b^s) = \emptyset$ for every $q \in Q_T$ and $s \in \{q^d, *\}$.

In what follows, by abuse of notation we write q^d instead of q_P^d and q^s instead of q_P^s. In this way, we can use the same permitted substitutions for both $\mathbf{P}$ and $\mathbf{I}$. Recall that q^d and q^s are the default and the selecting states of $\mathbf{I}$, respectively.

Transitions of $\mathbf{P}$. For every $f \in \Sigma_n$ and $b_0, \ldots, b_{n-1} \in Q_P$, the automaton $\mathbf{P}$ contains the transition $f(b_0, \ldots, b_{n-1}) \to b$ where b is defined to be the epsilon closure $\overline{b'}$ of the following state b':

- $\mathsf{Iset}(b') = [f(\mathsf{Iset}(b_0), \ldots \mathsf{Iset}(b_{n-1}))]_{\mathbf{I}}$,
- $\mathsf{LA}(b') = [f(\mathsf{LA}(b_0), \ldots \mathsf{LA}(b_{n-1}))]_{\mathbf{A}}$,
- $\mathsf{seen}(b') = \mathsf{yes}$ if there exists i such that $\mathsf{seen}(b_i) = \mathsf{yes}$, and $\mathsf{seen}(b') = \mathsf{no}$ otherwise,
- $D_{(q,s)}(b') = \{a \mid$ there exists a Σ-transition as in (8) such that $q_A \in \mathsf{LA}(b')$, ($z = *$ iff $s = *$), and $a \in [t^{\varepsilon}_{s,q^d}]_{\mathbf{I}}\}$ for every $q \in Q_T$ and $s \in \{q^d, q^s, *\}$, and
- $S_{(q,s)}(b') = \{a \mid$ there exists a Σ-transition as in (8) such that $q_A \in \mathsf{LA}(b')$, ($z = *$ iff $s = *$), and $a \in [t^{\varepsilon}_{s,q^s}]_{\mathbf{I}}$ or $a \in [t^{\varepsilon,k}_{s,q^d}]_{\mathbf{I}}$ for some $k\}$ for every $q \in Q_T$ and $s \in \{q^d, *\}$.

Final States of $\mathbf{P}$. The set of final states F_P of $\mathbf{P}$ is defined as follows:

$$\begin{aligned} F_P \;=\; & \{b \mid \text{there exists } q \in I_T \text{ and } a \in F_I \text{ such that} \\ & (\mathsf{seen}(b) = \mathsf{yes} \text{ and } a \in D_{(q,*)}(b)) \text{ or } (\mathsf{seen}(b) = \mathsf{no} \text{ and } a \in S_{(q,*)}(b))\} \end{aligned}$$

3.3 Correctness of the Construction

The following proposition states that our construction is correct.

Proposition 13 $T(\mathbf{P}) = \tau_{\mathbf{T}}^{-1}(T(\mathbf{I}))$.

To prove this proposition, we need to prove two lemmas. The first lemma, which immediately follows from the construction, states that $\mathbf{P}$ is complete and deterministic, and that reachable states are epsilon closed.

Lemma 14 *For every t and permitted substitution σ, there exists exactly one state b such that $b \in [\sigma(t)]_{\mathbf{P}}$. This state is epsilon closed, i.e., $\overline{b} = b$.*

The second lemma is more involved, and it is the key for proving Proposition 13.

Lemma 15 *For every term $t \in T_{\Sigma^C}$, $b \in Q_P$, and permitted substitution σ such that $b \in [\sigma(t)]_{\mathbf{P}}$ the following is true, where we write $\sigma'|_{occ_C(t)} = \sigma|_{occ_C(t)}$ to say that σ' is a permitted substitution which coincides with σ on $occ_C(t)$.*

1. $\mathsf{Iset}(b) = [\sigma(t)]_{\mathbf{I}}$.
2. $\mathsf{LA}(b) = [t]_{\mathbf{A}}$.
3. $\mathsf{seen}(b) = \mathsf{yes}$ *iff there exists* $c \in occ_C(t)$ *such that* $\sigma(c) = q^s$.
4. $D_{(q,*)}(b) = \{a \mid t' \in T_{\Sigma^C}, \sigma'|_{occ_C(t)} = \sigma|_{occ_C(t)}, \sigma'(c) = q^d$ *for every* $c \in occ_C(t') \setminus occ_C(t)$, $a \in [\sigma'(t')]_{\mathbf{I}}$, *and* $q(*, t) \vdash^* t'\}$.
5. $D_{(q,q^d)}(b) = \{a \mid t' \in T_{\Sigma^C}, \sigma'|_{occ_C(t)} = \sigma|_{occ_C(t)}, \sigma'(c') = q^d$ *for every* $c' \in occ_C(t') \setminus occ_C(t)$, $\sigma'(c) = q^d$, $a \in [\sigma'(t')]_{\mathbf{I}}$, *and* $q(c, t) \vdash^* t'\}$ *for every* $c \notin occ_C(t)$.
6. *If* $\mathsf{seen}(b) = \mathsf{no}$, *then* $D_{(q,q^s)}(b) = \{a \mid t' \in T_{\Sigma^C}, \sigma'|_{occ_C(t)} = \sigma|_{occ_C(t)}, \sigma'(c) = q^s, a \in [\sigma'(t')]_{\mathbf{I}}$, *and* $q(c, t) \vdash^* t'\}$ *for every* $c \notin occ_C(t)$. *Note that* $\sigma(c') = q^d$ *for every* $c' \in occ_C(t)$ *since* $\mathsf{seen}(b) = \mathsf{no}$.
7. *If* $\mathsf{seen}(b) = \mathsf{no}$, *then* $S_{(q,*)}(b) = \{a \mid t' \in T_{\Sigma^C}, \sigma'|_{occ_C(t)} = \sigma|_{occ_C(t)}, a \in [\sigma'(t')]_{\mathbf{I}}$, *and* $q(*, t) \vdash^* t'\}$.
8. *If* $\mathsf{seen}(b) = \mathsf{no}$, *then* $S_{(q,q^d)}(b) = \{a \mid t' \in T_{\Sigma^C}, \sigma'|_{occ_C(t)} = \sigma|_{occ_C(t)}, \sigma'(c) = q^d, a \in [\sigma'(t')]_{\mathbf{I}}$, *and* $q(c, t) \vdash^* t'\}$ *for every* $c \notin occ_C(t)$.

Before proving this lemma, we use it to establish Proposition 13:

"$T(\mathbf{P}) \subseteq \tau_{\mathbf{T}}^{-1}(T(\mathbf{I}))$": Assume that $t \in T(\mathbf{P})$. It follows that there exists a permitted substitution σ and a final state $b \in F_P$ such that $b \in [\sigma(t)]_{\mathbf{P}}$. We consider two cases. First, assume that $\mathsf{seen}(b) = \mathsf{yes}$. Then, there exists $q \in I_T$ and $a \in F_I$ such that $a \in D_{(q,*)}(b)$. Lemma 15, 4. implies that there exists t' and a permitted substitution σ' such that $q(*, t) \vdash^* t'$ and $a \in [\sigma'(t')]_{\mathbf{I}}$, and thus, $t' \in T(\mathbf{I})$ since $a \in F_I$. This means that $t \in \tau_{\mathbf{T}}^{-1}(T(\mathbf{I}))$. Second, assume that $\mathsf{seen}(b) = \mathsf{no}$. Then, there exists

$q \in I_T$ and $a \in F_I$ such that $a \in S_{(q,*)}(b)$. By Lemma 15, 7. there exists t' and a permitted substitution σ' such that $q(*,t) \vdash^* t'$ and $a \in [\sigma'(t')]_{\mathbf{I}}$. Thus, $t' \in T(\mathbf{I})$ and $t \in \tau_{\mathbf{T}}^{-1}(T(\mathbf{I}))$.

"$T(\mathbf{P}) \supseteq \tau_{\mathbf{T}}^{-1}(T(\mathbf{I}))$": Assume that $t \in \tau_{\mathbf{T}}^{-1}(T(\mathbf{I}))$. This means that there exists t', $a \in F_I$, and a permitted substitution σ such that $q(*,t) \vdash^* t'$ for some $q \in I_T$ and $a \in [\sigma(t')]_{\mathbf{I}}$. Let $b \in [\sigma(t)]_{\mathbf{P}}$. Such a b exists and it is uniquely determined due to Lemma 14. We show that $b \in F_P$, and thus, $t \in T(\mathbf{P})$. First, assume that $\mathsf{seen}(b) = \mathsf{yes}$. Then, $\sigma(c) = q^d$ for every $c \notin \mathit{occ}_C(t)$ and by Lemma 15, 4. we can conclude that $a \in D_{(q,*)}(b)$. Otherwise, if $\mathsf{seen}(b) = \mathsf{no}$, Lemma 15, 7. implies that $a \in S_{(q,*)}(b)$. In both cases, we get that $b \in F_P$.

Proof of Lemma 15. Let $t \in T_{\Sigma^c}$, $b \in Q_P$, and σ be a permitted substitution such that $b \in [\sigma(t)]_{\mathbf{P}}$. The Statements 1., 2., and 3. are easy to see by the construction of $\mathbf{P}$. We prove 4.–8. simultaneously. We first show that the left-hand side is included in the right-hand side by structural induction on t and then establish the inclusion in the other direction by induction on the length of computations.

"$\subseteq$": Base case. Assume that $t \in C$. We know that $b = \overline{b^d}$ or $b = \overline{b^s}$. Let $b_0 = b^d$ or $b_0 = b^s$ (both cases can be dealt with in the same way). For b_0 the inclusions hold trivially. By induction on i, we show that they hold for b_{i+1}. We concentrate on 8. as it is one of the more interesting cases. The other inclusions can be shown analogously. We assume that $\mathsf{seen}(b) = \mathsf{no}$ and $a \in S_{(q,q^d)}(b_{i+1})$. We need to show for every $c \notin \mathit{occ}_C(t)$ that there exists t' and a permitted substitution σ' such that $\sigma'|_{\mathit{occ}_C(t)} = \sigma|_{\mathit{occ}_C(t)}$, $\sigma'(c) = q^d$, $q(c,t) \vdash^* t'$, and $a \in [\sigma'(t')]_{\mathbf{I}}$. If $a \in S_{(q,q^d)}(b_i)$, this follows by the induction hypothesis. Otherwise, we know that there exists an epsilon transition as in (9) such that $q_A \in \mathsf{LA}(b_i)$, $z = v_R$, and $a \in [t^{\varepsilon}_{q^d,q^s,b_i}]_{\mathbf{I}}$ or $a \in [t^{\varepsilon,k}_{q^d,q^d,b_i}]_{\mathbf{I}}$ for some k. First suppose that $a \in [t^{\varepsilon}_{q^d,q^s,b_i}]_{\mathbf{I}}$. Then, there exist $a_j \in D_{(q_j,s_j)}(b_i)$ such that $a \in [t'[q^d,\ldots,q^d,q^s,\ldots,q^s,a_0,\ldots,a_{r-1},\mathsf{Iset}(b_i),\ldots,\mathsf{Iset}(b_i)]]_{\mathbf{I}}$. Let $c' \notin \mathit{occ}_C(t) \cup \{c\}$ and $c_i = *$ if $z_i = *$, $c_i = c$ if $z_i = v_R$, and $c_i = c'$ if $z_i = v_N$. Define $\sigma'(c') = q^s$ and $\sigma'(c'') = q^d$ for every $c'' \neq c'$. Note that $\sigma'|_{\mathit{occ}_C(t)} = \sigma|_{\mathit{occ}_C(t)}$ and $\sigma'(c) = q^d$. Using the induction hypothesis on i, it is easy to verify that there exist $t'_0,\ldots,t'_{r-1}$ such that $q_j(c_j,t) \vdash^* t'_j$, $a_j \in [\sigma'(t'_j)]_{\mathbf{I}}$, and the t'_j are chosen in such a way that new constants generated in the computation $q_j(c_j,t) \vdash^* t'_j$ are different from c, c', the constants occurring in t, and those that are generated in $q_{j'}(c_{j'},t) \vdash^* t'_{j'}$ for $j' \neq j$. Note that to establish the existence of the t'_j with the above properties, we can in fact use σ' as the permitted substitution for every j. Thus, we have $q(c,t) \vdash^* t'[c,\ldots,c,c',\ldots,c',t'_0,\ldots,t'_{r-1},t,\ldots,t] := t''$ and $a \in [\sigma'(t'')]_{\mathbf{I}}$, which means that a belongs to the right-hand side of the identity in 8. The case where $a \in [t^{\varepsilon,k}_{q^d,q^d,b_i}]_{\mathbf{I}}$ for some k can be dealt with analogously. This concludes the proof of the base case. What we have basically shown here is that if the inclusions hold for some state b_0, then they hold for the epsilon closure of this state. The fact that t is an anonymous constant was only used to show the inclusions for b_0.

Induction step. Assume that $t = f(t_0,\ldots,t_{n-1})$ and that the inclusions hold true for the subterms t_i of t. Let b_i be the unique element with $b_i \in [\sigma(t_i)]_{\mathbf{P}}$. One first shows the inclusions for b' as defined in the definition of transitions of $\mathbf{P}$. This can be done along the same lines as above. From the base case we know that the inclusions stay true when taking the epsilon closure of a state. Thus, they hold true for $b = \overline{b'}$.

"$\supseteq$": We prove 4.–8. simultaneously by induction on the length of computations. For computations of length zero nothing is to show since in this case the sets on the right-hand side are empty. In the

induction step, we again concentrate on 8. The other cases can be shown analogously. Assume that $\mathsf{seen}(b) = \mathsf{no}$ and let $c \notin \mathit{occ}_C(t)$. For every $t'' \in T_{\Sigma^C}$, $a \in Q_I$, and permitted subsitution σ' such that $\sigma'|_{\mathit{occ}_C(t)} = \sigma|_{\mathit{occ}_C(t)}$, $\sigma'(c) = q^d$, $a \in [\sigma'(t'')]_{\mathbf{I}}$, and $q(c,t) \vdash^* t''$ we need to show that $a \in S_{(q,q^d)}(b)$. We distinguish two cases depending on whether the first transition T applied in $q(c,t) \vdash^* t''$ is a Σ- or epsilon transition.

Σ-transition. Assume that T is a Σ-transition of the form (8) where $z = v_R$. We use $c' \in C \backslash (\mathit{occ}_C(t) \cup \{c\})$ as the new constant generated by T (in case T is generative). We have that $t = f(t_0, \ldots, t_{n-1})$ for some $t_i \in T_{\Sigma^C}$. Let b_i be the uniquely determined element in $[\sigma(t_i)]_{\mathbf{P}} = [\sigma'(t_i)]_{\mathbf{P}}$. Then, we know that b is $\overline{b'}$ (see the paragraph on transitions for the definition of b'). After applying T to $q(c,t)$ we obtain

$$t'[c, \ldots, c, c', \ldots, c', q_0(c_0, t_{j_0}), \ldots, q_{r-1}(c_{r-1}, t_{j_{r-1}}), t_{i_0}, \ldots, t_{i_{l-1}}]$$

where $c_i = *$ if $z_i = *$, $c_i = c$ if $z_i = v_R$, and $c_i = c'$ if $z_i = v_N$. Let $t'_0, \ldots, t'_{r-1}$ be the terms such that $q_i(c_i, t_{j_i}) \vdash^* t'_i$ and

$$t'' = t'[c, \ldots, c, c', \ldots, c', t'_0, \ldots, t'_{r-1}, t_{i_0}, \ldots, t_{i_{l-1}}].$$

There exist $a_i \in Q_I$ such that $a_i \in [\sigma'(t'_i)]_{\mathbf{I}}$ and

$$a \in [\sigma'(t'[c, \ldots, c, c', \ldots, c', a_0, \ldots, a_{r-1}, [\sigma'(t_{i_0})]_{\mathbf{I}}, \ldots, [\sigma'(t_{i_{l-1}})]_{\mathbf{I}}]]_{\mathbf{I}}.$$

By Lemma 15, 1. and since σ' and σ coincide on $\mathit{occ}_C(t)$, we have that $\mathsf{Iset}(b_{i_j}) = [\sigma'(t_{i_j})]_{\mathbf{I}}$. Lemma 15, 2. ensures that $q_A \in \mathsf{LA}(b') = \mathsf{LA}(b) = [t]_{\mathbf{A}}$. We distinguish two cases.

First, assume that $\sigma'(c') = q^s$, and thus, all other anonymous constants are mapped to q^d. It is easy to check that σ' meets the conditions in the sets characterizing $D_{(q_i,s_i)}(b_{j_i})$ w.r.t. t_{j_i}. Thus, the induction hypothesis on the length of computations yields that $a_i \in D_{(q_i,s_i)}(b_{j_i})$. Now, it follows that $a \in [t^{\varepsilon}_{q^d,q^s}]_{\mathbf{I}}$, and thus, $a \in S_{(q,q^d)}(b') \subseteq S_{(q,q^d)}(\overline{b'}) = S_{(q,q^d)}(b)$.

Second, assume that σ' is a permitted substitution such that $\sigma'(c'') = q^d$ for every $c'' \in \mathit{occ}_C(t) \cup \{c, c'\}$. We consider two subcases. First, suppose that there exists k and $c'' \in \mathit{occ}_C(t'_k)$ such that $\sigma'(c'') = q^s$. It follows that $c'' \notin \mathit{occ}_C(t) \cup \{c, c'\}$, and thus, c'' was newly generated in $q_k(c_k, t_{j_k}) \vdash^* t'_k$. Consequently, c'' does not occur in t'_i for $i \neq k$. It is easy to check that σ' meets the conditions in the sets characterizing $D_{(q_i,s_i)}(b_{j_i})$ w.r.t. t_{j_i} for every $i \neq k$ and $S_{(q_k,s_k)}(b_{j_k})$ w.r.t. t_{j_k}. Now, the induction hypothesis on the length of computations yields that $a_i \in D_{(q_i,s_i)}(b_{j_i})$ for every $i \neq k$ and $a_k \in S_{(q_k,s_k)}(b_{j_k})$. Thus, $a \in [t^{\varepsilon,k}_{q^d,q^s}]_{\mathbf{I}}$. Consequently, $a \in S_{(q,q^d)}(b') \subseteq S_{(q,q^d)}(\overline{b'}) = S_{(q,q^d)}(b)$. If there is no k and $c'' \in \mathit{occ}_C(t'_k)$ such that $\sigma'(c'') = q^s$, then one can similarly show that $a \in [t^{\varepsilon,k}_{q^d,q^s}]_{\mathbf{I}}$ even for every k, and thus, $a \in S_{(q,q^d)}(b)$.

Epsilon transition. This case can be shown very similar to the case for Σ-transitions. Instead of using the definition of transitions of $\mathbf{P}$ we use the definition of epsilon closure and the fact that b is epsilon closed by Lemma 14. □

4 The Tree Transducer-based Protocol Model

In this section we introduce our protocol and intruder model. The basic assumptions of our model coincide with those for decidable models of non-looping protocols: First, we analyze protocols with

respect to a finite number of receive-send actions, and in particular, a finite number of sessions. Second, the intruder is based on the Dolev-Yao intruder. He can derive new messages from known messages by decomposition, decryption, composition, encryption, and hashing. We do not put a bound on the size of messages. As in [2], we assume keys to be atomic messages; in [24, 19, 4] they may be complex messages.

The main difference between the model presented here and models for non-looping protocols is the way receive-send actions are described—instead of single rewrite rules, we use TTACs. These transducers have two important features necessary to model recursive receive-send actions, but missing in models for non-looping protocols: First, they allow to apply a set of rewrite rules recursively to a term. Second, they allow to generate new constants.

We now provide the formal definition of our tree transducer-based model by defining messages, the intruder, protocols, and attacks.

4.1 Messages

The definition of messages we use here is rather standard, except that we allow an infinite number of (anonymous) constants. As mentioned, we assume keys to be atomic.

More precisely, messages are defined as terms over the signature $(\Sigma_{\mathcal{A}}, \mathcal{C})$ with anonymous constants. The set $\mathcal{C}$ is some *countably infinite* set of anonymous constants, which in this paper will be used to model session keys (Section 7.1). The finite signature $\Sigma_{\mathcal{A}}$ is defined relatively to a *finite* set $\mathcal{A}$ of constants, the set of *atomic messages*, which may for instance contain principal names and (long-term) keys. It also contains a subset $\mathcal{K} \subseteq \mathcal{A}$ of public and private keys which is equipped with a bijective mapping $\cdot^{-1}$ assigning to a public (private) key $k \in \mathcal{K}$ its corresponding private (public) key $k^{-1} \in \mathcal{K}$. Now, $\Sigma_{\mathcal{A}}$ denotes the (finite) signature consisting of the constants $\mathcal{A}$, the unary symbols hash_a (*keyed hash*) and enc^s_a (*symmetric encryption*) for every $a \in \mathcal{A}$, enc^a_k (*asymmetric encryption*) for every $k \in \mathcal{K}$, and the binary symbol $\langle\rangle$ (*pairing*). Instead of $\langle\rangle(t, t')$ we write $\langle t, t'\rangle$. We point out that $\mathsf{hash}_a(m)$ shall represent the keyed hash of m under the key a plus m itself. Note that anonymous constants are not allowed as keys (see also Section 4.4 and 8). The set of *messages* over $(\Sigma_{\mathcal{A}}, \mathcal{C})$ is denoted $\mathcal{M} = T_{\Sigma_{\mathcal{A}}}(\mathcal{C})$.

4.2 The Intruder

As in the case of models for non-looping protocols, our intruder model is based on the Dolev-Yao intruder [11]. That is, an intruder has complete control over the network and can derive new messages from his current knowledge by composing, decomposing, encrypting, decrypting, and hashing messages. We do not impose any restrictions on the size of messages.

The (possibly infinite) set of messages $\mathsf{d}(\mathcal{S})$ the intruder can derive from some set $\mathcal{S} \subseteq \mathcal{M}$ is the smallest set satisfying the following conditions:

1. $\mathcal{S} \subseteq \mathsf{d}(\mathcal{S})$;
2. if $\langle m, m'\rangle \in \mathsf{d}(\mathcal{S})$, then $m, m' \in \mathsf{d}(\mathcal{S})$ (decomposition);
3. if $\mathsf{enc}^s_a(m) \in \mathsf{d}(\mathcal{S})$ and $a \in \mathsf{d}(\mathcal{S})$, then $m \in \mathsf{d}(\mathcal{S})$ (symmetric decryption);
4. if $\mathsf{enc}^a_k(m) \in \mathsf{d}(\mathcal{S})$ and $k^{-1} \in \mathsf{d}(\mathcal{S})$, then $m \in \mathsf{d}(\mathcal{S})$ (asymmetric decryption);

5. if $\mathsf{hash}_a(m) \in \mathsf{d}(\mathcal{S})$, then $m \in \mathsf{d}(\mathcal{S})$ (obtaining hashed messages);

6. if $m, m' \in \mathsf{d}(\mathcal{S})$, then $\langle m, m' \rangle \in \mathsf{d}(\mathcal{S})$ (composition);

7. if $m \in \mathsf{d}(\mathcal{S})$ and $a \in \mathcal{A} \cap \mathsf{d}(\mathcal{S})$, then $\mathsf{enc}^s_a(m) \in \mathsf{d}(\mathcal{S})$ (symmetric encryption);

8. if $m \in \mathsf{d}(\mathcal{S})$ and $k \in \mathcal{K} \cap \mathsf{d}(\mathcal{S})$, then $\mathsf{enc}^a_k(m) \in \mathsf{d}(\mathcal{S})$ (asymmetric encryption);

9. if $m \in \mathsf{d}(\mathcal{S})$ and $a \in \mathcal{A} \cap \mathsf{d}(\mathcal{S})$, then $\mathsf{hash}_a(m) \in \mathsf{d}(\mathcal{S})$ (keyed hash).

Let $\mathsf{an}(\mathcal{S})$ denote the closure of $\mathcal{S}$ under 2.–5., and $\mathsf{syn}(\mathcal{S})$ the closure of $\mathcal{S}$ under 5.–9.

It is well-known that $\mathsf{d}(\mathcal{S})$ can be obtained by first applying an to $\mathcal{S}$ and to the result apply syn. This is because we employ atomic keys; for complex keys this does not hold (see, e.g., [22]):

Lemma 16 *For every $\mathcal{S} \subseteq \mathcal{M}$: $\mathsf{d}(\mathcal{S}) = \mathsf{syn}(\mathsf{an}(\mathcal{S}))$.*

We note that although principals have the ability to generate new (anonymous) constants, as they are defined in terms of TTACs, for the intruder adding this ability is not necessary since it would *not* increase his power to attack protocols (see also Section 4.4).

4.3 Protocols

Protocols are described by sets of principals and every principal is defined by a sequence of receive-send actions, which in a protocol run are performed one after the other. Every receive-send action is specified by a certain TTAC, which we call message transducer.

Definition 17 *A* message transducer $\mathbf{T}$ *is a TTAC over* $(\Sigma_{\mathcal{A}}, \mathcal{C})$.

Roughly speaking, a principal is defined as a sequence of message transducers.

Definition 18 *A* (TTAC-based) principal Π *is a tuple* $((\mathbf{T_0}, \ldots, \mathbf{T_{n-1}}), \mathcal{I})$ *consisting of a sequence* $(\mathbf{T_0}, \ldots, \mathbf{T_{n-1}})$ *of message transducers and an n-ary relation $\mathcal{I} \subseteq I_0 \times \cdots \times I_{n-1}$ where I_i denotes the set of initial states of $\mathbf{T_i}$.*

The single message transducers $\mathbf{T_i}$ in the definition of Π are called *receive-send actions*. In a protocol run, Π performs the receive-send actions one after the other. More precisely, at the beginning of a protocol run, a tuple $(q_0, \ldots, q_{n-1}) \in \mathcal{I}$ is chosen non-deterministically where q_i will be the initial state of $\mathbf{T_i}$ in the current run. Now, if in the protocol run the first message Π receives is m_0, then Π returns some message m_0' with $(m_0, m_0') \in \tau_{\mathbf{T_0}(q_0)}$. Then, on receiving the second message, say m_1, Π returns m_1' with $(m_1, m_1') \in \tau_{\mathbf{T_1}(q_1)}$, and so on. By fixing the initial states at the beginning, we model that Π can convey (a finite amount of) information from one receive-send action to another. For example, if q_0 encodes that Π expects to talk to Bob, then q_1 might describe that in the second message Π expects to see Bob's name again.

We implicitly assume that the last receive-send action of a principal is marked "yes" or "no", indicating whether or not this receive-send action is considerd to be a *challenge output action*. The use of these actions will further be explained in Section 4.4.

A protocol is defined as a finite family of principals plus, since we are interested in attacks on this protocol, the initial intruder knowledge.

Definition 19 *A* (TTAC-based) protocol P *is a tuple* $(\{\Pi_i\}_{i<n}, \mathcal{S})$ *where*

- $\{\Pi_i\}_{i<n}$ *is a family of* n *(TTAC-based) principals, and*
- $\mathcal{S} \subseteq \mathcal{M}$ *is a finite set, the* initial intruder knowledge.

4.4 Attacks on Protocols

In an attack on a protocol, the intruder, who has complete control over the communication, interleaves the receive-send actions of the principals in some way (i.e., determines a total ordering on the receive-send actions), and tries to produce inputs for the principals such that from the corresponding outputs and his initial knowledge he can derive some secret, i.e., some message not supposed to fall into the hands of the intruder. Such a secret can for example be a session key or some secret message. Thus, one can check whether a protocol preserves *secrecy*. One can also check *authentication*. In this case, the secret may be some auxiliary message indicating that a principal completed a session with an instance of another principal which does not exist in the specified protocol model. Now, if the intruder gets to see the secret message, this means that the authentication property is violated.

In the definition of attacks we make use of challenge output actions. Recall that an action is called challenge output action if it is the last receive-send action of a principal and marked to be a challenge output action. In the interleaving of receive-send actions determined by the intruder, we require that the last receive-send action (and only this action) is a challenge output action. This action determines the secret the intruder tries to derive. That is, the output of this action is *not* added to the intruder's knowledge but it is presented to him as a challenge, i.e., a message to be derived.

The use of challenge output actions allows to determine secrets dynamically, depending on the protocol run. This is for example needed when asking whether the intruder is able to derive a session key (an anonymous constant, which may change from one protocol run to another) generated by a key distribution server. Alternatively and equivalently (to dispense with challenge output actions), one could ask whether the intruder can derive an a priori fixed atomic message, say secret, which is encrypted by an anonymous constant (the session key): The encrypted secret can be derived by the intruder iff the intruder knows the anonymous constant used to encrypt secret. However, since in general we do not allow anonymous constants as keys (see Section 4.1 and the conclusion), we find the use of challenge output actions more elegant than introducing special kinds of messages with anonymous constants as keys. Moreover, challenge output actions are somewhat related to the way security is defined in computational models for key distribution protocols where at the end of an attack, the intruder is presented a string for which he needs to decide whether it is an actual session key or just some random string [3].

We remark that the way attacks are defined here allows to ask whether the intruder can derive a message that belongs to some pre-defined *regular* tree language. In models for non-looping procotols this is usually not possible.

The third condition in the following definition ensures that new anonymous constants generated in one receive-send action are also new w.r.t. the knowledge of the intruder before this action is performed.

Definition 20 *Let* $P = (\{\Pi_i\}_{i<n}, \mathcal{S})$ *be a protocol with* $\Pi_i = ((\mathbf{T^i_0}, \ldots, \mathbf{T^i_{n_i-1}}), \mathcal{I}_i)$ *and* $\mathcal{I}_i \subseteq I^i_0 \times \cdots \times I^i_{n_i-1}$ *for* $i < n$*. An* attack *on* P *is a tuple consisting of the following components:*

- *a total ordering $<$, the interleaving of the receive-send actions, on a subset O of $\{(i,j) \mid i < n, j < n_i\}$ such that (i) $(i,j) \in O$ implies $(i,j') \in O$ for every $j' < j$, (ii) $(i,j) < (i,j')$ implies $j < j'$, and (iii) if $(i,j) \in O$ is the greatest element in O (w.r.t. $<$), then $\mathbf{T^i_j}$ is a challenge output action and all previous receive-send actions are not challenge output actions.*
- *a mapping ψ assigning to every $(i,j) \in O$ a tuple $\psi(i,j) = (q^i_j, m^i_j, m'^i_j)$*

such that

1. *for every i, if $(i,j) \in O$ is maximal, i.e., there does not exist a j' with $j < j'$ and $(i,j') \in O$, then there exist $q^i_{j+1}, \ldots, q^i_{n_i-1}$ such that $(q^i_0, \ldots, q^i_{n_i-1}) \in \mathcal{I}_i$,*
2. *$m^i_j, m'^i_j \in \mathcal{M}$ and $(m^i_j, m'^i_j) \in \tau_{\mathbf{T^i_j}(q^i_j)}$ for every $(i,j) \in O$,*
3. *$(occ_{\mathcal{C}}(m'^i_j) \setminus occ_{\mathcal{C}}(m^i_j)) \cap occ_{\mathcal{C}}(S^i_j) = \emptyset$ for every $(i,j) \in O$, and*
4. *$m^i_j \in \mathsf{d}(S^i_j)$ for every $(i,j) \in O$*

where $S^i_j = \mathcal{S} \cup \{m'^{i'}_{j'} \mid (i',j') < (i,j)\}$ is the current intruder knowledge before performing the receive-send action in step (i,j).

An attack is called successful *if the last receive-send action, the challenge output action, say with index $(i,j) \in O$, returns some c such that $c \in \mathsf{d}(S^i_j) \cap (\mathcal{A} \cup \mathcal{C})$.*

The decision problem we are interested in is:

ATTACK: Given a protocol P, decide whether there exists a successful attack on P.

If there is no successful attack on a protocol, we say that the protocol is *secure*.

As mentioned above, extending the intruder by allowing him to generate new constants does not increase his ability to attack protocols. The following remark makes this more precise.

Remark 21 *If the initial intruder knowledge contains at least one anonymous constant, then there exists an attack on a protocol P iff there exists an attack on P in which the intruder may generate new anonymous constants.*

The reason for this is that TTACs cannot check anonymous constants for *dis*equality. If (m, m') belongs to the transduction of a TTAC, then so does $(\sigma(m), \sigma(m'))$ where σ maps anonymous constants in m to some arbitrary constant $c' \notin occ_C(m') \setminus occ_C(m)$, i.e., some constant not newly generated. As a consequence, to attack a protocol the intruder can always use the same (old) anonymous constant instead of creating new once.

Since in models for non-looping protocols *dis*equality tests between messages are usually not possible as well (see, e.g., [24, 19]), in these models extending the intruder with the ability to generate new constants would also not increase his power to attack protocols.

5 The Decidability Result

The main result of this section is the following:

Theorem 22 *For TTAC-based protocols,* ATTACK *is decidable.*

To prove this theorem it obviously suffices to show that the following problem is decidable.

INTERLEAVINGATTACK. *Given a finite set $\mathcal{S} \subseteq \mathcal{M}$ (the initial intruder knowledge), a sequence $\mathbf{T_0}, \ldots, \mathbf{T_{l-1}}$ of message transducers (the interleaving of receive-send actions) with $\mathbf{T_i} = (Q_i, I_i, \mathbf{A_i}, \Gamma_i)$ for $i < l$, decide whether there exist messages $m_i, m_i' \in \mathcal{M}$, $i < l$, such that*

1. *$(m_i, m_i') \in \tau_{\mathbf{T_i}}$ for every $i < l$,*
2. *$(occ_{\mathcal{C}}(m_i') \setminus occ_{\mathcal{C}}(m_i)) \cap occ_{\mathcal{C}}(\mathcal{S}_i) = \emptyset$ for every $i < l$,*
3. *$m_i \in \mathsf{d}(\mathcal{S}_i)$ for every $i < l$, and*
4. *$m_{l-1}' \in \mathsf{d}(\mathcal{S}_{l-1}) \cap (\mathcal{A} \cup \mathcal{C})$*

where $\mathcal{S}_i = \mathcal{S} \cup \{m_0', \ldots, m_{i-1}'\}$ is the intruder's knowledge before the ith receive-send action is performed.

We write $(\mathcal{S}, \mathbf{T_0}, \ldots, \mathbf{T_{l-1}}) \in$ INTERLEAVINGATTACK if all the above conditions are satisfied.

The proof proceeds in two steps. First, we show that the intruder can be simulated by a TTAC. Then, we reduce INTERLEAVINGATTACK to ITERATEDPREIMAGE.

5.1 Derive is TTAC realizable

We wish to show that the messages in $\mathsf{d}(\{m\})$ for some message m can be produced by a TTAC. More precisely, we will construct a TTAC $\mathbf{T}_{der}$ such that $\tau_{\mathbf{T}_{der}}(m) = \mathsf{d}(\{m\})$ for every message m.

We first define what we call the key discovery automaton which is used as look-ahead in $\mathbf{T}_{der}$.

5.1.1 Key Discovery

The key discovery automaton $\mathbf{D}$ is a complete and deterministic WTAAC containing all information about which keys can be accessed in a given message. More precisely, the set of states Q_D of the key discovery automaton are the functions $2^{\mathcal{A}} \to 2^{\mathcal{A}}$ and the automaton is set up in a way such that $[m]_D(K) = \mathsf{an}(\{m\} \cup K) \cap \mathcal{A}$ for every $K \subseteq \mathcal{A}$ and message m. Note that since $\mathbf{D}$ is complete and deterministic, $[m]_D = \{\varphi\}$ for some function $\varphi \in Q_D$. In the following, it is argued that this is indeed possible, that is, we will construct $\mathbf{D}$ with the desired property.

The default state of $\mathbf{D}$ is the identity mapping. The transitions of $\mathbf{D}$ are defined as follows. For every $a \in \mathcal{A}$, $\mathbf{D}$ contains a transition

$$a \to d$$

where $d(K) = K \cup \{a\}$ for every $K \subseteq \mathcal{A}$.

For every $a \in \mathcal{A}$ and $d, d' \in Q_D$, $\mathbf{D}$ contains a transition

$$\mathsf{enc}_a^s(d') \to d$$

iff $d(K) = K$ if $a \notin K$ and $d(K) = d'(K)$ otherwise, for every $K \subseteq \mathcal{A}$.

For every $k \in \mathcal{K}$ and $d, d' \in Q_D$, $\mathbf{D}$ contains a transition

$$\mathsf{enc}_k^a(d') \to d$$

iff $d(K) = K$ if $k^{-1} \notin K$ and $d(K) = d'(K)$ otherwise, for every $K \subseteq \mathcal{A}$.

For every $a \in \mathcal{A}$ and $d, d' \in Q_D$, $\mathbf{D}$ contains a transition

$$\mathsf{hash}_a(d') \to d$$

iff $d(K) = d'(K)$.

The transitions for $\langle \cdot, \cdot \rangle$ are more complicated. For every $d, d', d'' \in Q_D$, $\mathbf{D}$ contains a transition

$$\langle d', d'' \rangle \to d$$

iff for every $K \subseteq \mathcal{A}$, $d(K)$ is the smallest set such that

- $K \subseteq d(K)$,
- if $K' \subseteq d(K)$, then $d'(K') \subseteq d(K)$,
- if $K' \subseteq d(K)$, then $d''(K') \subseteq d(K)$.

The following lemma is easy to prove.

Lemma 23 *For every message $m \in \mathcal{M}$ and $K \subseteq \mathcal{A}$ we have that*

$$[m]_{\mathbf{D}}(K) = \mathsf{an}(\{m\} \cup K).$$

5.1.2 The Transducer $\mathbf{T}_{der}$

The TTAC $\mathbf{T}_{der}$ has a distinguished initial state q_I and, for each $K \subseteq \mathcal{A}$, there are two states (q_S, K) and (q_A, K), the indices being reminiscent of "syn" and "an". The transducer works in three phases. The first phase is just one step and simply determines the keys that can be discovered from the given message m. In the second phase, the "syn part" is carried out, that is, non-deterministically a message m_S is constructed which can be written as $t[m, m, m, \ldots, m]$ where $t(x_0, \ldots, x_{r-1})$ is a linear term which is built using messages from $\mathsf{an}(\{m\})$ only. In the third phase, the "an phase", every copy of m in $t[m, \ldots, m]$ is (non-deterministically) replaced by some message from $\mathsf{an}(\{m\})$.

To be more precise, we have the following transitions in $\mathbf{T}_{der}$. Since for $\mathbf{T}_{der}$ no register is used, in what follows we write $s(t)$ instead of $s(*, t)$ where s is a state of $\mathbf{T}_{der}$, i.e., $s = q_I$, $s = (q_S, K)$, or $s = (q_A, K)$ for some $K \subseteq \mathcal{A}$.

For the first phase, for every $d \in Q_D$, $\mathbf{T}_{der}$ contains the transition

$$q_I(x) \to^d (q_S, d(\emptyset))(x)$$

For the second phase, for every $K \subseteq \mathcal{A}$, $\mathbf{T}_{der}$ contains the following transitions:

$$
\begin{array}{rcll}
(q_S,K)(x) & \to & \langle (q_S,K)(x),(q_S,K)(x)\rangle & \\
(q_S,K)(x) & \to & \mathsf{enc}^s_a((q_S,K)(x)) & \text{for } a \in K \\
(q_S,K)(x) & \to & \mathsf{enc}^a_k((q_S,K)(x)) & \text{for } k \in K \cap \mathcal{K} \\
(q_S,K)(x) & \to & \mathsf{hash}_a((q_S,K)(x)) & \text{for } a \in K \\
(q_S,K)(x) & \to & (q_A,K)(x) &
\end{array}
$$

For the third phase, for every $K \subseteq \mathcal{A}$, $\mathbf{T}_{der}$ contains the following transitions:

$$
\begin{array}{rcll}
(q_A,K)(x) & \to & x & \\
(q_A,K)(x) & \to & a & \text{for } a \in K \\
(q_A,K)(\langle x_0,x_1\rangle) & \to & (q_A,K)(x_0) & \\
(q_A,K)(\langle x_0,x_1\rangle) & \to & (q_A,K)(x_1) & \\
(q_A,K)(\mathsf{enc}^s_a(x)) & \to & (q_A,K)(x) & \text{for } a \in K \\
(q_A,K)(\mathsf{enc}^a_k(x)) & \to & (q_A,K)(x) & \text{for } k^{-1} \in K \cap \mathcal{K} \\
(q_A,K)(\mathsf{hash}_a(x)) & \to & (q_A,K)(x) & \text{for } a \in \mathcal{A}
\end{array}
$$

Writing τ_{der} instead of $\tau_{\mathbf{T}_{der}}$, we can state:

Lemma 24 *$\tau_{der}(m) = \mathsf{d}(\{m\})$ for every $m \in \mathcal{M}$.*

Note that even if we allowed the intruder to generate anonymous constants, we could model such an intruder by a TTACs since TTACs can generate anonymous constants. More precisely, one could simulate the intruder by a composition of two TTACs: The first TTAC copies the input into the output and adds an arbitrary number of new anonymous constants to the output. This can be achieved by using epsilon transitions. The second transducers works just as the transducer described above. Note that this transducer obtains the original message together with the constants generated by the first transducer as input. However, as stated in Remark 21, since the intruder is not more powerful if he can generate anonymous constants, it suffices to model the simpler intruder.

5.2 Reduction to the Iterated Pre-image Word Problem

In the reduction, we describe INTERLEAVINGATTACK as a composition of transducers. We need to introduce two variants of $\mathbf{T}_{der}$ and one variant of $\mathbf{T}_i$, mainly to pass on the intruder's knowledge from one transducer to the next.

The first variant of $\mathbf{T}_{der}$, called $\mathbf{T}^{copy}_{der}$, copies its input to the first component of a pair and simulates $\mathbf{T}_{der}$ on the second component, i.e.,

$$\tau_{\mathbf{T}^{copy}_{der}} = \{(m,\langle m,m'\rangle) \mid m' \in \mathsf{d}(\{m\})\}.$$

The modification of $\mathbf{T}_{der}$ that accomplishes this is very simple. It gets one more state, say q_I^{copy}, which is the inital state of $\mathbf{T}^{copy}_{der}$, and we add the following transition:

$$q_I^{copy}(*,x) \to \langle x, q_I(*,x)\rangle.$$

Recall that q_I is the initial state of $\mathbf{T}_{der}$. Let $\tau^{copy}_{der} = \tau_{\mathbf{T}^{copy}_{der}}$.

The second variant, called $\mathbf{T}^{chall}_{der}$, expects an input of the form $\langle m,m'\rangle$, copies the *second* component into the output and simulates $\mathbf{T}_{der}$ on the *first* component. We call this transducer the *challenge*

transducer since it receives in the second component the challenge and tries to derive it from the first component, the intruder's knowledge. The modifiction of $\mathbf{T}_{der}$ that accomplishes this is again very simple. It gets one more state, say q_I^{chall}, which is the inital state of $\mathbf{T}_{der}^{chall}$, and we add the following transition:

$$q_I^{chall}(*, \langle x_0, x_1 \rangle) \to \langle q_I(*, x_0), x_1 \rangle.$$

Let $\tau_{der}^{chall} = \tau_{\mathbf{T}_{der}^{copy}}$. Finally, we introduce a variant $\hat{\mathbf{T}}_\mathbf{i}$ of $\mathbf{T}_i$ to i) pass on the intruder's knowledge and ii) to satisfy condition 2. in the definition of INTERLEAVINGATTACK. To this end, $\hat{\mathbf{T}}_\mathbf{i}$ only accepts pairs as input, copies the first component into the output (this component stands for the intruder's knowledge) and simulates $\mathbf{T}_i$ on the second component (this component corresponds to the input for $\mathbf{T}_i$). Obviously, $\hat{\mathbf{T}}_i$ accomplishes i) but also ii) since by definition of computations, anonymous constants generated by a transducer are different from those that occur in the input. It is again straighforward to obtain $\hat{\mathbf{T}}_\mathbf{i}$ from $\mathbf{T}_i$: We add one state q to the set of states of $\mathbf{T}_i$, which is the new initial state, and for every initial state q_I of $\mathbf{T}_i$, we add the transition

$$q(*, \langle x_0, x_1 \rangle) \to \langle x_0, q_I(*, x_1) \rangle.$$

Let $\hat{\tau}_i = \tau_{\hat{\mathbf{T}}_\mathbf{i}}$.

We also need the tree language

$$R = \{\langle a, a \rangle \mid a \in \mathcal{A}\} \cup \{\langle c, c \rangle \mid c \in \mathcal{C}\}.$$

which using Example 1 can easily be seen to be TAAC recognizable. For a finite set $S = \{u_0, \ldots, u_{n-1}\}$ of messages let m_S be the message $\langle u_0, \langle u_1, \langle \cdots \langle u_{n-2}, u_{n-1} \rangle \cdots \rangle \rangle \rangle$ (the order of the $u_i's$ does not matter). Finally, let

$$\tau = \tau_{der}^{chall} \circ \hat{\tau}_{l-1} \circ \tau_{der}^{copy} \circ \hat{\tau}_{l-2} \circ \tau_{der}^{copy} \circ \cdots \circ \tau_{der}^{copy} \circ \hat{\tau}_0 \circ \tau_{der}^{copy}.$$

Then, we obtain the following characterization for INTERLEAVINGATTACK.

Lemma 25 *For every $\mathcal{S}$ and $\mathbf{T_0}, \ldots, \mathbf{T_{l-1}}$ as in the definition of* INTERLEAVINGATTACK*, we have*

$$(\mathcal{S}, \mathbf{T_0}, \ldots, \mathbf{T_{l-1}}) \in \text{INTERLEAVINGATTACK } \textit{iff } m_\mathcal{S} \in \tau^{-1}(R).$$

Together with Corollary 10 this immediately implies

Theorem 26 INTERLEAVINGATTACK *is decidable.*

This concludes the proof of Theorem 22.

By reduction from the intersection problem for top-down tree automata, which is known to be EXPTIME-complete [25], it is easy to see that ATTACK and INTERLEAVINGATTACK are EXPTIME-hard. Our decision procedure for the iterated pre-image word problem is non-elementary. Thus, it remains to find a tight complexity bound.

6 Adding Features of Models for Non-looping Protocols and Undecidability Results

As mentioned in Section 4, the basic assumptions of our tree transducer-based protocol model and models for non-looping protocols coincide (finite number of receive-send actions, Dolev-Yao intruder without a bound on the size of messages). In fact, in the TTAC-based protocol model as introduced in Section 4, many non-looping protocols can be analyzed with the same precision as in decidable models for non-looping protocols with atomic keys (see, e.g., [2]). More precisely, this is the case for protocols where a) the receive-send actions can be described by rewrite rules with linear left-hand side, since TTACs can simulate *all* such rewrite rules, and b) only a finite amount of information needs to be conveyed from one receive-send action to the next. This includes for instance many of the protocols in the Clark-Jacobs library [9]. (To illustrate this, in Section 7.2 we provide a formal TTAC-based model of the Needham-Schroeder Public Key Protocol.)

However, some features present in decidable models for non-looping protocols are missing in the TTAC-based protocol model:

1. Equality tests for messages of arbitrary size, which are possible when left-hand sides of rewrite rules may be non-linear (this corresponds to allowing non-linear left-hand sides in transitions of TTACs) or arbitrary messages can be conveyed from one receive-send action to another and can then be compared with other messages [2, 24, 19, 4];

2. complex keys, i.e., keys that may be arbitrary messages [24, 19, 4]; and

3. relaxing the free term algebra assumption by adding the XOR operator [7, 10] or Diffie-Hellman Exponentiation [8].

The main result of this section is that these features cannot be added without losing decidability.

Our undecidability results show that if *one* equality test can be performed than ATTACK is undecidable. While in 1. the equality test is explicitly present, in 2. and 3. implicit equality tests are possible. In the following subsections, the undecidability results are presented in detail.

We remark that when an intruder is allowed to use an *unbounded* number of copies of a principal to perform an attack, i.e., the protocol is analyzed w.r.t. an unbounded number of sessions—and thus, receive-send actions—, then ATTACK is undecidable as well. This is not surprising. The same is true for models of non-looping protocols (see, e.g., [2]).

6.1 Equality Tests on Messages

We define the following extension of TTACs. A *top-down tree transducer with non-linear left-hand side* (TTNL) is a TTAC with transitions of the form as defined in (5) but where t is not required to be linear. A protocol where the receive-send actions are defined by TTNLs is called a *TTNL-based protocol*.

We show the following:

Theorem 27 *For TTNL-based protocols,* ATTACK *and* INTERLEAVINGATTACK *are undecidable.*

The proof of the theorem is by reduction from Post's Correspondence Problem (PCP), which is well-known to be undecidable. The proof for for ATTACK and INTERLEAVINGATTACK is essentially the same.

An instance of PCP is defined as follows: Given an alphabet Σ with at least two letters and two sequences $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_n$ of words over the alphabet Σ (including the empty word ε), decide whether there exist indices $i_0, \ldots, i_{k-1}$, $k > 0$, such that $\alpha_{i_0} \ldots \alpha_{i_{k-1}} = \beta_{i_0} \ldots \beta_{i_{k-1}}$.

We will encode a word $\alpha = a_0 \ldots a_{l-1} \in \Sigma^*$ with $a_i \in \Sigma$ by the term $t_\alpha := \langle a_0, \langle a_1, \ldots, \langle a_{l-1}, \bot\rangle \cdots\rangle$, where $\bot$ is a new constant. We also have to encode the indices $i \in \{1, \ldots, n\}$. Let b be a new constant and let b^i denote the word $b \cdots b$ of length i. Then, i is encoded by $t_i = t_{b^i}$. In addition to b, we will use the new constants b_1, b_2, and secret. The set of atomic messages is defined to be $\mathcal{A} := \Sigma \cup \{b, b_1, b_2, \bot, \mathsf{secret}\}$.

The main idea of the reduction is that the intruder tries to guess a solution of the PCP and then a principal checks whether this is in fact a solution. More specifically, we use one principal who performs four receive-send actions. The initial intruder knowledge is $\mathcal{S} = \Sigma \cup \{b, \bot\}$. The intruder guesses a solution of PCP, i.e., a sequence $i_0, \ldots, i_{k-1}$ of indices encoded as $m_0 = \langle t_{i_0}, \langle t_{i_1}, \ldots, \langle t_{i_{k-1}}, \bot\rangle \cdots\rangle$ and sends m_0 to the principal. In the following steps, it is checked whether m_0 in fact encodes a solution of PCP. This is done as follows. When the principal receives m_0, in her first receive-send action she duplicates m_0 and encrypts it with b_1, i.e., the principal returns $m_0' = \mathsf{enc}^s_{b_1}(\langle m_0, m_0\rangle)$. The encryption is done to prevent the intruder from changing this message. In the second receive-send action, the principal will only accept messages encrypted by b_1. Thus, the intruder must send m_0' to the principal. Now, the principal performs the second receive-send action by reading m_0' and turning the m_0's into words over Σ (encoded as terms) by replacing every index i by the corresponding word α_i, for the m_0 on the left-hand side, and β_i, for the m_0 on the right-hand side. The message, m_1', which the second receive-send action returns, is encrypted by b_2, for the same reason the first message was encrypted by b_1. In the third receive-send action, the principal reads m_1' and checks whether the two words encoded in m_1' are equal. This can be done because the left-hand sides of transitions may be non-linear. If the two words coincide, the secret message secret is given to the intruder. The last receive-send action is a challenge output action which returns secret as output. Thus, the challenge given to the intruder is secret. The input of the challenge output action does not matter. For concreteness, we will assume that the challenge output action expects secret as input. It is easy to see that the intruder is successful iff the instance of the PCP has a solution.

More formally, we now define $\mathbf{T_0}$, $\mathbf{T_1}$, $\mathbf{T_2}$, and $\mathbf{T_3}$, the receive-send actions of the principal informally explained above. Since we do not need registers we simple write $s(t)$ instead of $s(*, t)$. The state q_I will always denote the initial state. The TTNL $\mathbf{T_0}$ is given by one transition:

$$q_I(\langle x, x'\rangle) \to \mathsf{enc}^s_{b_1}(\langle\langle x, x'\rangle, \langle x, x'\rangle\rangle).$$

By writing $\langle x, x'\rangle$ we make sure that the sequence of indices is not empty. The transitions of $\mathbf{T_1}$ are:

$$q_I(\mathsf{enc}^s_{b_1}(\langle x, x'\rangle)) \to \mathsf{enc}^s_{b_2}(\langle q_\alpha(x), q_\beta(x')\rangle)$$

and for every $i \in \{1, \ldots, n\}$ with $\alpha_i = a_0 \cdots a_{l-1}$:

$$\begin{aligned} q_\alpha(\langle t_i, x\rangle) &\to \langle a_0, \langle a_1, \ldots, \langle a_{l-1}, q_\alpha(x)\rangle \cdots\rangle \\ q_\alpha(\bot) &\to \bot. \end{aligned}$$

The transitions for the β_i's are defined analogously. The transducer $\mathbf{T_2}$ is given by the transition:

$$q_I(\mathsf{enc}^s_{b_2}(\langle x, x\rangle)) \quad \to \quad \mathsf{secret}.$$

Note that the term on the left-hand side of the transition is non-linear. This is the only place where an equality test is used. Finally, $\mathbf{T_3}$ which is considered a challenge output action has the following transition:

$$q_I(\mathsf{secret}) \to \mathsf{secret}.$$

One easily shows that the instance of the PCP has a solution iff there exists a successful attack on the protocol just described, and that this is the case iff $(\mathcal{S}, \mathbf{T_0}, \mathbf{T_1}, \mathbf{T_2}, \mathbf{T_3}) \in$ INTERLEAVINGATTACK. This concludes the proof of Theorem 27.

We point out that the transducers $\mathbf{T_i}$ used in the reduction are deterministic and that they do not use epsilon transitions or anonymous constants.

Since TTNLs (with epsilon transition) can simulate the intruder, as a consequence of the reduction presented in Section 5.2 we obtain:

Corollary 28 *For TTNLs the pre-image word problem is undecidable.*

The same results hold true for other extensions of TTACs which enable the transducer to perform equality tests. For example:

- TTACs which can store one message of arbitrary size and compare it with the input message for equality, either directly or recursively by comparing the head of the current input with the head of the message stored in memory.
- TTACs which are equipped with a parameter (taking a message) used to remember a message when going from one TTAC to another and which can compare the parameter with another message (see 1. at the beginning of Section 6). Recall that in the model presented in Section 4, only a finite amount of information can be conveyed from one receive-send action to another.
- TTACs where in the right-hand side of transitions the state symbols can occur everywhere in the term instead of only at subterms of the term on the left-hand side.

We note that for the undecidability result of Corollary 28, the existence of epsilon transitions is essential: If epsilon transitions are not allowed, then on a given input, a transducer can only produce a finite number of outputs modulo new anonymous constants. It is easy to bound the number of anonymous constants to be considered. Thus, we obtain:

Observation 29 *For TTACs or TTNLs without epsilon transitions, i.e., only Σ-transitions of the form (8), the pre-image word problem is decidable.*

6.2 Complex Keys

To model complex keys, we replace the unary symbol $\mathsf{enc}^s_a(\cdot)$, by the binary symbol $\mathsf{enc}(\cdot,\cdot)$. The message $\mathsf{enc}(m, m')$ with $m, m' \in \mathcal{M}$ stands for the message m' encrypted by m. Note that the key m may be a complex message.

Accordingly, we extend the intruder's ability to derive messages. If the intruder knows $m, m' \in \mathcal{M}$, then he can generate $\mathsf{enc}(m, m')$. If he knows $\mathsf{enc}(m, m')$ and m, then he knows m' as well.

The transducers used to define principals are not extended, except that the signature changes.

To see that in this setting ATTACK and INTERLEAVINGATTACK are undecidable it suffices to observe that in the reduction from Section 6.1, the transition defining $\mathbf{T_2}$ can be replaced by

$$q_I(\mathsf{enc}(b_2, \langle x, x' \rangle)) \ \rightarrow \ \langle \mathsf{enc}(x, \mathsf{secret}), x' \rangle.$$

This ensures that the intruder can get hold of secret iff the messages substituted for x and x' coincide. In other words, the reduction uses that decryption for complex keys requires equality tests for messages of arbitrary size. We have shown:

Theorem 30 *For TTAC-protocols with complex keys,* ATTACK *and* INTERLEAVINGATTACK *are undecidable.*

This result is also true if the challenge may be an aribtrary message. More precisely, the setting is as follows. The message space is defined as in Section 4. So far we require that in a challenge output action the principal returns a challenge in $\mathcal{C} \cup \mathcal{A}$. Now we drop this requirement and allow any message from $\mathcal{M}$ as challenge. It is easy to see that then ATTACK and INTERLEAVINGATTACK are undecidable:

In the above reduction we define the transition of $\mathbf{T_2}$ to be

$$q_I(\mathsf{enc}^s_{b_2}(\langle x, x' \rangle)) \ \rightarrow \ \langle \mathsf{enc}^s_{b_3}(x), \mathsf{enc}^s_{b_4}(x') \rangle$$

and the one for $\mathbf{T_3}$ to be

$$q_I(\mathsf{enc}^s_{b_4}(x)) \ \rightarrow \ \mathsf{enc}^s_{b_3}(x).$$

Thus, the challenge for the intruder is $\mathsf{enc}^s_{b_3}(x')$ which he can meet only if the messages substituted for x and x' in the transition of $\mathbf{T_2}$ coincide. This shows:

Theorem 31 *The problems* ATTACK *and* INTERLEAVINGATTACK *are undecidable in the TTAC-based protocol model in case challenges may be arbitrary messages (instead of constants from $\mathcal{C} \cup \mathcal{A}$).*

6.3 XOR and Diffie-Hellman Exponentiation

We first consider XOR. The message space is extended as follows: We add the constant 0 and the binary symbol $\oplus$ which among others has the following algebraic property: $m \oplus m = 0$ (see, e.g., [7] for other properties of XOR.) These properties induce an equivalence relation on messages. For instance, $\mathsf{enc}^s_a(m \oplus m) \equiv \mathsf{enc}^s_a(0)$. Note that this gives a way to compare messages for equality.

In general, one would extend the intruder by the ability to combine messages using the XOR operator. For the undecidability result this is however not needed.

Also, one would require the transducers to work on equivalence classes of messages. However, it is easy to see that for the reduction this does not make a difference.

To show undecidability, we can again use a similar reduction from PCP as in Section 6.1. The only difference is the definition of $\mathbf{T_2}$ and $\mathbf{T_3}$ and that we need another transducer $\mathbf{T_4}$.

The transducer $\mathbf{T_2}$ is given by the following transition:

$$q_I(\mathsf{enc}^s_{b_2}(\langle x, x' \rangle)) \ \rightarrow \ \mathsf{enc}^s_{b_3}(x \oplus x').$$

Thus, the intruder obtains $\mathsf{enc}^s_{b_3}(0)$ iff the messages substituted for x and x' coincide.

Now, $\mathbf{T_3}$ checks whether the intruder knows $\mathsf{enc}^s_{b_3}(0)$ and in this case returns secret. That is, $\mathbf{T_3}$ is defined by the following transition:

$$q_I(\mathsf{enc}^s_{b_3}(0)) \;\rightarrow\; \mathsf{secret}.$$

Finally, $\mathbf{T_4}$ is defined just as $\mathbf{T_3}$ in Section 6.1.

A similar reduction is possible for Diffie-Hellman Exponentiation [8] since the normalization also involves comparison of arbitrary messages. We obtain:

Theorem 32 *For TTAC-protocols with XOR or Diffie-Hellman Exponentiation, the problems* ATTACK *and* INTERLEAVINGATTACK *are undecidable.*

7 Modeling Cryptographic Protocols

In this section, we present formal TTAC-based protocols models for the recursive authentication protocol (as an example of a recursive protocol) and the Needham Schroeder Public Key Protocol (as an example of a non-looping protocol).

7.1 The Recursive Authentication Protocol

In Section 7.1.1, we first give an informal description of the recursive authentication protocol (RA protocol). Section 7.1.2 provides a formal TTAC-based model for this protocol. In what follows, we abbreviate messages of the form $\langle m_0, \ldots, \langle m_{n-1}, m_n\rangle \cdots\rangle$ by $m_0 \cdots m_n$ or $m_0, \ldots, m_n$.

7.1.1 Informal Description of the RA Protocol

The RA protocol was proposed by Bull and Otway [6] and it extends the authentication protocol by Otway and Rees [21] in that it allows to establish session keys between an a priori unbounded number of principals in one protocol run. Our description of the RA protocol follows Paulson [22].

In the RA protocol one assumes that a key distribution server S shares long-term keys with the principals. In Figure 1 a typical protocol run is depicted. In this run, A wants to establish a session key with B and B wants to establish a session key with C. The number of principals involved in a protocol run is not bounded. In particular, C could send a message to some principal D in order to establish a session key with D and D could continue and send a message to E and so on. In the protocol run depicted in Figure 1, we assume that C does not want to talk to another principal and therefore sends a message to the key distribution server S, who is involved in every protocol run.

In Figure 1, K_a denotes the long-term key shared between A and S. Similarly, K_b and K_c are the long-term keys shared between B, C, and S, respectively. With N_a, N_b, and N_c we denote nonces (i.e., random numbers) generated by A, B, and C, respectively. Finally, K_{ab}, K_{bc}, and K_{cs} are the session keys generated by the server and used by the principals for secure communication between A and B, B and C, and C and S, respectively. The numbers (1. – 6.) attached to the messages only indicate the order in which the messages are sent and do not belong to the protocol.

We now take a closer look at the messages exchanged between the principals in the order they are sent: In the first messages (1.), principal A indicates that she requests a session key from the server

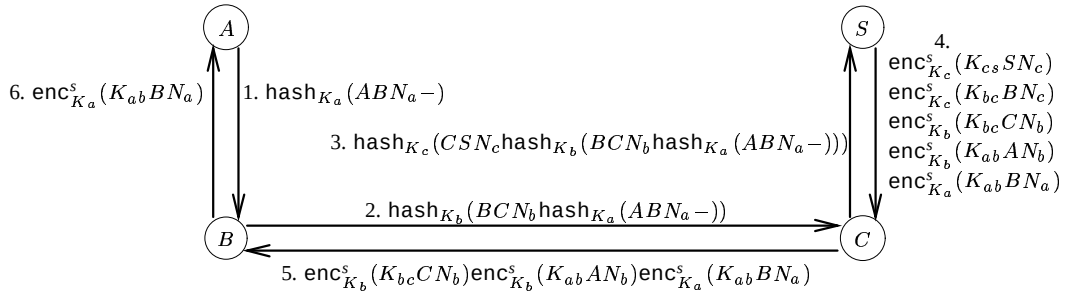

Figure 1: A Run of the Recursive Authentication Protocol

for secure communication with B. The symbol "$-$" says that this message started the protocol run. Now, in the second message (2.), B sends something similar to C but with A's message instead of "$-$", indicating that he wants to share a session key with C. As mentioned, this step could be repeated as many times as desired, yielding an ever-growing stack of requests. The process is terminated if one principal contacts S. In our example, we assume that C does not request another session key, and therefore, sends the message received from B to S (3.). This message is now processed by S. This can be done in different ways. In what follows, we describe one possible way.

First, S checks whether the outer request is in fact addressed to S. If so, S generates a new session key and stores it. Now, S processes the requests starting from the outermost. In general, S has a "frame" containing two requests at a time. In the example, S starts with a frame containing the requests CSN_c and BCN_b. Thus, S knows that C wants to talk to S and that B wants to talk to C. Consequently, S has to generate two certificates for C, one that contains the session key for communication with S and the other one for communication with B. These certificates are generated by S as follows. The first one contains the session key stored, the name S of the server, and C's nonce N_c. For the second certificate, S generates a new session key, stores it for later use, and then assembles the second certificate for C containing the session key just generated, B's name, and C's nonce N_c. At this point, all certificates for C have been prepared. Therefore, S moves the frame one request further and processes this frame as before. Note that now the frame contains the requests BCN_b and ABN_a, and that for the first certificate sent to B, S uses the session key stored. After the two certificates for B have been prepared, S moves the frame one request further. Now this frame contains only one request, namely, ABN_a-. The marker "$-$" indicates that A started the protocol. Therefore, only one certificate for A is generated. It contains the session key stored, B's name, and A's nonce N_a. After this, S has prepared all certificates and sends them back to the principal who called S. In the example this is C.

Principal C accepts the first two certificates, extracts the two session keys, and forwards the rest of the message to his predecessor in the chain (5.). Then, B does the same, and forwards the last certificate to A (6.) Since according to the intruder model, the message send by S is sent to the intruder, we may assume that every principal only receives his or her certificates and does not need to forward the rest of the message to his or her predecessor.

7.1.2 The TTAC-based Protocol Model

We now provide a formal description of the RA protocol in the TTAC-based protocol model.

First we note that although in the RA protocol the number of receive-send actions performed in

one protocol run is unbounded, in our model we assume a fixed bound—extending the TTAC-based protocol model to handle an unbounded number of receive-send actions would lead to undecidability (Section 6). Nevertheless, even with such a fixed bound it is still necessary to model recursive processes: In the RA protocol, the intruder can generate an unbounded sequence of requests which must be processed by the server. In other protocols in which the number of receive-send action in a protocol run is fixed, recursive processes may also occur independently of the intruder. One example is IKE [14].

In what follows, let $P_0, \ldots, P_n$ be the principals participating in the RA protocol. We assume that $P_n = S$ is the server. Every P_i, $i < n$, shares a long-term key K_i with S. The nonce sent by P_i in the request message is denoted N_i, $i < n$.

7.1.3 Modeling the Agents

An agent P_i, $i < n$, performs two receive-send actions and is given by the tuple $(\mathbf{T_0^i}, \mathbf{T_1^i}, \mathcal{I}^i)$. The different components are defined next.

The message transducer $\mathbf{T_0^i}$ for sending the request message consists of the following transitions:

$$\begin{aligned} (\mathsf{request}, \bot, P_{j'})(*, init) &\rightarrow \mathsf{hash}_{K_i}(P_i, P_{j'}, N_i, -), \\ (\mathsf{request}, P_j, P_{j'})(*, \mathsf{hash}_a(P_j, P_i, x_0, x_1)) &\rightarrow \mathsf{hash}_{K_i}(P_i, P_{j'}, N_i, \mathsf{hash}_a(P_j, P_i, x_0, x_1)) \end{aligned}$$

where x_0 and x_1 are variables, $j' \leq n$, $j < n$, $a \in \mathcal{A}$, and $init \in \mathcal{A}$ is some atomic message known to the intruder. The first transition is applied if P_i initiates a protocol run and calls $P_{j'}$. The second transition is applied if P_i is called by P_j and sends a message to $P_{j'}$. The initial states of $\mathbf{T_0^i}$ are $(\mathsf{request}, \bot, P_{j'})$ and $(\mathsf{request}, P_j, P_{j'})$ for every $j' \leq n$ and $j < n$.

The transducer $\mathbf{T_1^i}$ is a challenge output action which receives a session key and sends it out to the intruder as a challenge. For every $j' \leq n$ and $j < n$, $\mathbf{T_1^i}$ contains the following transitions:

$$\begin{aligned} (\mathsf{key}, \bot, P_{j'})(*, \mathsf{enc}^s_{K_i}(x_0, P_{j'}, N_i)) &\rightarrow x_0 \\ (\mathsf{key}, P_j, P_{j'})(*, \langle \mathsf{enc}^s_{K_i}(x_0, P_{j'}, N_i), \mathsf{enc}^s_{K_i}(x_1, P_j, N_i)\rangle) &\rightarrow x_0 \\ (\mathsf{key}, P_j, P_{j'})(*, \langle \mathsf{enc}^s_{K_i}(x_0, P_{j'}, N_i), \mathsf{enc}^s_{K_i}(x_1, P_j, N_i)\rangle) &\rightarrow x_1 \end{aligned}$$

where x_0 and x_1 are variables. The first transition is applied if P_i initiated the protocol run for communication with $P_{j'}$. The other two transitions are applied if P_i was called by P_j and called $P_{j'}$. All states occurring in $\mathbf{T_1^i}$ are initial states.

It remains to define $\mathcal{I}^i$. We want to guarantee that P_i remembers who he called and who wants to communicate with P_i. Therefore, we set

$$\begin{aligned} \mathcal{I}^i := \; & \{((\mathsf{request}, \bot, P_{j'}), (\mathsf{key}, \bot, P_{j'})) \mid j' \leq n\} \cup \\ & \{((\mathsf{request}, P_j, P_{j'}), (\mathsf{key}, P_j, P_{j'})) \mid j' \leq n, j < n\}. \end{aligned}$$

This model of the agents is more precise than the one presented in [15] where word transducers have been used instead of tree transducers. While in [15], the nonces (the messages substituted for x_0 in $\mathbf{T_0^i}$) needed to be typed since a word transducer can not parse arbitrary message, here any message can be substituted for x_0.

7.1.4 Modeling the Server

Since the server $S = P_n$ performs only one receive-send action, it can be described by a single message transducer, which we call $\mathbf{T_n}$.

The transducer $\mathbf{T_n}$ has two states and works as described in Section 7.1.1. In state start, the initial state, $\mathbf{T_n}$ checks whether the first request is addressed to S and generates a session key which is stored in the register. In state read, the requests are processed. In this phase, the register is used to store a session key while moving the frame to the next request.

The transitions of $\mathbf{T_n}$ are specified as follows:

$$\begin{aligned}
\mathsf{start}(*, \mathsf{hash}_{K_i}(P_i, P_n, x_0, x_1)) &\rightarrow \mathsf{read}(v_N, \mathsf{hash}_{K_i}(P_i, P_n, x_0, x_1)) \\
\mathsf{read}(v_R, \mathsf{hash}_{K_i}(P_i, P_j, x_0, -)) &\rightarrow \mathsf{enc}^s_{K_i}(v_R, P_j, x_0) \\
\mathsf{read}(v_R, \mathsf{hash}_{K_i}(P_i, P_j, x_0, \mathsf{hash}_{K_{i'}}(P_{i'}, P_i, x_1, x_2))) &\rightarrow \mathsf{enc}^s_{K_i}(v_R, P_j, x_0), \mathsf{enc}^s_{K_i}(v_N, P_{i'}, x_0), \\
&\quad \mathsf{read}(v_N, \mathsf{hash}_{K_{i'}}(P_{i'}, P_i, x_1, x_2))
\end{aligned}$$

where $i, i', j \leq n$ and x_0, x_1, x_2 are variables which take arbitrary messages, and v_R and v_N are the variables for the register and the new anonymous constants, respectively.

This model of the server is more precise than the one presented in [15]. First, we do not need to assume that nonces are typed. The server accepts any message as nonce. In the word transducer model this was not possible since i) word transducers cannot parse arbitrarily nested messages and ii) they cannot copy messages of arbitrary size, which is however necessary in the last transition of the server. Second, TTACs allow to generate anonymous constants, and thus, provide a very natural way of modeling the generation of new session keys. The word transducers as considered in [15] did not have this capability. Therefore, in [15], the server could only choose from a finite set of session keys. Since the number of session keys the server needs to generate is not fixed a priori, this was only an approximation of the server's actual behavior.

It is clear that with decidable models for non-looping protocols [24, 19, 4, 2] the server cannot be modeled faitfully since these models do not allow to describe recursive processes.

7.2 The Needham Schroeder Public Key Protocol

The Needham Schroeder Public Key Protocol is a famous public key challenge response protocol (see, e.g., [9] for a more detailed description). In our terminology it is a non-looping protocol since its receive-send actions do not require iteration or recursion.

In the standard Alice and Bob notation the protocol can be described as follows where K_A and K_B denote A's and B's public key, respecitvely, and N_A and N_B denote nonces generated by A and B, respectively:

$$\begin{aligned}
A \rightarrow B&: \ \mathsf{enc}^a_{K_B}(N_A, A) \\
B \rightarrow A&: \ \mathsf{enc}^a_{K_A}(N_A, N_B) \\
A \rightarrow B&: \ \mathsf{enc}^a_{K_B}(N_B) \\
B \rightarrow \ &: \ N_B
\end{aligned}$$

The last action of B is a challenge output action. That is, B presents N_B as a challenge for the intruder since N_B may be used as session key.

We model the protocol as follows: We assume that A runs one instance of the protocol as initiator with the intruder I. We also model one instance of B running in the role of a responder with A.

All receive-send actions can be modeled by TTACs with only one state, which we call start, and one transition.

Principal A performs two receive-send actions, and thus, is described by two TTACs, $\mathbf{T_0^A}$ and $\mathbf{T_1^A}$, with the following transitions:

$$\begin{array}{lrcl} \mathbf{T_0^A}\colon & \mathsf{start}(*, init) & \to & \mathsf{enc}^s_{K_I}(N_A, A) \\ \mathbf{T_1^A}\colon & \mathsf{start}(*, \mathsf{enc}^s_{K_A}(N_A, x)) & \to & \mathsf{enc}^s_{K_I}(x) \end{array}$$

Principal B performs two receive-send actions as well, described by $\mathbf{T_0^B}$ and $\mathbf{T_1^B}$:

$$\begin{array}{lrcl} \mathbf{T_0^B}\colon & \mathsf{start}(*, \mathsf{enc}^s_{K_B}(x, A)) & \to & \mathsf{enc}^s_{K_A}(x, N_B) \\ \mathbf{T_1^B}\colon & \mathsf{start}(*, \mathsf{enc}^s_{K_B}(N_B)) & \to & N_B \end{array}$$

It is easy to see that there is an attack on this protocol, which was first found by Lowe [16]. In particular, this attack can automatically be found by our decision algorithm.

We point out that nonces are not required to be typed. The principals accept any message as nonce. In fact, the formulation of the Needham-Schroeder Protocol as described here is as accurate as other formulations based on models for non-looping protocols [24, 19, 4, 2]. As above, in [15] one would have to assume that nonces are typed.

8 Conclusion

The main goal of this paper was to shed light on the feasibility of automatic analysis of recursive cryptographic protocols. The results obtained here trace a fairly tight boundary of the decidability of security for such protocols. To obtain our results we introduced tree automata (TAACs) and transducers (TTACs) over signatures with an infinite set of (anonymous) constants and proved that for TTACs the iterated pre-image word problem is decidable. We believe that the study of TAACs and TTACs started here is of independent interest.

One open problem is to establish tight complexity bounds for our decidability results. While so far we do not allow anonymous constants as keys, this would be an interesting extension of our model. In this paper, we have identified the computation of pre-images as a means to analyze protocols. It is worthwhile to investigate to what extent this method is practical and whether it could be an altnerative to constraint solving approaches usually employed for the analysis of (non-looping) protocols.

References

[1] N. Alon, T. Milo, F. Neven, D. Suciu, and V. Vianu. XML with Data Values: Typechecking Revisited. *To appear in JCSS*, 2003.

[2] R.M. Amadio, D. Lugiez, and V. Vanackere. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1):695–740, 2002.

[3] M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing (STOC'95)*, pages 57–66. ACM, 1995.

[4] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Automata, Languages and Programming, 28th International Colloquium (ICALP 2001)*, volume 2076 of *Lecture Notes in Computer Science*, pages 667–681. Springer-Verlag, 2001.

[5] J. Bryans and S.A. Schneider. CSP, PVS, and a Recursive Authentication Protocol. In *DIMACS Workshop on Formal Verification of Security Protocols*, 1997.

[6] J.A. Bull and D.J. Otway. The authentication protocol. Technical Report DRA/CIS3/PROJ/CORBA/SC/1/CSM/436-04/03, Defence Research Agency, Malvern, UK, 1997.

[7] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP Decision Procedure for Protocol Insecurity with XOR. In *Proceedings of the Eighteenth Annual IEEE Symposium on Logic in Computer Science (LICS 2003)*, pages 261–270. IEEE, Computer Society Press, 2003.

[8] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In *Proc. of FSTTCS 2003*, 2003. To appear.

[9] J. Clark and J. Jacob. *A Survey of Authentication Protocol Literature*, 1997. Web Draft Version 1.0 available from http://citeseer.nj.nec.com/.

[10] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Proceedings of the Eighteenth Annual IEEE Symposium on Logic in Computer Science (LICS 2003)*, pages 271–280. IEEE, Computer Society Press, 2003.

[11] D. Dolev and A.C. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[12] J. Engelfriet. Three hierarchies of transducers. *Mathematical Systems Theory*, 15:95–125, 1982.

[13] N. Ferguson and B. Schneier. A Cryptographic Evaluation of IPsec. Technical report, 2000. Available from http://www.counterpane.com/ipsec.pdf.

[14] D. Harkins and D. Carrel. *The Internet Key Exchange (IKE)*, November 1998. RFC 2409.

[15] R. Küsters. On the decidability of cryptographic protocols with open-ended data structures. In L. Brim, P. Jancar, M. Kretinsky, and A. Kucera, editors, *13th International Conference on Concurrency Theory (CONCUR 2002)*, volume 2421 of *Lecture Notes in Computer Science*, pages 515–530. Springer-Verlag, 2002.

[16] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.

[17] C. Meadows. Extending formal cryptographic protocol analysis techniques for group protocols and low-level cryptographic primitives. In P. Degano, editor, *Proceedings of the First Workshop on Issues in the Theory of Security (WITS'00)*, pages 87–92, 2000.

[18] C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In *Proceedings of DISCEX 2000*, pages 237–250. IEEE Computer Society Press, 2000.

[19] J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 166–175. ACM Press, 2001.

[20] F. Neven, Th. Schwentick, and V. Vianu. Towards regular languages over infinite alphabets. In *Mathematical Foundations of Computer Science (MFCS 2001)*, volume 2136 of *Lecture Notes in Computer Science*, pages 560–572. Springer, 2001.

[21] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, January 1987.

[22] L.C. Paulson. Mechanized Proofs for a Recursive Authentication Protocol. In *10th IEEE Computer Security Foundations Workshop (CSFW-10)*, pages 84–95, 1997.

[23] O. Pereira and J.-J. Quisquater. A Security Analysis of the Cliques Protocols Suites. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 73–81, 2001.

[24] M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 174–190, 2001.

[25] H. Seidl. Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52(2), 1994.

[26] J. Zhou. Fixing a security flaw in IKE protocols. *Electronic Letter*, 35(13):1072–1073, 1999.