Available online at www.sciencedirect.com
ScienceDirect
Information and Computation 205 (2007) 1741–1776
Information and Computation
www.elsevier.com/locate/ic

# Transducer-based analysis of cryptographic protocols☆

Ralf Küsters [a,*], Thomas Wilke [b]

[a] ETH Zurich, Institute of Theoretical Computer Science, IFW E 48.3, Haldeneggsteig 4, CH-8092 Zurich, Switzerland
[b] Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany

Received 14 March 2006; revised 31 July 2007
Available online 14 September 2007

**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, by 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 much is known on the decidability of security for non-looping protocols, only little is known for recursive protocols. In this paper, we prove decidability of security (with respect to the standard Dolev–Yao intruder) for a core class of recursive protocols and undecidability for several extensions. The key ingredient of our protocol model is 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.


*Keywords:* Cryptographic protocols; Automatic analysis; Decidability; Transducers

## 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 [34,2,5,26] 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 $\sigma$ 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 with respect

☆ An extended abstract of this paper appeared in STACS 2004 (Küsters and Wilke, 2004). This work was partially supported by the "Deutsche Forschungsgemeinschaft (DFG)" under Grant KU 1434/4-1.
* Corresponding author.
*E-mail addresses:* ralf.kuesters@inf.ethz.ch (R. Küsters), wilke@ti.informatik.uni-kiel.de (T. Wilke).

0890-5401/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2007.08.003

to 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 [34,2,5,26], complex keys are allowed [34,5,26], and the free term algebra assumption is relaxed by algebraic properties of various operators, such as exclusive or (XOR), Diffie-Hellman exponentiation, and RSA encryption [8,13,9,36,10].

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*.

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 [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 session 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 [25] for a description of some recursive protocols. As pointed out by [25] and illustrated in [39,17], 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 (TTAC's) with look-ahead and $\varepsilon$-transitions which work with a signature containing an *infinite* set of what we call *anonymous constants* (AC's), over which the TTAC's has only very limited control. TTAC's can generate new (anonymous) constants, a feature often needed to model recursive receive–send actions; in the RA protocol, for instance, the key distribution server needs to generate (an a priori unbounded number of) session keys.

The main result of this paper is that (1) 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 TTAC's (Section 5), and that (2) certain features of models for non-looping protocols cannot be added without losing decidability: As soon as TTAC's are equipped with the ability to perform equality tests between arbitrary messages, as soon as complex keys are allowed, or as soon as 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 TTAC's 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 TTAC's, which we show to be decidable (Section 3). Given a term $t$, a "regular set" $R$ of terms, and a sequence of TTAC's, the *iterated pre-image word problem* asks whether on input $t$ the composition of the TTAC's can produce an output in $R$. Here, "regular set" means a set of terms recognizable by a new kind of tree automata, *tree automata over signatures with anonymous constants* (TAAC's), which can compare anonymous constants for equality.

### 1.1. Related work

Recursive protocols, such as the RA protocol and the A-GDH.2 protocol [3], have been analyzed manually [33] and semi-automatically using theorem provers or special purpose tools [32,6,24].

Decidability for recursive protocols has initially been investigated in [20]. However, there are significant differences to the present paper. First, in [20] *word* transducers are employed, which are less powerful than tree transducers. As a result, tree transducers provide a 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 [20]. 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 [20], a quite involved and technical pumping argument is 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 a more elegant proof, and anonymous constants present a completely new challenge.

Recently, Truderung [38] proposed an alternative model for recursive protocols in which he shows that security is NEXPTIME-complete. Instead of transducers, he uses what he calls selecting theories, which are certain classes of Horn clauses, to model recursive receive–send actions. The expressivity of these theories is orthogonal to the expressivity of our transducers. On the one hand, selecting theories can only output sequences of simply structured messages, while transducers can produce much more complex messages. For instance, transducers can produce output of size unrelated to the size of the input. In Truderung's model the output size is linear in the input size. Also, new constants cannot be generated, and hence, session key generation can only be approximated. On the other hand, selecting theories allow to test arbitrary messages for equality. The proof techniques applied by Truderung are very different from the ones employed in the present paper. While we, as explained, use automata-theoretic techniques, the techniques employed by Truderung are closer to those for non-looping protocols. In [21], Truderung's model has been extended to include the exclusive OR (XOR) and decidability and undecidability results have been shown in this extended model.

In various papers, automata-theoretic techniques have been applied to the analysis of cryptographic protocols (see, e.g., [27,18,19,12]). However, these works aim at analyzing non-looping protocols with respect to an unbounded number of sessions and do not seem to be applicable to recursive protocols in an obvious way. To the best of our knowledge, the work in [20] and the present work are the first to employ transducers (over infinite signatures) for protocol analysis. Automata and transducers over infinite signatures (although quite different from those considered here) have been studied in the context of type checking and type inference for XML queries with data values (see, e.g., [1,29]).

### 1.2. Structure of the paper

In Section 2, we introduce the mentioned tree automata (TAAC's) and transducers (TTAC's) 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 TTAC's. 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 (1) to briefly discuss the relationship between our model and models for non-looping protocols, and (2) to prove the aforementioned 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.

### 1.3. 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 $\Sigma$ denotes a signature, then $\Sigma_n$ denotes the set of symbols from $\Sigma$ with arity $n$.

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

We fix an infinite supply $X$ of variables among which we find $x_0, x_1, x_2, \ldots$ For $n \geqslant 0$, we write $T^n_\Sigma$ 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 at most 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 $\Sigma$ 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 $\mathbb{N}^*$ we denote the set of finite strings over the non-negative integers $\mathbb{N}$. The empty string is denoted $\varepsilon$. As usual, $v \in \mathbb{N}^*$ is called a *prefix* of $w \in \mathbb{N}^*$ if there exists $v' \in \mathbb{N}^*$ such that $w = vv'$ where $vv'$ denotes the concatenation of $v$ and $v'$. A set $S \subseteq \mathbb{N}^*$ 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 $S \subseteq \mathbb{N}^*$ into $\Sigma$ such that if $t(\pi) \in \Sigma_n$ for some $n \geqslant 0$ and $\pi \in S$, then $\{i \mid \pi i \in S\} = \{0, \ldots, n-1\}$, and if $t(\pi)$ is a variable, then $\{i \mid \pi i \in S\} = \emptyset$. 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 $\pi$, 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 $\tau$ of $T_\Sigma \times T_\Sigma$ is called a *transduction* over $\Sigma$. For a term $t$, we define $\tau(t) = \{t' \mid (t,t') \in \tau\}$. If $\tau$ and $\tau'$ are transductions over $\Sigma$, then their *composition* $\tau \circ \tau'$ defines the transduction $\{(t,t') \mid \exists t''((t,t'') \in \tau' \wedge (t'',t') \in \tau)\}$, i.e., the composition is read from right to left. Given a transduction $\tau$ over $\Sigma$ and a set $R \subseteq T_\Sigma$, the *pre-image* of $R$ under $\tau$ is the set $\tau^{-1}(R) = \{t \mid \exists t' \in R \text{ with } (t,t') \in \tau\}$.

## 2. Tree automata and tree 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. Before defining our tree automata and transducers, we introduce signatures with anonymous constants.

### 2.1. Signatures and anonymous constants

A pair $(\Sigma, C)$ consisting of a finite signature $\Sigma$ and an arbitrary *infinite* set $C$ of constant symbols disjoint from $\Sigma$ is called a *signature with anonymous constants;* the elements of $\Sigma$ and $C$ are referred to as *regular symbols* and *anonymous constants,* respectively. With such a signature, we associate the signature $\Sigma \cup C$, denoted $\Sigma^C$. That is, when we speak of a *term over* $(\Sigma, C)$ we mean a term over $\Sigma^C$. In what follows, let $occ_C(t)$ denote the set of elements from $C$ that occur in the term $t$; similarly, let $occ_C(\mathcal{S})$ denote the set of elements from $C$ that occur in any term of a 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 on a tree $t \in T_{\Sigma^C}$, the automaton first non-deterministically assigns $q^d$ and $q^s$ to the anonymous constants that occur in $t$ in an arbitrary way 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.* Note that different occurrences of the same anonymous constant in $t$ get assigned the same state, $q^d$ or $q^s$. Once these states have been assigned to the anonymous constants, the rest of the run of the automaton on $t$ proceeds in the standard bottom-up fashion.

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

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

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,* $\Delta$ is a finite *set of transitions* as specified below, and $F \subseteq Q$ is a *set of final states.* If the set of final states is omitted, we speak of a *semi TAAC.*

Formally, transitions are pairs of a certain type, but for better reading, we write a transition $(t,q)$ as $t \to q$. 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 *ε-transition* is of the form

$$q \to q'$$

where $q', q \in Q$.

We call a TAAC *deterministic* if it does not contain $\varepsilon$-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 $(\Sigma, C)$ 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 $[t]_{\mathcal{A}}$ of states which the automaton reaches after having read the term $t$.

We first give an inductive definition for terms $t \in T_\Sigma(Q)$ without anonymous constants. In this case, $[t]_{\mathcal{A}}$ is the smallest set satisfying the following rules:

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

A *permitted* substitution $\sigma$ 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

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

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

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

We say that a tree language over $\Sigma^C$ is *TAAC-recognizable* over $(\Sigma, C)$ if it is recognized by some TAAC over $(\Sigma, C)$.

Before we provide some examples of (non) 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

$$[t[S_0, \ldots, S_{n-1}]]_{\mathcal{A}} = \bigcup_{(q_0,\ldots,q_{n-1}) \in S_0 \times \ldots \times S_{n-1}} [t[q_0, \ldots, q_{n-1}]]_{\mathcal{A}}.$$

**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 $(\Sigma, C)$ with anonymous constants. For every $i \leqslant 3$, let $T_i$ be the tree language over $\Sigma^C$ which contains a tree $t$ iff in $t$ at least $i$ pairwise distinct anonymous constants occur. Then, it is easy to see that $T_i$ is TAAC-recognizable over $(\Sigma, C)$ for $i \leqslant 2$, but not for $i = 3$: for $i = 0$, the TAAC should accept every tree. For $i = 1$, one defines a TAAC with a distinguished state $q = q^d = q^s$ to which anonymous constants are mapped. The TAAC checks whether this state occurs in a run on a tree at least once to make sure that the tree contains at least one anonymous constant. For $i = 2$, the TAAC has states $q^d$ and $q^s$ with $q^d \neq q^s$ and checks whether both $q^d$ and $q^s$ occur in a run on a tree. The negative result for $i = 3$ immediately follows from Lemma 3, which in particular implies that if a TAAC accepts $f(c, g(c', c''))$ for anonymous constants $c$, $c'$, $c''$, then it also accepts one of $f(c, g(c'', c''))$, $f(c'', g(c', c''))$, and $f(c', g(c', c''))$.

To state the mentioned lemma, we need the notion of $c$-equivalent terms. Let $t$ and $t'$ be arbitrary terms and $c$ an anonymous constant. Now, $t$ and $t'$ are $c$-equivalent if they coincide except for occurrences of anonymous constants other than $c$, i.e., $t$ and $t'$ are *c-equivalent* if there is a term $u$ and substitutions $\sigma$ and $\sigma'$ with $\sigma(X), \sigma'(X) \subseteq C \setminus \{c\}$ such that $\sigma(u) = t$ and $\sigma'(u) = t'$. For example, $f(c, g(c', c''))$ and $f(c, g(c'', c'''))$ are $c$-equivalent but $f(c, g(c', c''))$ and $f(c', g(c, c''))$ are not.

**Lemma 3.** *Let $T$ be a TAAC recognizable language and $t \in T$. Then there exists an anonymous constant $c$ such that $t' \in T$ for every tree $t'$ which is $c$-equivalent to $t$.*

**Proof** (*sketch*)**.** Let $\mathcal{A}$ be a TAAC recognizing $T$. Take for $c$ the constant which gets assigned $q^s$ in some accepting run of $\mathcal{A}$ on $T$. Now, the claim follows easily. □

Another example of TAAC-recognizable languages is the following class of languages:

**Example 4.** Fix any signature $(\Sigma, C)$ with anonymous constants. For every $i$, let $T_i$ be the tree language over $\Sigma^C$ which contains a tree $t$ iff in $t$ there are at least $i$ occurrences of (not necessarily different) anonymous constants. Then, $T_i$ is TAAC-recognizable over $(\Sigma, C)$ for every $i$. Indeed, $T_i$ is recognized by the TAAC $\mathcal{A} = (\{0, \ldots, i, q\}, q, q, \Delta, \{i\})$ where for every $f \in \Sigma_n$ and states $q_1, \ldots, q_n \in \{0, \ldots, i, q\}$, $\Delta$ contains the transition $f(q_1, \ldots, q_n) \to j$ where $j = \min(i, \#\{l \mid q_l = q\} + \sum_{l \in \{1,\ldots,n\}, q_l \neq q} q_l)$. That is, anonymous constants are assigned to $q$ and $\mathcal{A}$ counts the number of $q$'s up to $i$. If $i$ is reached at the root of the tree, the tree is accepted.

The above TAAC is what we call a weak TAAC. A TAAC is called *weak (WTAAC)* if the default and the selecting state are identical, i.e., $q^d = q^s$. This means that there actually is no selecting state. WTAAC's are really weaker because it is easy to see that, for instance, $T_=$ is not WTAAC-recognizable over $(\Sigma, C)$.

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

**Lemma 5.** *Every TAAC is equivalent to a deterministic TAAC. The same holds true for WTAAC's.*

But observe that for TAAC's, there is still non-determinism involved because of the freedom to choose which anonymous constant is assigned $q^s$.

In the following lemma, we consider closure properties of TAAC's as well as WTAAC's. As we will see, the behavior of TAAC's is quite different from that of tree automata over finite signatures.

**Lemma 6.** *Let $(\Sigma, C)$ be a signature with anonymous constants. Then*:

(1) *The set of tree languages over $(\Sigma, C)$ recognized by WTAAC's over $(\Sigma, C)$ is closed under union, intersection, and complementation.*
(2) *The set of tree languages over $(\Sigma, C)$ recognized by TAAC's over $(\Sigma, C)$ is closed under union.*
(3) *The set of tree languages over $(\Sigma, C)$ recognized by TAAC's over $(\Sigma, C)$ is closed under complementation iff $\Sigma = \Sigma_0 \cup \Sigma_1$. The same holds true for intersection.*

**Proof.** (1) This can be proved just as for bottom-up tree transducers: closure under union follows from closure under intersection and complementation; closure under intersection is shown using a standard product construction; closure under complementation follows from Lemma 5, because complementing a deterministic WTAAC can be achieved by complementing its set of final states.

(2) Let $\mathcal{A}_0 = (Q_0, q_0^d, q_0^s, \Delta_0, F_0)$ and $\mathcal{A}_1 = (Q_1, q_1^d, q_1^s, \Delta_1, F_1)$ be TAAC's. We want to show that $T(\mathcal{A}_0) \cup T(\mathcal{A}_1)$ is recognized by some TAAC. Basically, we construct the product automaton of $\mathcal{A}_0$ and $\mathcal{A}_1$ and use the following equivalence: Given a tree $t$, there exists a permitted substitution $\sigma$ such that $\mathcal{A}_0$ or $\mathcal{A}_1$ accepts $\sigma(t)$ iff there exists a permitted substitution $\sigma_0$ such that $\mathcal{A}_0$ accepts $\sigma(t)$ or there exists a permitted substitution $\sigma_1$ such that $\mathcal{A}_1$ accepts $\sigma_1(t)$.

More precisely, to construct a TAAC that recognizes $T(\mathcal{A}_0) \cup T(\mathcal{A}_1)$, we first make $\mathcal{A}_0$ and $\mathcal{A}_1$ complete as follows: (i) We add new states $s_0$ and $s_1$ to $\mathcal{A}_0$ and $\mathcal{A}_1$, respectively, and ii) for every $f \in \Sigma_n, q_0, \ldots, q_{n-1} \in Q_i \cup \{s_i\}$ we add $f(q_0, \ldots, q_{n-1}) \to s_i$ to the set of transitions of $\mathcal{A}_i$ for $i = 0, 1$. Let's denote the resulting automata by $\mathcal{A}_2$ and $\mathcal{A}_3$, respectively. We clearly have $T(\mathcal{A}_i) = T(\mathcal{A}_{i+2})$ for $i < 2$. In the second step, we construct the following product automaton:

$$(Q_2 \times Q_3, (q_2^d, q_3^d), (q_2^s, q_3^s), \Delta, (F_0 \times Q_3) \cup (Q_2 \times F_1))$$

with $(q_2, q_3) \to (q_2', q_3') \in \Delta$ if $q_2 \to q_2' \in \Delta_2$ and $q_3 = q_3'$, or $q_2 = q_2'$ and $q_3 \to q_3' \in \Delta_3$, and $f((q_2^1, q_3^1), \ldots, (q_2^n, q_3^n)) \to (q, q')$ if $f(q_2^1, \ldots, q_2^n) \to q \in \Delta_2$ and $f(q_3^1, \ldots, q_3^n) \to q' \in \Delta_3$ for $f \in \Sigma_n$. Clearly, the resulting TAAC recognizes $T(\mathcal{A}_2) \cup T(\mathcal{A}_3) = T(\mathcal{A}_0) \cup T(\mathcal{A}_1)$.

(3) First of all, if $\Sigma = \Sigma_0 \cup \Sigma_1$ and $\mathcal{A}$ is a TAAC over $(\Sigma, C)$, then $T(\mathcal{A}) = T((Q, q^d, q^d, \Delta, F)) \cup T((Q, q^s, q^s, \Delta, F))$ since terms over $(\Sigma, C)$ can only contain at most one anonymous constant, and hence, in a run on such a term this constant is either assigned $q^d$ or $q^s$. Both these automata are WTAAC's, so by (1). we know that $T(\mathcal{A})$ is recognized by a WTAAC, and, again by 1, so is its complement. This proves one direction of the implication.

We now show that TAAC's are not closed under complementation and intersection if there is a binary symbol in $\Sigma$, say $f$. It is straightforward to extend this to any signature with at least one symbol of arity $\geqslant 2$. In what follows, let $\#_c(t)$ denote the number of occurrences of $c$ in $t$.

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

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

It is easy to see that $L_n$ is TAAC-recognizable for every $n \geqslant 1$ through a construction similar to Example 4. However, we show that neither $\overline{L_2} = T_{\Sigma^C} \setminus L_2$ nor $L_2 \cap L_3$ are TAAC-recognizable.

By contradiction, assume that $T = T_{\Sigma^C} \setminus L_2$ is recognized by some TAAC $\mathcal{A}$ 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 $c_1$. Let $c$ be a constant such that the assignment $q_s \mapsto c$ yields an accepting run for $t$. We proceed by a case distinction. If $c \notin \{c_0, c_1\}$, then $\mathcal{A}$ has an accepting run on $t$ that does not distinguish between $c_0$ and $c_1$, and hence, $f(f(c_0, c_0), f(c_0, c_1)) \in T$—a contradiction. If $c = c_0$, we have $f(f(c_0, c_1), f(c_0, c_2)) \in T$ for a new constant $c_2$—a contradiction. And if $c = c_1$, we have $f(f(c_0, c_1), f(c_2, c_1)) \in T$ for a new constant $c_2$—again a contradiction.

Now assume that there exists a TAAC $\mathcal{A}$ recognizing $L_2 \cap L_3$. For $c \in C$ and $n \geqslant 2$, let $t_c^n = f(c, f(c, \cdots f(c, c)))$ such that $\#_c(t_c^n) = n$. Let $c_0$ and $c_1$ be two distinct anonymous constants. Obviously, $t = f(f(t_{c_0}^3, t_{c_1}^2), f(t_{c_0}^6, t_{c_1}^4)) \in L_2 \cap L_3$. Just as above, by considering different cases for $c$, one shows that variants of $t$ are recognized by $\mathcal{A}$ although they do not belong to $L_2 \cap L_3$, which leads to a contradiction. □

We finally note that for TAAC's the word and emptiness problem are decidable:

**Lemma 7.** *The word and the emptiness problem are decidable for TAAC's, and thus, WTAAC's.*

**Proof.** 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 on bottom-up tree automata that if a TAAC recognizes a tree then also a tree of depth bounded by the number of states of the TAAC, and, clearly, only two fixed different anonymous constants have to be considered (one for $q^s$ and one for $q^d$). Now, decidability of the emptiness problem easily follows. □

### *2.3. Tree transducers over signatures with anonymous constants*

Tree transducers 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. It has several specific features:

- a WTAAC look-ahead,
- a mechanism for generating new anonymous constants, and
- a register for one anonymous constant.

In addition, our tree transducers may be non-deterministic and may contain $\varepsilon$-transitions.

To define our transducers, we need some more notation. We fix a signature $(\Sigma, C)$ 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 $s \in S, z \in C \cup \{*, v_R, v_N\}$, and $t \in T_\Sigma(C \cup X)$. The term $t$ is called the *core term* of the state term. If $z$ belongs to some set $D \subseteq C \cup \{*, v_R, v_N\}$, 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 $s(c,t)$. We now define our tree transducers and their computation formally, along with examples.

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

$$\boldsymbol{T} = (S, I, \boldsymbol{A}, \Gamma) \tag{2}$$

where $S$ is a finite set of *states,* $I \subseteq S$ is a set of *initial states,* $\boldsymbol{A}$ is a semi WTAAC over $(\Sigma, C)$, the *look-ahead automaton*, and $\Gamma$ is a finite set of transitions as described below.

A *transition* is of the form

$$s(z,t) \to^q t'[v_R, v_N, t'_0, \ldots, t'_{r-1}] \tag{3}$$

where

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

Observe that the condition in (3) is not a real restriction, it is just to make the formalism well defined: if $z = *$, the register is undefined, which means it does not make any sense to use its content ($v_R$) on the right-hand side of the rule, so $v_R$ should not appear in it.

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 $\boldsymbol{A}$ accepts every term. If $\boldsymbol{A}$ does not contain such a state, then $\boldsymbol{A}$ can be extended accordingly. For $s \in S$, we denote by $\boldsymbol{T}(s)$ the transducer $\boldsymbol{T}$ with $s$ as its only initial state.

Before defining the computation of TTAC's formally, let us consider a simple example of a TTAC and its computation (see Example 9 for a more complex setting).

**Example 8.** Let $\Sigma_0 = \{d\}$, $\Sigma_1 = \{f\}$, $\Sigma_2 = \{g\}$, and $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$. We consider a TTAC $\boldsymbol{T}$ over $(\Sigma, C)$ with only one state $s$, no look-ahead, and the following transitions:

$$s(*, f(x)) \to g(s(v_N, x), v_N), \tag{4}$$
$$s(v_R, f(x)) \to g(s(v_R, x), v_R), \tag{5}$$
$$s(v_R, d) \to v_R. \tag{6}$$

Assume that $t = f(f(d))$ is given as input to $\boldsymbol{T}$. Then, the initial configuration of $\boldsymbol{T}$ is $s(*,t)$, which means that $\boldsymbol{T}$ is in state $s$, is about to read $t$, and does not contain an anonymous constant in its register. The only transition that can be applied to this configuration is (4), where $x$ is replaced by $f(d)$ and $v_N$ is replaced by a newly generated constant, say $c$. The result of applying (4) to $s(*,t)$ is the new configuration $g(s(c, f(d)), c)$. Now, (5) can be applied to $s(c, f(d))$ where $v_R$, the register variable, is replaced by $c$ and $x$ by $d$. (In general, the intermediate output produced by a TTAC may contain several state terms and to these state terms transitions are applied independently.) The result of applying (5) to $s(c, f(d))$ is $g(s(c,d), c)$, and the overall output produced by $\boldsymbol{T}$ so far is $g(g(s(c,d), c), c)$. At this point, (6) can be applied to $s(c,d)$ yielding the overall output $g(g(c,c),c)$. This term does not contain a state term and is considered an output of $\boldsymbol{T}$ on input $t$. Note that since $\boldsymbol{T}$ could have generated

other anonymous constants than $c$ in the first step of the computation, all terms of the form $g(g(c', c'), c')$ with $c' \in C$ are possible outputs of $\boldsymbol{T}$.

The fact that TTAC's may produce several different outputs for the same input is not only due to the fact that anonymous constants cannot be distinguished. Even modulo renaming of anonymous constants, given one input term, several outputs can be produced by a TTAC's as TTAC's may be non-deterministic. Also, since TTAC's may contain transitions which do not consume input symbols, the set of outputs may be infinite even modulo renaming of anonymous constants. Consider, for instance, the transition

$$s(v_R, f(x)) \to g(s(v_R, f(x)), v_R). \tag{7}$$

This transition works as (5), but with one difference: it does not consume the input symbol $f$. Adding this transition to $\boldsymbol{T}$ above and given the input term $t$ from above, all terms of the form

$$\underbrace{g(g(\cdots g(c, c)\cdots), c), c)}_{n \text{ occurrences of } g}$$

for $n \geqslant 2$ are possible outputs of $\boldsymbol{T}$.

Formally, the computation a TTAC carries out is described by a sequence of rewriting steps. The corresponding rewrite relation $\vdash_U$ is defined with respect to a subset $U \subseteq C$ of anonymous constants to ensure that whenever the TTAC is supposed to generate a new constant this constant does not belong to $U$. Later $U$ will be the set of anonymous 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 hence, $t \in T_{\Sigma^C}(\{x_0, \ldots, x_{n-1}\})$ (see p. 5), and $u_1$ is linear, and a transition $\tau$ as in (3) with $z = v_R$. Let $\sigma$ be the substitution defined by $\sigma(x_i) = t_i$. Then, if $q \in [u_2]_{\boldsymbol{A}}$ (i.e., the current input satisfies the look-ahead),

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

for every $c' \in C \setminus (occ_C(u_0) \cup U)$. Observe that if $\tau$ 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 rewriting step in case $u_0 = u_1[s(*, u_2)]$ is defined in the same way, but it is required that $z = *$.

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

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

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

We say that a transduction $\tau$ on $(\Sigma, C)$ is *TTAC-realizable* if there exists a TTAC $\boldsymbol{T}$ such that $\tau_{\boldsymbol{T}} = \tau$. We call two TTAC's *equivalent* if they realize the same transduction.

Let us look at another slightly more complex example.

**Example 9.** Let $\Sigma_0 = \{d\}$, $\Sigma_1 = \{f\}$, $\Sigma_2 = \{g\}$ and $C$ be an infinite set of anonymous constants. Consider the transduction $\tau$ on $(\Sigma, C)$ 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 any term of the form $g(f(f(\ldots f(c)\ldots)), f(f(\ldots f(c)\ldots)))$ for a new anonymous constant $c$, where the arguments of $g$ may be of any depth and do not need to be equal, only the anonymous constant $c$ needs to be the same in both arguments of $g$. Subterms of $t$ that contain an anonymous constant are simply copied. For example, a possible transduction of $g(g(d, d), g(g(c', c', ), c'))$ for an anonymous constant $c'$ is $g(g(f(f(f(c))), f(c)), g(g(c', c'), c'))$ where $c$ is a new anonymous constant, in particular $c \neq c'$. We show that $\tau$ is TTAC-realizable.

Let $\boldsymbol{A}$ be the semi WTAAC 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 $q_f \in [t]_{\boldsymbol{A}}$ iff $f$ does not occur in $t$; $q_R \in [t]_{\boldsymbol{A}}$ iff $t$ does not contain $f$ nor anonymous constants; and $q_M \in [t]_{\boldsymbol{A}}$ 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:

$$s_I(*,x_0) \to^{q_f} s_0(*,x_0), \tag{8}$$
$$s_0(*,x_0) \to^{q_C} x_0, \tag{9}$$
$$s_0(*,g(x_0,x_1)) \to^{q_M} g(s_0(*,x_0),s_0(*,x_1)), \tag{10}$$
$$s_0(*,x_0) \to^{q_R} g(s_f(v_N,x_0),s_f(v_N,x_0)), \tag{11}$$
$$s_f(v_R,x_0) \to f(s_f(v_R,x_0)), \tag{12}$$
$$s_f(v_R,x_0) \to v_R. \tag{13}$$

Transition (8) is used to check whether the input term does not contain $f$. Transition (9) is applied if the input term is an anonymous constant. This constant is simply copied into the output term. Transition (10) is applied if the input term starts with $g$ but contains an anonymous constant. Hence, this term is not replaced at this point and the computation proceeds with the arguments of $g$. Transition (11) is applied if the current input term does not contain an anonymous constant, and hence, since the TTAC processes terms top-down, the current input term is a maximal subterm of the original input term without anonymous constants, and hence, is supposed to be replaced by $g(f(\cdots f(c)\cdots), f(\cdots f(c)\cdots))$ for a new anonymous constant $c$, which in (11) is created using $v_N$. The transitions (12) and (13) are used to produce terms of the form $f(\cdots f(c)\cdots)$ where $c$ is the anonymous constant that was generated when transition (11) was applied.

A class $\mathcal{C}$ of transductions is *closed under composition* if for all transductions $\tau$ and $\tau'$ we have that $\tau, \tau' \in \mathcal{C}$ implies $\tau \circ \tau' \in \mathcal{C}$. It is well known [16] 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 TTAC's. One example illustrating this fact is the following: obviously, for $g \in \Sigma_1$, one can define a TTAC $\boldsymbol{T}_1$ that on input $a \in \Sigma_0$ generates $g^n(a) = g(g(\cdots g(a)))$ for every $n \geqslant 0$. Also, it is easy to construct a TTAC $\boldsymbol{T}_2$ that when given a term $t$ as input outputs $f(t,t)$. Now, on input $a$ the transduction $\tau_{\boldsymbol{T}_2} \circ \tau_{\boldsymbol{T}_1}$ produces the tree language $\{f(g^n(a),g^n(a)) \mid n \geqslant 0\}$. By standard pumping arguments, it is easy to prove that no single TTAC can produce such an output on input $a$.

**Lemma 10.** *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 $(\Sigma, C)$, a TAAC $\boldsymbol{B}$ over $(\Sigma, C)$, and a sequence of TTAC's $\boldsymbol{T}_0, \ldots, \boldsymbol{T}_{l-1}$ over $(\Sigma, C)$ with $\tau = \tau_{T_0} \circ \cdots \circ \tau_{T_{l-1}}$, decide whether $t \in \tau^{-1}(T(\boldsymbol{B}))$.*

The key for proving decidability of this problem is:

**Theorem 11.** *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, we immediately obtain:

**Corollary 12.** IteratedPreImage *is decidable.*

**Proof.** Given $t$, $\boldsymbol{B}$, $\boldsymbol{T}_0, \ldots, \boldsymbol{T}_{l-1}$, and $\tau$ as above, by Theorem 11 it follows that a TAAC $\boldsymbol{A}$ recognizing $\tau^{-1}(T(\boldsymbol{B}))$ can be constructed effectively. Since by Lemma 7 the word problem for TAAC's is decidable, we can decide whether $t \in T(\boldsymbol{A})$, and thus, whether $t \in \tau^{-1}(T(\boldsymbol{B}))$. □

The runtime of the decision procedure described in the proof of the corollary may be non-elementary since, as we will see, the number of states of the TAAC constructed in the proof of Theorem 11 may be exponential in the size of the input.

The proof of Theorem 11 is carried out in three steps. We first show how TTAC's can be turned into what we call simple TTAC's (Section 3.1). We then construct a TAAC recognizing the pre-image of a TAAC-recognizable tree language under a simple TTAC (Section 3.2) and finally prove the correctness of this construction (Section 3.3).

### *3.1. Simple TTAC's*

We say that a transition of the form (3) is *simple* if

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

We call a TTAC *simple* if it only contains simple transitions. The main advantage of simple TTACs is that the left-hand side of transitions of simple TTACs are flat, and hence, their application is more local and applications of different transitions do not overlap too much. This makes these transitions easier to handle and therefore helps in the proof of Theorem 11.

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

**Lemma 13.** *For every linear term $t \in T^n_{\Sigma^C}$, there exists a WTAAC $\boldsymbol{A}_t$ over $(\Sigma, C)$ and a state, say $q_t$, in $\boldsymbol{A}_t$ such that $q_t$ is the only final state of $\boldsymbol{A}_t$ and $T(\boldsymbol{A}_t) = \{t' \mid$ there exists a substitution $\sigma$ such that $\sigma(t) = t'\}$.*

**Proof.** The proof can be carried out by a straightforward structural induction on $t$. □

Note that the lemma does not hold for non-linear terms.

We next show:

**Lemma 14.** *Every TTAC is equivalent to a simple TTAC, which can be constructed in polynomial time.*

**Proof.** Let $\boldsymbol{T}$ be a TTAC as in (2) and let $\Gamma$ contain a non-simple transition of the form

$$s(z, t) \to^q t'[v_R, v_N, t'_0, \ldots, t'_{r-1}, x_{i_0}, \ldots, x_{i_{l-1}}] \tag{14}$$

where the $t'_i$ are state terms and the $x_{i_j}$ are variables occurring in $t$.

We explain how (14) can be replaced by several simple transitions; by iterating this argument all non-simple transitions of $\boldsymbol{T}$ can be replaced by simple transitions.

The idea is as follows: we first use an $\varepsilon$-transition to check the look-ahead $q$ and nothing else. Then we extend $\boldsymbol{A}$ by $\boldsymbol{A}_t$ (Lemma 13) to be able to check whether the input term matches with $t$. This is done in an $\varepsilon$-transition with $q_t$ as look-ahead. Next, we take care of the most difficult problem, namely eliminating occurrences of subtrees of $t$ on the right-hand side. We simply encode the positions of these subtrees in appropriate states. Finally, we add transitions for the states introduced in the previous step, which make sure that the subtrees of trees they encode are processed in the right way.

In the first step, we simply add the $\varepsilon$-transition

$$s(z,x) \to^{q} s'(z,x), \tag{15}$$

where $s'$ is a new state. This transition does not change the input, but it can only be taken if the input satisfies the look-ahead condition.

In the second step, we check that the input term matches $t$. Therefore, we add the states and the transitions of $\boldsymbol{A}_t$, assuming disjoint state sets. In addition, we add the $\varepsilon$-transition

$$s'(z,x) \to^{q_t} s''(z,x), \tag{16}$$

where $s''$ is another new state and $q_t$ is as in Lemma 13. This transition does not change the input either, but it can only be taken if the input matches $t$.

In the third step, we encode references to subterms of $t$ in new states. To this end, we add states of the form $p_\pi^q$ and $p_\pi$ for $\pi \in \mathcal{P}(t)$ and $q \in S$. We will later add transitions in such a way that these states satisfy the following properties:

(1) $p_\pi(z,t'') \vdash_U^* t'''$ iff $t''|_\pi = t'''$, and, similarly,
(2) $p_\pi^q(z,t'') \vdash_U^* t'''$ iff $q(z,t''|_\pi) \vdash_U^* t'''$,

for any choice of terms $t''$ and $t'''$, any choice of $U \subseteq C$, and any choice of $z$. Given these properties, it is now easy to add the right transition. We assume that each $t_i'$ is of the form $s_i(z_i,t_i'')$ and for each $i$ we pick a position $\pi_i$ such that $t_i'' = t|_{\pi_i}$. (If there are several positions that satisfy this condition, any one will be good.) We also pick for every $j$ a position $\pi_j'$ such that $x_{i_j} = t|_{\pi_j'}$. Now we can add the following $\varepsilon$-transition:

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

In the fourth step, we add the transitions that are needed to guarantee (1) and (2) from above. First, for every $f \in \Sigma_n$, $q \in Q$, $i < n$, $z \in \{v_R, *\}$, and $\pi$ with $i\pi \in \mathcal{P}(t)$ we add

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

where, by convention, $p_\varepsilon^q = q$. Similarly, for every $f \in \Sigma_n$, $i < n$, and $\pi$ with $i\pi \in \mathcal{P}(t)$ we add

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

where, by convention, $p_\pi(*,x_i)$ is replaced by $x_i$ in case $\pi = \varepsilon$.

It can now easily be verified that the new TTAC is equivalent to the original one, by induction on the length of a computation. □

### 3.2. Construction of the TAAC recognizing the pre-image

Given a TAAC $\boldsymbol{I}$ (the image automaton) and a TTAC $\boldsymbol{T}$, we construct a TAAC $\boldsymbol{P}$ (the pre-image automaton) such that $T(\boldsymbol{P}) = \tau_{\boldsymbol{T}}^{-1}(T(\boldsymbol{I}))$. In Section 3.3, we then prove that our construction is correct.

In what follows, let

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

be a TAAC over $(\Sigma, C)$ and

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

be a TTAC over $(\Sigma, C)$ with

$$\boldsymbol{A} = (Q_A, q_A^d, q_A^d, \Delta_A)$$

as its look-ahead automaton. As mentioned, we want to construct a TAAC

$$\boldsymbol{P} = (Q_P, q_P^d, q_P^s, \Delta_P, F_P)$$

over $(\Sigma, C)$ such that

$$T(\boldsymbol{P}) = \tau_{\boldsymbol{T}}^{-1}(T(\boldsymbol{I})).$$

Due to Lemma 14, we may assume that $\boldsymbol{T}$ is simple. Thus, we may assume that $\Gamma_T$ consists of $\Sigma$-transitions of the form

$$q(z, f(x_0, \ldots, x_{n-1})) \to^{q_A} t'[v_R, \ldots, v_R, v_N, \ldots, v_N, t'_0, \ldots, t'_{r-1}, x_{i_0}, \ldots, x_{i_{l-1}}] \tag{17}$$

where $t'$ is linear, $i_j \in \{0, \ldots, n-1\}$ for every $j < l$, $t'_i$ is a $\{v_R, *\}$-state term of the form $q_i(z_i, x_{j_i})$ with $j_i \in \{0, \ldots, n-1\}$, and $v_R$ may only occur on the right-hand side of (17) if $z = v_R$, and $\varepsilon$-transitions of the form

$$q(z, x) \to^{q_A} t'[v_R, \ldots, v_R, v_N, \ldots, v_N, t'_0, \ldots, t'_{r-1}, x, \ldots, x] \tag{18}$$

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 (18) if $z = v_R$. Note that assuming $t'$ to be linear is w.l.o.g. since we can duplicate the entries $v_R$, $v_N$, $t'_i$, $x$, and $x_{i_j}$.

Roughly speaking, the idea behind the construction of $\boldsymbol{P}$ is that in a run of $\boldsymbol{P}$ on $t \in T_{\Sigma^C}$, $\boldsymbol{P}$ simulates the runs of $\boldsymbol{I}$ on all possible outputs $t'$ of $\boldsymbol{T}$ on input $t$ simultaneously. The problem is that runs of $\boldsymbol{I}$ are required to satisfy a global condition, namely that the default and the selecting states $q^d$ and $q^s$ of $\boldsymbol{I}$ are assigned to constants in a consistent way—by a permitted substitution. To capture this global condition in $\boldsymbol{P}$, we use that runs of $\boldsymbol{P}$ also meet such a global condition. More precisely, the permitted substitution in a run of $\boldsymbol{P}$ on $t$ will determine the permitted substitutions that are considered in the runs of $\boldsymbol{I}$ on the trees $t'$ in the following way:

(1) ("Yes" case) If in a run of $\boldsymbol{P}$ on $t$ the state $q_P^s$ is assigned to an anonymous constant $c$ (occurring in $t$), then in this run only those runs of $\boldsymbol{I}$ will be considered where $q^s$ is assigned to the same anonymous constant $c$ and $q^d$ is assigned to all other anonymous constants, in particular, to all the anonymous constants generated by $\boldsymbol{T}$.

(2) ("No" case) If in a run of $\boldsymbol{P}$ on $t$ the state $q_P^d$ is assigned to each anonymous constant occurring in $t$, then in this run only those runs of $\boldsymbol{I}$ will be considered where $q^d$ is assigned to each anonymous constant occurring in $t'$ except for at most one anonymous constant generated by $\boldsymbol{T}$, which may get assigned $q^s$.

In order to be able to distinguish between (1) and (2) and to make sure that in (2) at most one anonymous constant gets assigned $q^s$, the TAAC $\boldsymbol{P}$ does some book keeping.

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

*State space of* $\boldsymbol{P}$. The state space of $\boldsymbol{P}$ is defined by

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

where we use the following notation to access the individual components of a state. For a state $b = (S, L, \alpha, M_d, M_s)$ of $\boldsymbol{P}$, 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 $s \in \{q^d, *\}$.

The intuitive meaning of the different components of a state $b$ is as follows. The set $\mathsf{Iset}(b)$ collects the states reachable by $\boldsymbol{I}$ on the input tree $t$ to $\boldsymbol{P}$. The set $\mathsf{LA}(b)$ collects the values of the look-ahead of $\boldsymbol{T}$ for the given input tree $t$. The value $\mathsf{seen}(b)$ distinguishes between (1) and (2) above: if $\boldsymbol{P}$ has observed (seen) that we are in case (1), then $\mathsf{seen}(b) = \mathsf{yes}$, and $\mathsf{seen}(b) = \mathsf{no}$ otherwise. The other two components are more difficult to describe.

In $D_{(q,s)}(b)$ we collect all states reachable in a run of $\boldsymbol{I}$ on some output tree $t'$ obtained by running $\boldsymbol{T}$ on $t$ starting in state $q$ where the register is undefined ($s = *$), holds an anonymous constant which $q^d$ is assigned to ($s = q^d$), or holds an anonymous constant which $q^s$ is assigned to ($s = q^s$), where, in the two latter cases, the constant is assumed not to belong to $t$. The runs of $\boldsymbol{I}$ on $t'$ are simulated with respect to a permitted substitution that maps all constants generated by $\boldsymbol{T}$ 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 $\boldsymbol{P}$ on all constants occurring in $t$.

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 $\boldsymbol{I}$ 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.

To provide further intuition for these components, let us look at an example. Assume that $\boldsymbol{T}$ does not contain $\varepsilon$-transitions and exactly one transition with $f \in \Sigma_2$ on the left-hand side, namely the transition $q(v_R, f(x,y)) \to^{q_A} g(v_R, x, q'(v_N), y))$, where $g \in \Sigma_3$, $q, q'$ are states of $\boldsymbol{T}$, and $x$ and $y$ are variables. We refer to this transition by (*). Furthermore, assume that $\boldsymbol{P}$ just read terms $t_0$ and $t_1$, resulting in states $b_0$ and $b_1$, respectively, and that $\boldsymbol{P}$ is about to read $f \in \Sigma_2$, i.e., the current configuration of $\boldsymbol{P}$ is $f(b_0, b_1)$. We consider the components $D_{(q,s)}(b)$ and $S_{(q,s)}(b)$ in $b$. Clearly, we have that $D_{(q',s)}(b) = \emptyset$ and $S_{(q',s)}(b) = \emptyset$ for all $q' \neq q$ and all $s \in \{q^s, q^d, *\}$ since there is no transition in $\boldsymbol{T}$ with $q'$ and $f$ on the left-hand side. In other words, when applied to some term with head $f$ in state $q'$, $\boldsymbol{T}$ does not produce output. Also, $D_{(q,s)}(b) = \emptyset$ and $S_{(q,s)}(b) = \emptyset$ if $q_A \notin \mathsf{LA}(b)$ since (*) is the only transition in $\boldsymbol{T}$ with left-hand side containing $f$, but $\boldsymbol{T}$ cannot apply (*) if the look-ahead automaton of $\boldsymbol{T}$ does not accept the term $t = f(t_0, t_1)$ in state $q_A$. Now, assume that $q_A \in \mathsf{LA}(b)$. We have $D_{(q,*)}(b) = \emptyset$ and $S_{(q,*)}(b) = \emptyset$ since the left-hand side of transition (*) contains $v_R$ instead of '*'. For $D_{(q,q^d)}(b)$ we obtain the set $[g(q^d, \mathsf{Iset}(b_0), D_{(q',q^d)}(b_1)]_{\boldsymbol{I}}$. This is the set of states $\boldsymbol{I}$ can possibly be in after having read a term $t'$ produced by $\boldsymbol{T}$ on input $t$ in state $q$ where in the run of $\boldsymbol{I}$ on some $t'$ all anonymous constants generated by $\boldsymbol{T}$ are assigned to $q^d$ and the value of the register of $\boldsymbol{T}$ when starting to read $t$ is assigned to $q^d$ as well. Similarly, we have that $D_{(q,q^s)}(b) = [g(q^s, \mathsf{Iset}(b_0), D_{(q',q^d)}(b_1)]_{\boldsymbol{I}}$. The only difference here is that in the run of $\boldsymbol{I}$ the value of the register of $\boldsymbol{T}$ when starting to read $t$ is assigned to $q^s$, instead of $q^d$. The component $S_{(q,q^d)}(b)$ also contains those states $\boldsymbol{I}$ can possibly be in after having read outputs $t'$ of $\boldsymbol{T}$ where an anonymous constant generated by $\boldsymbol{T}$ may be assigned to $q^s$. Therefore, we have that $S_{(q,q^d)}(b) = [g(q^d, \mathsf{Iset}(b_0), D_{(q',q^s)}(b_1)]_{\boldsymbol{I}} \cup [g(q^d, \mathsf{Iset}(b_0), S_{(q',q^d)}(b_1)]_{\boldsymbol{I}}$. Note that by distinguishing between the $D$- and $S$-components, we can keep track of where we allow anonymous constants generated by $\boldsymbol{T}$ to be assigned to $q^s$ in runs of $\boldsymbol{I}$ and where we do not allowed such assignments. We finally note that if $\boldsymbol{T}$ had $\varepsilon$-transitions, the components just described would have to be augmented since $\boldsymbol{T}$ could also apply $\varepsilon$-transitions, and hence, produce more output terms. Below, we will therefore consider $\varepsilon$-closures of states of $\boldsymbol{P}$.

The intuition behind the components of the states of $\boldsymbol{P}$ is formally captured in Lemma 17.

*The $\varepsilon$-Closure.* We need one more definition before we can proceed.

Given a transition as in (18), 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, for $k < r$, $t^{\varepsilon,k}_{q',q'',b}$ as abbreviation for the term

$$t'[q', \ldots, q', q'', \ldots, q'', D_{(q_0,s_0)}(b), \ldots, D_{(q_{k-1},s_{k-1})}(b), S_{(q_k,s_k)}(b), D_{(q_{k+1},s_{k+1})}(b),$$

$$\ldots, D_{(q_{r-1},s_{r-1})}(b), \mathsf{Iset}(b), \ldots, \mathsf{Iset}(b)]$$

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 (17) and states $b_0, \ldots, b_{n-1}$ (they will be the states assigned to the arguments of $f$ in a run of $\boldsymbol{P}$), we write $t^{\Sigma}_{q',q''}$ as abbreviation for the term

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

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

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

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

Now we can define the $\varepsilon$-closure of a state of $\boldsymbol{P}$. Let $b \in Q_P$. We define a sequence $b_0, b_1, \ldots$ by induction, where, in the base case, we set $b_0 = b$, and for $i \geqslant 0$ we define the components of $b_{i+1}$ as follows:

- $\mathsf{Iset}(b_{i+1}) = \mathsf{Iset}(b)$, $\mathsf{LA}(b_{i+1}) = \mathsf{LA}(b)$, $\mathsf{seen}(b_{i+1}) = \mathsf{seen}(b)$.
- For every $q \in Q_T$ and $s \in \{q^d, q^s, *\}$, the set $D_{(q,s)}(b_{i+1})$ is obtained from $D_{(q,s)}(b_i)$ by adding all states $a$ for which there exists an $\varepsilon$-transition as in (18) such that $q_A \in \mathsf{LA}(b_i)$, $z = *$ iff $s = *$, and $a \in [t^{\varepsilon}_{s,q^d,b_i}]_{\boldsymbol{I}}$.
- For every $q \in Q_T$ and $s \in \{q^d, *\}$, the set $S_{(q,s)}(b_{i+1})$ is obtained from $S_{(q,s)}(b_i)$ by adding all states $a$ for which there exists an $\varepsilon$-transition as in (18) such that $q_A \in \mathsf{LA}(b_i)$, $z = *$ iff $s = *$, and $a \in [t^{\varepsilon}_{s,q^s,b_i}]_{\boldsymbol{I}}$ or $a \in [t^{\varepsilon,k}_{s,q^d,b_i}]_{\boldsymbol{I}}$ for some $k < r$.

Clearly, there exists an index $j$ such that $b_j = b_{j+1} = \ldots$ We define the *$\varepsilon$-closure of $b$* to be $b_j$ for such a $j$ and denote it by $\overline{b}$.

*The Default and selecting states of $\boldsymbol{P}$.* The default state $q^d_P$ of $\boldsymbol{P}$ is defined as the $\varepsilon$-closure $\overline{b^d}$ of the following state $b^d$:

- $\mathsf{Iset}(b^d) = [q^d]_{\boldsymbol{I}}$, $\mathsf{LA}(b^d) = [q^d_A]_{\boldsymbol{A}}$, $\mathsf{seen}(b^d) = \mathsf{no}$,
- $D_{(q,s)}(b^d) = \emptyset$ 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 $s \in \{q^d, *\}$.

Similarly, the selecting state $q^s_P$ of $\boldsymbol{P}$ is defined as the $\varepsilon$-closure $\overline{b^s}$ of the following state $b^s$:

- $\mathsf{Iset}(b^s) = [q^s]_{\boldsymbol{I}}$, $\mathsf{LA}(b^s) = [q^d_A]_{\boldsymbol{A}}$, $\mathsf{seen}(b^s) = \mathsf{yes}$,
- $D_{(q,s)}(b^s) = \emptyset$ 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 $s \in \{q^d, *\}$.

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

For every $f \in \Sigma_n$ and $b_0, \ldots, b_{n-1} \in Q_P$, the automaton $\boldsymbol{P}$ contains the transition $f(b_0, \ldots, b_{n-1}) \to b$ where $b$ is defined to be the $\varepsilon$-closure $\overline{b'}$ of the following state $b'$ determined by the following three conditions.

- $\mathsf{Iset}(b') = [f(\mathsf{Iset}(b_0), \ldots, \mathsf{Iset}(b_{n-1}))]_{\boldsymbol{I}}$, $\mathsf{LA}(b') = [f(\mathsf{LA}(b_0), \ldots, \mathsf{LA}(b_{n-1}))]_{\boldsymbol{A}}$, $\mathsf{seen}(b') = \mathsf{yes}$ if there exists $i$ such that $\mathsf{seen}(b_i) = \mathsf{yes}$, and $\mathsf{seen}(b') = \mathsf{no}$ otherwise.
- For any $q$ and $s$, we have $a \in D_{(q,s)}(b')$ if there exists a $\Sigma$-transition as in (17) such that $q_A \in \mathsf{LA}(b')$, $z = *$ iff $s = *$, and $a \in [t^{\Sigma}_{s,q^d}]_{\boldsymbol{I}}$ for every $q \in Q_T$ and $s \in \{q^d, q^s, *\}$.
- For any $q$ and $s$, we have $a \in S_{(q,s)}(b')$ if there exists a $\Sigma$-transition as in (17) such that $q_A \in \mathsf{LA}(b')$, ($z = *$ iff $s = *$), and $a \in [t^{\Sigma}_{s,q^s}]_{\boldsymbol{I}}$ or $a \in [t^{\Sigma,k}_{s,q^d}]_{\boldsymbol{I}}$ for some $k$ for every $q \in Q_T$ and $s \in \{q^d, *\}$.

*Final states of* $\boldsymbol{P}$. The set of final states $F_P$ of $\boldsymbol{P}$ contains a state $b$ if there exists $q \in I_T$ and $a \in F_I$ such that

- $\mathsf{seen}(b) = \mathsf{yes}$ and $a \in D_{(q,*)}(b)$, or
- $\mathsf{seen}(b) = \mathsf{no}$ and $a \in S_{(q,*)}(b)$.

### 3.3. Correctness of the construction

The following proposition states that our construction is correct.

**Proposition 15.** $T(\boldsymbol{P}) = \tau_{\boldsymbol{T}}^{-1}(T(\boldsymbol{I}))$.

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

**Lemma 16.** *For every $t$ and permitted substitution $\sigma$, there exists a state $b$ such that $\{b\} = [\sigma(t)]_{\boldsymbol{P}}$. This state $b$ is $\varepsilon$-closed, i. e., $\overline{b} = b$.*

The second lemma is more involved, and it is the key for proving Proposition 15. It formalizes all the intuition behind our construction.

**Lemma 17.** *For every term $t \in T_{\Sigma^C}$, $b \in Q_P$, and permitted substitution $\sigma$ such that $b \in [\sigma(t)]_{\boldsymbol{P}}$ the following is true, where we write $\sigma' =_t \sigma$ to say that $\sigma'$ is a permitted substitution which coincides with $\sigma$ on $\mathrm{occ}_C(t)$.*

(1) $\mathsf{lset}(b) = [\sigma(t)]_{\boldsymbol{I}}$.
(1) $\mathsf{LA}(b) = [t]_{\boldsymbol{A}}$.
(3) *The following are equivalent*:
  (*A*) $\mathsf{seen}(b) = \mathsf{yes}$.
  (*B*) *There exists $c \in \mathrm{occ}_C(t)$ such that $\sigma(c) = q^s$.*
(4) *For every $a \in Q_I$ and $q \in Q_T$, the following are equivalent*:
  (*A*) $a \in D_{(q,*)}(b)$.
  (*B*) *There exists $t' \in T_{\Sigma^C}$ and $\sigma'$ such that $\sigma' =_t \sigma$, $\sigma'(c) = q^d$ for every $c \in \mathrm{occ}_C(t') \setminus \mathrm{occ}_C(t)$, $a \in [\sigma'(t')]_{\boldsymbol{I}}$, and $q(*,t) \vdash^* t'$.*
(5) *For every $a \in Q_I$, $q \in Q_T$, and $c \notin \mathrm{occ}_C(t)$, the following are equivalent*:
  (*A*) $a \in D_{(q,q^d)}(b)$.
  (*B*) *There exists $t' \in T_{\Sigma^C}$ and $\sigma' =_t \sigma$ s.t. $\sigma'(c') = q^d$ for every $c' \in \mathrm{occ}_C(t') \setminus \mathrm{occ}_C(t)$, $\sigma'(c) = q^d$, $a \in [\sigma'(t')]_{\boldsymbol{I}}$, and $q(c,t) \vdash^* t'$.*
(6) *If $\mathsf{seen}(b) = \mathsf{no}$, then for every $a \in Q_I$, $q \in Q_T$, $c \notin \mathrm{occ}_C(t)$, the following are equivalent*:
  (*A*) $a \in D_{(q,q^s)}(b)$.
  (*B*) *There exists $t' \in T_{\Sigma^C}$ and $\sigma'$ s.t. $\sigma' =_t \sigma$, $\sigma'(c) = q^s$, $a \in [\sigma'(t')]_{\boldsymbol{I}}$, and $q(c,t) \vdash^* t'$.*
  *Note that $\sigma(c') = q^d$ for every $c' \in \mathrm{occ}_C(t)$ since $\mathsf{seen}(b) = \mathsf{no}$.*
(7) *If $\mathsf{seen}(b) = \mathsf{no}$, then for every $a \in Q_I$, $q \in Q_T$, the following are equivalent*:
  (*A*) $a \in S_{(q,*)}(b)$.
  (*B*) *There exists $t' \in T_{\Sigma^C}$ and $\sigma'$ s.t. $\sigma' =_t \sigma$, $a \in [\sigma'(t')]_{\boldsymbol{I}}$, and $q(*,t) \vdash^* t'$.*
(8) *If $\mathsf{seen}(b) = \mathsf{no}$, then for every $a \in Q_I$, $q \in Q_T$, $c \notin \mathrm{occ}_C(t)$, the following are equivalent*:
  (*A*) $a \in S_{(q,q^d)}(b)$.
  (*B*) *There exists $t' \in T_{\Sigma^C}$ and $\sigma'$ s.t. $\sigma' =_t \sigma$, $\sigma'(c) = q^d$, $a \in [\sigma'(t')]_{\boldsymbol{I}}$, and $q(c,t) \vdash^* t'$.*

Before proving this lemma, we use it to prove Proposition 15.

**Proof** (*Proposition 15*)**.**
$T(\boldsymbol{P}) \subseteq \tau_{\boldsymbol{T}}^{-1}(T(\boldsymbol{I}))$: Assume that $t \in T(\boldsymbol{P})$. It follows that there exists a permitted substitution $\sigma$ and a final state $b \in F_P$ such that $b \in [\sigma(t)]_{\boldsymbol{P}}$. 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 17, (4) implies that there exists $t'$ and a permitted substitution $\sigma'$ such that $q(*,t) \vdash^* t'$ and $a \in [\sigma'(t')]_{\boldsymbol{I}}$, and thus, $t' \in T(\boldsymbol{I})$ since $a \in F_I$. This means that $t \in \tau_{\boldsymbol{T}}^{-1}(T(\boldsymbol{I}))$.

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 17, (7) there exists $t'$ and a permitted substitution $\sigma'$ such that $q(*,t) \vdash^* t'$ and $a \in [\sigma'(t')]_{\boldsymbol{I}}$. Thus, $t' \in T(\boldsymbol{I})$ and $t \in \tau_{\boldsymbol{T}}^{-1}(T(\boldsymbol{I}))$.

$T(\boldsymbol{P}) \supseteq \tau_{\boldsymbol{T}}^{-1}(T(\boldsymbol{I}))$: assume that $t \in \tau_{\boldsymbol{T}}^{-1}(T(\boldsymbol{I}))$. This means that there exists $t'$, $a \in F_I$, and a permitted substitution $\sigma$ such that $q(*,t) \vdash^* t'$ for some $q \in I_T$ and $a \in [\sigma(t')]_{\boldsymbol{I}}$. Let $b \in [\sigma(t)]_{\boldsymbol{P}}$. Such a $b$ exists and is uniquely determined due to Lemma 16. We show that $b \in F_P$, and thus, $t \in T(\boldsymbol{P})$. First, assume that $\mathsf{seen}(b) = \mathsf{yes}$. Then, by Lemma 17, (3) and since $\sigma$ is a permitted substitution, we have that $\sigma(c) = q^d$ for every $c \notin \mathit{occ}_C(t)$. By Lemma 17, (4), we can conclude that $a \in D_{(q,*)}(b)$. Second, if $\mathsf{seen}(b) = \mathsf{no}$, Lemma 17, (7). implies that $a \in S_{(q,*)}(b)$. In both cases, we get $b \in F_P$. □

**Proof** (*Lemma 17*)**.** Let $t \in T_{\Sigma^C}$, $b \in Q_P$, and $\sigma$ be a permitted substitution such that $b \in [\sigma(t)]_{\boldsymbol{P}}$. Statements (1), (2), and (3) are easy to verify by the construction of $\boldsymbol{P}$. We prove (4)–(8) simultaneously. We first show the implications from left to right by structural induction on $t$ and then establish the other direction by induction on the length of computations.

"⇒": *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 implications 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 that there exists $t'$ and a permitted substitution $\sigma'$ such that $\sigma' =_t \sigma$, $\sigma'(c) = q^d$, $q(c,t) \vdash^* t'$, and $a \in [\sigma'(t')]_{\boldsymbol{I}}$. If $a \in S_{(q,q^d)}(b_i)$, this follows by the induction hypothesis. Otherwise, by definition of $S_{(q,q^d)}(b_{i+1})$, we know that there exists an $\varepsilon$-transition as in (18) such that $q_A \in \mathsf{LA}(b_i)$, $z = v_R$, and $a \in [t^{\varepsilon}_{q^d,q^s,b_i}]_{\boldsymbol{I}}$ or $a \in [t^{\varepsilon,k}_{q^d,q^d,b_i}]_{\boldsymbol{I}}$ for some $k$. First suppose that $a \in [t^{\varepsilon}_{q^d,q^s,b_i}]_{\boldsymbol{I}}$. Then, there exist $a_j \in D_{(q_j,s_j)}(b_i)$ such that $a \in [t'[q^d,\dots,q^d,q^s,\dots,q^s,a_0,\dots,a_{r-1},\mathsf{Iset}(b_i),\dots,\mathsf{Iset}(b_i)]]_{\boldsymbol{I}}$. 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 $\sigma' =_t \sigma$ and $\sigma'(c) = q^d$. Using the induction hypothesis on $i$, it is easy to verify that there exist $t'_0,\dots,t'_{r-1}$ such that $q_j(c_j,t) \vdash^* t'_j$, $a_j \in [\sigma'(t'_j)]_{\boldsymbol{I}}$, 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 $\sigma'$ as the permitted substitution for every $j$. Thus, we have $q(c,t) \vdash^* t'[c,\dots,c,c',\dots,c',t'_0,\dots,t'_{r-1},t,\dots,t] := t''$ and $a \in [\sigma'(t'')]_{\boldsymbol{I}}$. The case where $a \in [t^{\varepsilon,k}_{q^d,q^d,b_i}]_{\boldsymbol{I}}$ for some $k$ can be dealt with in an analogous fashion. This concludes the proof of the base case.

What we have basically shown here is that if the implications hold for some state, then they hold for the $\varepsilon$-closure of this state. The fact that $t$ is an anonymous constant was only used to show the implication for $b_0$.

*Induction step.* Assume that $t = f(t_0,\dots,t_{n-1})$ and that the inclusions hold true for the subterms $t_i$ of $t$. Let $b_i$ be the unique element with $b_i \in [\sigma(t_i)]_{\boldsymbol{P}}$. Similar to the base case, one can show that the inclusions hold for $b'$ where $b'$ is defined as in the definition of transitions of $\boldsymbol{P}$. Moreover, from the proof of the base case it follows that the inclusions stay true when taking the $\varepsilon$-closure of a state. Thus, they hold true for $b = \overline{b'}$.

"⇐": We prove (4)–(8) simultaneously by induction on the length of computations. For computations of length zero nothing is to show since the conditions in (B) are void. 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 substitution $\sigma'$ such that $\sigma' =_t \sigma$, $\sigma'(c) = q^d$, $a \in [\sigma'(t'')]_{\boldsymbol{I}}$, 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 $\Sigma$- or $\varepsilon$-transition.

*$T$ is a $\Sigma$-transition.* Assume that $T$ is a $\Sigma$-transition of the form (17) where $z = v_R$. We use $c' \in C \setminus (\mathit{occ}_C(t) \cup \{c\})$ as the new constant generated by $T$ (in case $T$ is generative). We have that $t = f(t_0,\dots,t_{n-1})$ for some $t_i \in T_{\Sigma^C}$. Let $b_i$ be the uniquely determined element in $[\sigma(t_i)]_{\boldsymbol{P}} = [\sigma'(t_i)]_{\boldsymbol{P}}$. Then, we know that $b$ is $\overline{b'}$ where $b'$ is defined as in the definition of transitions. 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

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

Since $a \in [\sigma'(t'')]_{\boldsymbol{I}}$, there exist $a_i \in Q_I$ such that $a_i \in [\sigma'(t'_i)]_{\boldsymbol{I}}$ and

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

By Lemma 17, (1) and since $\sigma'$ and $\sigma$ coincide on $occ_C(t)$, we have $\mathsf{Iset}(b_{i_j}) = [\sigma'(t_{i_j})]_{\boldsymbol{I}}$. Lemma 17, (2) ensures that $q_A \in \mathsf{LA}(b') = \mathsf{LA}(b) = [t]_{\boldsymbol{A}}$. 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 $\sigma'$ meets the conditions for $D_{(q_i,s_i)}(b_{j_i})$ with respect to $t_{j_i}$. Thus, the induction hypothesis on the length of computations yields $a_i \in D_{(q_i,s_i)}(b_{j_i})$. Now, it follows that $a \in [t^{\Sigma}_{q^d,q^s}]_{\boldsymbol{I}}$, and thus, $a \in S_{(q,q^d)}(b') \subseteq S_{(q,q^d)}(\overline{b'}) = S_{(q,q^d)}(b)$.

Second, assume that $\sigma'$ is a permitted substitution such that $\sigma'(c'') = q^d$ for every $c'' \in occ_C(t) \cup \{c,c'\}$. We consider two subcases. First, suppose that there exists $k$ and $c'' \in occ_C(t'_k)$ such that $\sigma'(c'') = q^s$. It follows that $c'' \notin occ_C(t) \cup \{c,c'\}$, 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 again easy to check that $\sigma'$ meets the conditions for $D_{(q_i,s_i)}(b_{j_i})$ with respect to. $t_{j_i}$ for every $i \neq k$ and $S_{(q_k,s_k)}(b_{j_k})$ with respect to. $t_{j_k}$. 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, $a \in [t^{\Sigma,k}_{q^d,q^s}]_{\boldsymbol{I}}$. 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 $c'' \in occ_C(t'_k)$ such that $\sigma'(c'') = q^s$, then one can similarly show that $a \in [t^{\Sigma,k}_{q^d,q^s}]_{\boldsymbol{I}}$ even for every $k$, and thus, $a \in S_{(q,q^d)}(b)$.

*T is an ε-transition.* This case can be dealt with very similar to the case for Σ-transitions. Instead of using the definition of transitions of $\boldsymbol{P}$ one uses the definition of the $\varepsilon$-closure and the fact that $b$ is $\varepsilon$-closed by Lemma 16. □

## 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 [34,26,5] 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 TTAC's. These transducers have two important features necessary to model recursive receive–send actions, but missing in decidable 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—a feature not necessary for non-looping protocols when analyzed with respect to a finite number of receive–send actions.

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 $\mathcal{K}$ 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 from $\mathcal{A}$, the unary symbols $\mathtt{hash}_a$ (*keyed hash*) and $\mathtt{enc}_a^{\mathsf{s}}$ (*symmetric encryption*) for every $a \in \mathcal{A}$, the unary symbol $\mathtt{enc}_k^{\mathsf{a}}$ (*asymmetric encryption*) for every $k \in \mathcal{K}$, and the binary symbol $\langle\rangle$ (*pairing*). Instead of $\langle\rangle(t, t')$ we write $\langle t, t'\rangle$. The term $\mathtt{hash}_a(m)$ shall represent the keyed hash of $m$ under the key $a$ plus $m$ itself. One could make this explicit by writing $\langle m, \mathtt{hash}(\langle a, m\rangle)\rangle$ instead of $\mathtt{hash}_a(m)$. However, checking whether a message is of the form $\langle m, \mathtt{hash}(\langle a, m\rangle)\rangle$ for some message $m$ requires the *non-linear* term $\langle x, \mathtt{hash}(\langle a, x\rangle)\rangle$, where $x$ is a variable. To avoid such non-linear terms we use $\mathtt{hash}_a(m)$, but allow the intruder (see below) to derive $m$ from $\mathtt{hash}_a(m)$. The set of *messages* over $(\Sigma_{\mathcal{A}}, \mathcal{C})$ is denoted by $\mathcal{M} = T_{\Sigma_{\mathcal{A}}}(\mathcal{C})$.

Note that anonymous constants are not allowed as keys. It is an open problem whether the decision problem ATTACK (see Section 4.4) would still be decidable otherwise.

### 4.2. The Intruder

As in the case of models for non-looping protocols, our intruder model is based on the Dolev–Yao intruder [14]. 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 $\mathtt{enc}_a^{\mathsf{s}}(m) \in \mathsf{d}(\mathcal{S})$ and $a \in \mathsf{d}(\mathcal{S})$, then $m \in \mathsf{d}(\mathcal{S})$ (symmetric decryption);
(4) if $\mathtt{enc}_k^{\mathsf{a}}(m) \in \mathsf{d}(\mathcal{S})$ and $k^{-1} \in \mathsf{d}(\mathcal{S})$, then $m \in \mathsf{d}(\mathcal{S})$ (asymmetric decryption);
(5) if $\mathtt{hash}_a(m) \in \mathsf{d}(\mathcal{S})$, 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 $\mathtt{enc}_a^{\mathsf{s}}(m) \in \mathsf{d}(\mathcal{S})$ (symmetric encryption);
(8) if $m \in \mathsf{d}(\mathcal{S})$ and $k \in \mathcal{K} \cap \mathsf{d}(\mathcal{S})$, then $\mathtt{enc}_k^{\mathsf{a}}(m) \in \mathsf{d}(\mathcal{S})$ (asymmetric encryption);
(9) if $m \in \mathsf{d}(\mathcal{S})$ and $a \in \mathcal{A} \cap \mathsf{d}(\mathcal{S})$, then $\mathtt{hash}_a(m) \in \mathsf{d}(\mathcal{S})$ (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 (6)–(9)

It is well known that $\mathsf{d}(\mathcal{S})$ can be obtained by first applying $\mathsf{an}$ to $\mathcal{S}$ and to the result applying $\mathsf{syn}$. This is because we employ atomic keys; for complex keys this does not hold (see, e.g., [32,28]):

**Lemma 18.** *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 TTAC's, 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 19.** A *message transducer* $\boldsymbol{T}$ is a TTAC over $(\Sigma_{\mathcal{A}}, \mathcal{C})$.

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

**Definition 20.** A (TTAC-based) principal $\Pi$ is a tuple

$$((\boldsymbol{T}_0, \ldots, \boldsymbol{T}_{n-1}), \mathcal{I})$$

consisting of a sequence $(\boldsymbol{T}_0, \ldots, \boldsymbol{T}_{n-1})$ 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 $\boldsymbol{T}_i$.

The single message transducers $\boldsymbol{T}_i$ in the definition of $\Pi$ are called it receive–*send actions*. In a protocol run, $\Pi$ 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 $\boldsymbol{T}_i$ in the current run. Now, if in the protocol run the first message $\Pi$ receives is $m_0$, then $\Pi$ returns some message $m_0'$ with $(m_0, m_0') \in \tau_{\boldsymbol{T}_0(q_0)}$. Then, on receiving the second message, say $m_1$, $\Pi$ returns $m_1'$ with $(m_1, m_1') \in \tau_{\boldsymbol{T}_1(q_1)}$, and so on. By fixing the initial states at the beginning, we model that $\Pi$ can convey (a finite amount of) information from one receive–send action to another. For example, if $q_0$ encodes that $\Pi$ expects to talk to Bob, then $q_1$ might describe that in the second message $\Pi$ expects to see Bob's name again.

A protocol is defined to be a finite family of principals plus—since we are interested in attacks on protocols—an initial intruder knowledge and information about which receive–send actions are to be thought of as being "challenge" actions:

**Definition 21.** A (TTAC-based) protocol $P$ is a tuple $(\{\Pi_i\}_{i<n}, \mathcal{S}, C)$ 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*,
– $C \subseteq \{0, \ldots, n-1\}$ is the set of *challenge indices.*

The last action of principal $i$ will be called a *challenge output action* if $i \in C$. (The use of *challenge output actions* will be explained in the next section.)

The class of protocols that can be specified according to Definition 21 can roughly be characterized as follows: in every receive–send action, principals can perform recursive computations, which, since described as TTAC, allow to *recursively* carry out linear checks on input messages (i.e., checks that can be expressed in terms of matching against linear terms). For example, in one receive–send action a principal can go through a list of requests and check the format of every request by matching against a linear term. In the recursive process principals can also produce an unbounded number of fresh nonces (anonymous constants) and build complex output messages, e.g., a list of certificates containing sessions keys (freshly generated anonymous constants) or a message containing deeply nested encryptions. Furthermore, as explained above, principals can convey a finite amount of information from one receive–send action to the next. In Section 7, we illustrate the kind of protocols that can be modeled by examples.

### *4.4. Attacks*

In an attack on a protocol, the intruder, who has complete control over the communication network, 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 some weak kind of *authentication*. For example, the secret may be some auxiliary message indicating that a principal, say $P$, completed a session with an instance of another principal, say $P'$, 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. Stronger forms of authentication could also be checked

if the absence of certain receive–send actions would be tested. However, authentication is not the main focus of the present work and we therefore will not further investigate authentication here.

In the definition of attacks we make use of challenge output actions (see Definition 21). 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 Sections 4.1 and 8), 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 [4].

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 most models for non-looping protocols this is not possible (see, however, [37,18,27]).

**Definition 22.** Let $P = (\{\Pi_i\}_{i<n}, \mathcal{S}, C)$ be a protocol with $\Pi_i = ((\boldsymbol{T}^i_0, \ldots, \boldsymbol{T}^i_{n_i-1}), \mathcal{I}_i)$ 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

$$(O, <, \psi)$$

consisting of

- a non-empty set $O \subseteq \{(i,j) \mid i < n, j < n_i\}$ of indices of receive–send actions triggered during an attack,
- a total ordering $<$ on $O$, the interleaving of the receive–send actions, and
- a mapping $\psi$ assigning to every $(i,j) \in O$ a tuple $\psi(i,j) = (q^i_j, m^i_j, m'^i_j)$

and satisfying the following conditions:

(1) For every $(i,j) \in O$, if $(i,j) \in O$, then $(i,j') \in O$ and $(i,j') < (i,j)$ for every $j' < j$.
(2) Let $(i,j)$ be the greatest element of $O$ with respect to $<$. Then, for each $(i',j') \in O$, the action $\boldsymbol{T}^{i'}_{j'}$ is a challenge output action iff $(i,j) = (i',j')$.
(3) Let $i < n$ and $j$ maximal with $(i,j) \in O$. Then there exist $q^i_{j+1}, \ldots, q^i_{n_i-1}$ such that $(q^i_0, \ldots, q^i_{n_i-1}) \in \mathcal{I}_i$,
(4) For each $(i,j) \in O$ it holds that $m^i_j, m'^i_j \in \mathcal{M}$ and $(m^i_j, m'^i_j) \in \tau_{\boldsymbol{T}^i_j(q^i_j)}$.
(5) For each $(i,j) \in O$ it holds that $(occ_{\mathcal{C}}(m'^i_j) \setminus occ_{\mathcal{C}}(m^i_j)) \cap occ_{\mathcal{C}}(S^i_j) = \emptyset$ where $S^i_j = \mathcal{S} \cup \{m'^{i'}_{j'} \mid (i',j') < (i,j)\}$.
(6) For every $(i,j) \in O$ it holds that $m^i_j \in \mathsf{d}(S^i_j)$.

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 $c \in \mathsf{d}(S^i_j) \cap (\mathcal{A} \cup \mathcal{C})$.

Note that (5) ensures that new anonymous constants generated in one receive–send action are also new with respect to the knowledge of the intruder before this action is performed. Note also that in (6) the set $\mathsf{d}(S^i_j)$ is the intruder knowledge before performing the receive–send action in step $(i,j)$.

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 23.** 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.

**Proof.** Formally, an intruder which may generate anonymous constants is an intruder whose initial knowledge contains an infinite number of anonymous constants (in addition to his ordinary initial knowledge). We argue that one anonymous constant, say $c_I$, is enough.

The reason is that TTAC's 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 $\sigma$ 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. This means, in particular, if $(m_j^i, {m'}_j^i)$ is part of an attack, then so is $(\sigma(m_j^i), \sigma({m'}_j^i))$ where $\sigma$ maps anonymous constants from the initial intruder knowledge in $m_j^i$ to $c_I$. Observe that if $m_j^i$ can be derived by the intruder, then so can $\sigma(m_j^i)$. □

Since in models for non-looping protocols *dis*equality tests between messages are usually not possible as well (an exception is [15]), 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 24.** *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 $\boldsymbol{T}_0, \ldots, \boldsymbol{T}_{l-1}$ of message transducers (the interleaving of receive–send actions) with $\boldsymbol{T}_i = (Q_i, I_i, \boldsymbol{A}_i, \Gamma_i)$ for $i < l$, decide whether there exist messages $m_i, m_i' \in \mathcal{M}$, $i < l$, such that*

(1) $(m_i, m_i') \in \tau_{\boldsymbol{T}_i}$ *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 $(\mathcal{S}, \boldsymbol{T}_0, \ldots, \boldsymbol{T}_{l-1}) \in$ InterleavingAttack if all the above conditions are satisfied.

The proof of the decidability of InterleavingAttack proceeds in two steps. First, we show that the intruder can be simulated by a TTAC (see Section 5.1). Then, we reduce the problem InterleavingAttack to the problem IteratedPreImage (Section 5.2), which we know is decidable.

### 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 $\boldsymbol{T}_{der}$ such that $\tau_{\boldsymbol{T}_{der}}(m) = \mathsf{d}(\{m\})$ for every message $m$.

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

#### 5.1.1. Key discovery

The key discovery automaton $\boldsymbol{D}$ is a complete and deterministic WTAAC containing all information about which keys can be accessed in a given message. More precisely, a state of the key discovery automaton is a function $2^{\mathcal{A}} \to 2^{\mathcal{A}}$ and the automaton is set up in a way such that the state $[m]_D$ the automaton is in after reading the

message $m$—note that since $\boldsymbol{D}$ is complete and deterministic, $[m]_D = \{\varphi\}$ for some function $\varphi$—has the following property: $[m]_{\boldsymbol{D}}(K)$ is the set of atoms that may be derived from $K$ and $m$, i.e., $[m]_{\boldsymbol{D}}(K) = \mathsf{an}(\{m\} \cup K) \cap \mathcal{A}$ for every $K \subseteq \mathcal{A}$ and message $m$. In the following, it is argued that this is indeed possible, that is, we will construct $\boldsymbol{D}$ with the desired property. The set of all functions $2^{\mathcal{A}} \to 2^{\mathcal{A}}$, that is, the state set of $\boldsymbol{D}$, is denoted $Q_D$. We note that the cardinality of $Q_D$ is double exponential in the cardinality of $\mathcal{A}$.

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

$$a \to (K \mapsto K \cup \{a\})$$

for every $K \subseteq \mathcal{A}$.

For every $a \in \mathcal{A}$, $k \in \mathcal{K}$, and $d' \in Q_D$, the WTAAC $\boldsymbol{D}$ contains transitions

$$\begin{aligned} \mathtt{enc}_a^{\mathtt{s}}(d') &\to d, \\ \mathtt{enc}_k^{\mathtt{a}}(d') &\to d, \\ \mathtt{hash}_a(d') &\to d, \end{aligned}$$

where $d$ is determined by the following table:

| left-hand side | condition on $K$ | $d(K)$ |
|---|---|---|
| $\mathtt{enc}_a^{\mathtt{s}}(d')$ | $a \in K$ | $d'(K)$ |
| $\mathtt{enc}_a^{\mathtt{s}}(d')$ | $a \notin K$ | $K$ |
| $\mathtt{enc}_k^{\mathtt{a}}(d')$ | $k^{-1} \in K$ | $d'(K)$ |
| $\mathtt{enc}_k^{\mathtt{a}}(d')$ | $k^{-1} \notin K$ | $K$ |
| $\mathtt{hash}_a(d')$ | no condition | $K$. |

Finally, for every $d', d'' \in Q_D$, the WTAAC $\boldsymbol{D}$ contains a transition

$$\langle d', d'' \rangle \to d \tag{19}$$

where, 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 definition of (19) is more involved than the definition of the other transitions since in a message of the form $\langle m, m' \rangle$ keys in $m$ could be used to obtain new keys in $m'$ (and vice versa) and these keys can in turn be used to obtain new keys in $m$, which can in turn be used to obtain new keys in $m'$, and so on. Therefore $d(K)$ is defined by a least fixed point.

The correctness of the construction is asserted in the following lemma:

**Lemma 25.** *For every message $m \in \mathcal{M}$ and $K \subseteq \mathcal{A}$ we have that*

$$[m]_{\boldsymbol{D}}(K) = \mathsf{an}(\{m\} \cup K) \cap \mathcal{A}.$$

**Proof.** The proof is straightforward and can be carried out by induction on the structure of $m$, using the definition of $\mathsf{an}(\cdot)$. □

*5.1.2. The Transducer $\boldsymbol{T}_{der}$*

The TTAC $\boldsymbol{T}_{der}$ is based on a simple idea, which is motivated by Lemma 18. To describe it, we need some more notation. For a set $K \subseteq \mathcal{A}$, let $\Sigma(K)$ be the signature defined by

$$\Sigma(K) = \{\mathtt{enc}_a^s, \mathtt{hash}_a \mid a \in K\} \cup \{\mathtt{enc}_k^a \mid k \in K \cap \mathcal{K}\}.$$

Using this notation, Lemma 18 implies:

**Lemma 26.** *Let $m, m' \in \mathcal{M}$. Then the following statements are equivalent*:

(*A*) $m' \in \mathsf{d}(\{m\})$.

(*B*) *$m'$ is of the form $t[m_0, \ldots, m_{n-1}]$ where the $m_i$ are obtained by successive decryptions and splittings from $m$, and the keys in $t$ are also obtainable by successive decryptions and splittings from $m$, i.e., there exists a linear term $t(x_0, \ldots, x_{n-1}) \in T_{\Sigma(\mathsf{an}(\{m\})\cap\mathcal{A})}(X)$ such that $m' = t[m_0, \ldots, m_{n-1}]$ where $m_i \in \mathsf{an}(\{m\})$ for every $i < n$.*

The TTAC $\boldsymbol{T}_{der}$ 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 on a given message $m$. The first phase is just one step and simply determines the set $K$ of keys that can be discovered from the given message $m$, that is, it determines $K = \mathsf{an}\{m\} \cap \mathcal{A}$. In the second phase, the term $t$ from above is generated, 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 from $T_{\Sigma(\mathsf{an}(\{m\})\cap\mathcal{A})}(X)$. In the third phase, every copy of $m$ in $t[m, \ldots, m]$ is (non-deterministically) replaced by some message from $\mathsf{an}(\{m\})$. The above lemma guarantees that $\boldsymbol{T}_{der}$ exactly computes the messages derivable from $m$.

To be more precise, we have the following transitions in $\boldsymbol{T}_{der}$. Since for $\boldsymbol{T}_{der}$ no register is used, in what follows we write $s(t)$ instead of $s(*, t)$ where $s$ is a state of $\boldsymbol{T}_{der}$, 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 $d \in Q_D$, $\boldsymbol{T}_{der}$ contains the transition

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

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

$$\begin{aligned}
(q_S, K)(x) &\to \langle (q_S, K)(x), (q_S, K)(x)\rangle \\
(q_S, K)(x) &\to \mathtt{enc}_a^{\mathsf{s}}((q_S, K)(x)) && \text{for } a \in K \\
(q_S, K)(x) &\to \mathtt{enc}_k^{\mathsf{a}}((q_S, K)(x)) && \text{for } k \in K \cap \mathcal{K} \\
(q_S, K)(x) &\to \mathtt{hash}_a((q_S, K)(x)) && \text{for } a \in K \\
(q_S, K)(x) &\to (q_A, K)(x)
\end{aligned}$$

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

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

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

**Lemma 27.** *$\tau_{der}(m) = \mathsf{d}(\{m\})$ for every $m \in \mathcal{M}$.*

**Proof.** From the construction of $\boldsymbol{T}_{der}$, it is easy to see that if $m$ is a message with $K = \mathsf{an}(\{m\}) \cap \mathcal{A}$ and $m'$ is such that $q_I(m) \vdash^* m'$, then

$$\begin{aligned}
q_I(m) \vdash\ & (q_S, K)(m) \vdash^* t[(q_S, K)(m), \ldots, (q_S, K)(m)] \\
& \vdash^* t[(q_A, K)(m), \ldots, (q_A, K)(m)] \vdash^* t(m_0, \ldots, m_{n-1}) = m'
\end{aligned} \tag{20}$$

where $t \in T_{\Sigma(\mathsf{an}(\{m\})\cap\mathcal{A})}(X)$ and $m_i \in \mathsf{an}(\{m\})$ for every $i < n$. This shows that $\tau_{der}(m) \subseteq \mathsf{d}(\{m\})$.

Similarly, it is easy to see that for every choice of $t$ and $m$ as above, one has (20) with $K = \mathsf{an}(\{m\}) \cap \mathcal{A}$, which implies $\mathsf{d}(\{m\}) \subseteq \tau_{der}(m)$. □

Note that even if we allowed the intruder to generate anonymous constants, we could model such an intruder by a TTAC since TTAC's can generate anonymous constants. More precisely, one could simulate the intruder by a composition of two TTAC's: 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 $\varepsilon$-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 23, 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

We now reduce InterleavingAttack to IteratedPreImage by formulating an attack as a composition of transducers. We first need to introduce two variants of $\boldsymbol{T}_{der}$ and one variant of $\boldsymbol{T}_i$, mainly to pass on the intruder's knowledge from one transducer to the next.

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

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

The TTAC $\mathbf{T}_{der}^{copy}$ can be derived from $\boldsymbol{T}_{der}$ in a straightforward way. We equip $\boldsymbol{T}_{der}$ with an additional state, say $q_I^{copy}$, and declare it to be the initial state of $\mathbf{T}_{der}^{copy}$. In addition, 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 $\boldsymbol{T}_{der}$. For ease in notation, let $\tau_{der}^{copy} = \tau_{\mathbf{T}_{der}^{copy}}$.

The second variant, called $\mathbf{T}_{der}^{chall}$, expects an input of the form $\langle m, m'\rangle$, copies the *second* component into the output and simulates $\boldsymbol{T}_{der}$ 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. Again, this variant of $\boldsymbol{T}_{der}$ can easily be obtained from $\boldsymbol{T}_{der}$. We add one more state, say $q_I^{chall}$, which is the initial state of $\mathbf{T}_{der}^{chall}$, and we also add the following transition:

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

Let $\tau_{der}^{chall} = \tau_{\mathbf{T}_{der}^{chall}}$.

Finally, we introduce a variant $\hat{\boldsymbol{T}}_i$ of $\boldsymbol{T}_i$ to (i) pass on the intruder's knowledge and (ii) to satisfy condition (2) in the definition of InterleavingAttack, i.e., anonymous constants generated in a receive–send action are different from the anonymous constants generated so far. To this end, $\hat{\boldsymbol{T}}_i$ only accepts pairs as input, copies the first component into the output (this component stands for the intruder's knowledge) and simulates $\boldsymbol{T}_i$ on the second component (this component corresponds to the input for $\boldsymbol{T}_i$). Obviously, $\hat{\boldsymbol{T}}_i$ defined in this way accomplishes (i). But it also achieves (ii) since by our definition of the computations of a TTAC, anonymous constants generated by a transducer are different from those that occur in the input. It is again straightforward to obtain $\hat{\boldsymbol{T}}_i$ from $\boldsymbol{T}_i$: We add one state $q$ to the set of states of $\boldsymbol{T}_i$ and declare it to be the (only) initial state of $\hat{\boldsymbol{T}}_i$, and for every initial state $q_I$ of $\boldsymbol{T}_i$, we add the transition

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

Let $\hat{\tau}_i = \tau_{\hat{\boldsymbol{T}}_i}$.

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 defined by

$$m_S = \langle u_0, \langle u_1, \langle \cdots \langle u_{n-2}, u_{n-1}\rangle \cdots\rangle\rangle\rangle$$

(the order of the $u_i's$ does not matter); this makes sure that we have $\mathsf{d}(S) = \mathsf{d}(m_S)$. 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, by construction, we obtain the following characterization for the problem InterleavingAttack.

**Lemma 28.** *For every* $\mathcal{S}$ *and* $\boldsymbol{T}_0, \ldots, \boldsymbol{T}_{l-1}$ *as in the definition of the problem* InterleavingAttack, *we have*

$$(\mathcal{S}, \boldsymbol{T}_0, \ldots, \boldsymbol{T}_{l-1}) \in \text{InterleavingAttack} \; iff \; m_{\mathcal{S}} \in \tau^{-1}(R).$$

Together with Corollary 12 this immediately implies:

**Theorem 29.** InterleavingAttack *is decidable*.

This concludes the proof of Theorem 24.

By reduction from the intersection problem for top-down tree automata—given a sequence of top-down tree automata $A_1, \ldots, A_n$, decide whether the intersection $L(A_1) \cap \cdots \cap L(A_n)$ is empty—, which is known to be EXPTIME-complete [35], it is easy to see that the problems Attack and InterleavingAttack are EXPTIME-hard since TTACs can simulate such tree automata and by composing TTACs one can simulate the intersection of such automata. Since our decision procedure for the iterated pre-image word problem is non-elementary, this is also the case for the problems Attack and InterleavingAttack. Hence, it remains to find a tight complexity bound for these problems.

## 6. Extensions of the model 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 TTAC's 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 [11]. (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
   (a) left-hand sides of rewrite rules may be non-linear (this corresponds to allowing non-linear left-hand sides in transitions of TTAC) or
   (b) arbitrary messages can be conveyed from one receive–send action to another and can then be compared with other messages [2,34,26,5];
(2) complex keys, i.e., keys that may be arbitrary messages [34,26,5]; and

(3) relaxing the free term algebra assumption by adding the XOR operator [8,13] or Diffie-Hellman exponentiation [9].

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 then 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 with respect to an unbounded number of sessions—and thus, receive–send actions—, then ATTACK is undecidable as well. This is not surprising, since the same is true for models of non-looping protocols (see, e.g., [2]).

### 6.1. Encoding Post's Correspondence Problem

We start with an undecidability result which is purely automata-theoretic. It will then be used over and over again to obtain the protocol-related undecidability results. More precisely, we show how to model Post's Correspondence Problem (PCP) by composing transducers.

Recall that an instance of PCP is composed of two sequences $\alpha = \alpha_1, \ldots, \alpha_n$ and $\beta = \beta_1, \ldots, \beta_n$ of words over a two-letter alphabet $\Sigma$. The problem is to determine whether there exists a sequence $u = i_0, \ldots, i_k$ of indices such that $\alpha_{i_0} \ldots \alpha_{i_k} = \beta_{i_0} \ldots \beta_{i_k}$. Such a sequence of indices is called a feasible solution; any sequence is just called a solution. The two words are denoted $\alpha(u)$ and $\beta(u)$, respectively.

We will encode a word $\alpha = a_0 \ldots a_{l-1} \in \Sigma^*$ with $a_i \in \Sigma$ by the term $\langle a_0, \langle a_1, \ldots, \langle a_{l-1}, \bot\rangle \cdots\rangle$, which we denote $[\alpha]$. Here, $\bot$ is a new constant. We will also encode indices $i \in \{1, \ldots, n\}$. To this end, let $b$ be a new constant and let $b^i$ denote the word $b \cdots b$ of length $i$. Then, $i$ is encoded by $[b^i]$, which, for convenience, is also denoted $[i]$. If $u = i_0 i_1 \ldots i_l$ is a solution, we write $[u]$ for $\langle [i_0], \langle [i_1], \ldots, \langle [i_l], \bot\rangle \ldots\rangle\rangle$.

In addition to $b$, we will use the new constants $b_1$, $b_2$, and $\mathsf{secret}$. Thus the set of atomic messages is defined to be $\mathcal{A} = \Sigma \cup \{b, b_1, b_2, \bot, \mathsf{secret}\}$.

We next describe transducers $\boldsymbol{T}_0$, $\boldsymbol{T}_1$, and $\boldsymbol{T}_2$ such that an instance of the PCP as above has a feasible solution if and only if in $\tau_{\boldsymbol{T}_2}(\tau_{\boldsymbol{T}_1}(\tau_{\boldsymbol{T}_0}(b)))$ there exists a term $\langle t, t\rangle$.

Transducer $\boldsymbol{T}_0$ generates the encoding of any solution and therefore has the following transitions:

$$\begin{aligned} q_I(b) &\to \langle [i], q(b)\rangle \quad \text{for } i \in \{1, \ldots, n\},\\ q(b) &\to \langle [i], q(b)\rangle \quad \text{for } i \in \{1, \ldots, n\},\\ q(b) &\to \bot. \end{aligned}$$

Transducer $\boldsymbol{T}_1$ consists of one transition only and doubles every term:

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

In the above transition we write $\langle x_0, x_1\rangle$ instead of $x$ to make sure that the list of indices received from $\boldsymbol{T}_0$ is not empty.

Transducer $\boldsymbol{T}_2$ takes any pair of encoded solutions and replaces the lists by the concatenation of the corresponding words (encoded as described above). There is one transition which starts the substitution/concatenation process in each component:

$$q_I(\langle x, x'\rangle) \to \langle q_\alpha(x), q_\beta(x')\rangle.$$

In addition, for every $i \in \{1, \ldots, n\}$ with $\alpha_i = a_0 \cdots a_{l-1}$ and $\beta_i = b_0 \cdots b_{m-1}$, there are transitions for the substitution/concatenation process:

$$\begin{aligned} q_\alpha(\langle [i], x\rangle) &\to \langle a_0, \langle a_1, \ldots, \langle a_{l-1}, q_\alpha(x)\rangle \cdots\rangle,\\ q_\beta(\langle [i], x\rangle) &\to \langle b_0, \langle b_1, \ldots, \langle b_{m-1}, q_\beta(x)\rangle \cdots\rangle. \end{aligned}$$

Finally, there are two transitions to stop the substitution/concatenation process:

$$q_\alpha(\bot) \to \bot,$$
$$q_\beta(\bot) \to \bot.$$

Now, the following lemma is easy to see:

**Lemma 30.** *For a given instance of the PCP, let* $\boldsymbol{T}_0, \boldsymbol{T}_1, \boldsymbol{T}_2$ *be the transducers defined above. Then the following statements are equivalent*:

(*A*) *The instance has a feasible solution.*
(*B*) *There exists a term t such that* $\langle t, t\rangle \in \tau_{\boldsymbol{T}_2}(\tau_{\boldsymbol{T}_1}(\tau_{\boldsymbol{T}_0}(b)))$.

Before we apply this in the cryptographic setting, we use it for an automata-theoretic problem.

We define the following extension of TTAC's. A *top-down tree transducer with non-linear left-hand side* (TTNL) is a TTAC with transitions of the form as defined in (3) but where $t$ is not required to be linear.

Now we can prove:

**Theorem 31.** *For TTNL's, the iterated pre-image problem is undecidable.*

**Proof.** The proof is by reduction from the PCP. Given a PCP instance as above, we construct $\boldsymbol{T}_0$, $\boldsymbol{T}_1$, and $\boldsymbol{T}_2$ as above. In addition, we construct the TTNL $\boldsymbol{T}_3$ defined by the single transition

$$q_I(\langle x, x\rangle) \to \mathsf{secret}.$$

Clearly, $\tau_{\boldsymbol{T}_3}(t) \neq \emptyset$ only if $t = \langle t_0, t_1\rangle$ where $t_0 = t_1$, and if this is the case, then $\tau_{\boldsymbol{T}_3}(t) = \{\mathsf{secret}\}$. From this and Lemma 30, we obtain that the instance has a feasible solution if and only if $\mathsf{secret} \in (\tau_{\boldsymbol{T}_3} \circ \tau_{\boldsymbol{T}_2} \circ \tau_{\boldsymbol{T}_1} \circ \tau_{\boldsymbol{T}_0})^{-1}(\{b\})$, which completes the description of the desired reduction. □

We note that for this undecidability result to hold allowing $\varepsilon$-transitions in transducers is essential: if $\varepsilon$-transitions are not allowed, then on a given input, a transducer can only produce a finite number of outputs modulo new anonymous constants. In addition, it is easy to bound the number of anonymous constants to be considered. Thus, we obtain:

**Observation 32.** For TTNLs without $\varepsilon$-transitions, i.e., only with $\Sigma$-transitions of the form (17), the iterated pre-image word problem is decidable.

### 6.2. Equality tests on messages

We start with the following definition. A protocol where the receive–send actions are defined by TTNL's is called a *TTNL-based protocol.*

We show the following:

**Theorem 33.** *For TTNL-based protocols,* Attack *and* InterleavingAttack *are undecidable.*

**Proof.** The proof is by reduction from the PCP. We demonstrate the proof for InterleavingAttack; the one for Attack is essentially the same.

The reduction is essentially the same as the one in the proof of Theorem 31, but we cannot simply take the four transducers as the protocol description, because the intruder could interfere, and we have to produce a challenge output action. To prevent the intruder from interfering, we simply encrypt the output of every transducer with a key not known to the intruder and make sure that only encrypted messages are accepted. In addition, we use a different key for every transducer. This makes sure that the order in which the transducers are applied is preserved.

More precisely, we modify the transducers as follows:

– $\boldsymbol{T}_0$: We add the transition $q_I^*(x) \to \mathtt{enc}^{\mathtt{S}}_{b_0}(q_I(x))$ to $\boldsymbol{T}_0$ and declare $q_I^*$ to be the initial state.

– $\boldsymbol{T}_1$: We add the transition $q_I^*(\mathtt{enc}_{b_0}^{\mathtt{S}}(x)) \to \mathtt{enc}_{b_1}^{\mathtt{S}}(q_I(x))$ and declare $q_I^*$ to be the initial state.
– $\boldsymbol{T}_2$: We add the transition $q_I^*(\mathtt{enc}_{b_1}^{\mathtt{S}}(x)) \to \mathtt{enc}_{b_2}^{\mathtt{S}}(q_I(x))$ and declare $q_I^*$ to be the initial state.
– $\boldsymbol{T}_3$: We add the transition $q_I^*(\mathtt{enc}_{b_2}^{\mathtt{S}}(x)) \to q_I(x)$ and declare $q_I^*$ to be the initial state.

In addition, we add the transducer $\boldsymbol{T}_4$ defined by $q_I(b) \to \mathsf{secret}$. Now, it is clear that the instance of InterleavingAttack determined by the transducers $\boldsymbol{T}_0, \ldots, \boldsymbol{T}_4$ and the initial knowledge $\mathcal{S} = \{b\}$ is solvable iff the given instance of the PCP has a feasible solution. □

### 6.3. Complex keys and challenge outputs

To model complex keys, we replace the unary symbol $\mathtt{enc}_a^{\mathtt{S}}(\cdot)$ by the binary symbol $\mathtt{enc}(\cdot,\cdot)$. The message $\mathtt{enc}(m,m')$ 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 $\mathtt{enc}(m,m')$. If he knows $\mathtt{enc}(m,m')$ and $m$, then he knows $m'$ as well.

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

We have:

**Theorem 34.** *For TTAC-based protocols with complex keys*, Attack *and* InterleavingAttack *are undecidable*.

**Proof.** To see this it suffices to observe that in the reduction from the proof of Theorem 33, $\boldsymbol{T}_3$ can be replaced by the transducer defined by

$$q_I^*(\mathtt{enc}_{b_2}^{\mathtt{S}}(\langle x,x'\rangle)) \to \langle \mathtt{enc}^{\mathtt{S}}_{\mathtt{enc}_{b_3}^{\mathtt{S}}(x)}(\mathsf{secret}), \mathtt{enc}_{b_3}^{\mathtt{S}}(x')\rangle,$$

where $b_3$ is a new constant not known by the intruder. 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. □

Similarly, we obtain an undecidability result if we allow arbitrary challenge output actions. More precisely, the setting is as follows. The message space is defined as in Section 4, but in a challenge output action the principal is allowed to return any message from $\mathcal{M}$ as challenge (rather than an element of $\mathcal{C} \cup \mathcal{A}$).

**Theorem 35.** *For TTAC-based protocols with arbitrary challenge output messages*, Attack *and* InterleavingAttack *are undecidable*.

**Proof.** We modify the reduction from the proof of Theorem 33 appropriately. First, we define the transducer $\boldsymbol{T}_3$ by the transition

$$q_I\left(\mathtt{enc}_{b_2}^{\mathtt{S}}(\langle x,x'\rangle)\right) \to \langle \mathtt{enc}_{b_3}^{\mathtt{S}}(x), \mathtt{enc}_{b_4}^{\mathtt{S}}(x')\rangle$$

and $\boldsymbol{T}_4$ by

$$q_I\left(\mathtt{enc}_{b_4}^{\mathtt{S}}(x)\right) \to \mathtt{enc}_{b_3}^{\mathtt{S}}(x),$$

where $b_3$ and $b_4$ are new constants not known to the intruder. Thus, the challenge for the intruder is $\mathtt{enc}_{b_3}^{\mathtt{S}}(x')$ which he can derive only if the messages substituted for $x$ and $x'$ in the transition of $\boldsymbol{T}_2$ coincide. □

### 6.4. XOR and Diffie-Hellman exponentiation

We next prove that extending the protocol model by exclusive or (XOR) or by Diffie-Hellman exponentiation leads to undecidability. 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., [8] for other properties of XOR.) These properties induce an equivalence relation $\equiv$ on messages. For instance, $\mathtt{enc}^{\mathtt{S}}_a(m \oplus m) \equiv \mathtt{enc}^{\mathtt{S}}_a(0)$. 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 it does, however, not make a difference whether or not the intruder is equipped with this ability.

Also, one would require the transducers to work on equivalence classes of messages according to the XOR theory. However, it is easy to see that for the reduction this also does not matter.

**Theorem 36.** *For TTAC-protocols with XOR, the problem* Attack *and the problem* InterleavingAttack *are undecidable.*

**Proof.** To show undecidability, we can again modify the reduction from PCP as described in the proof of Theorem 33. Instead of just $\boldsymbol{T}_3$ we now need two transducers, $\boldsymbol{T}_3$ and $\boldsymbol{T}_4$. Transducer $\boldsymbol{T}_4$ in the proof of Theorem 33 is now called $\boldsymbol{T}_5$.

The transducer $\boldsymbol{T}_3$ is now given by

$$q_I\left(\mathtt{enc}^{\mathtt{S}}_{b_2}(\langle x, x'\rangle)\right) \to \mathtt{enc}^{\mathtt{S}}_{b_3}\left(x \oplus x'\right).$$

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

Now, $\boldsymbol{T}_4$ checks whether the intruder knows $\mathtt{enc}^{\mathtt{S}}_{b_3}(0)$ and if so returns secret. That is, $\boldsymbol{T}_4$ is defined by

$$q_I\left(\mathtt{enc}^{\mathtt{S}}_{b_3}(0)\right) \to \mathsf{secret}.$$

Transducer $\boldsymbol{T}_5$ is defined just as $\boldsymbol{T}_4$ in the proof of Theorem 33. □

A similar reduction is possible for Diffie-Hellman exponentiation [9] since the normalization also involves comparison of arbitrary messages, for instance, $Exp(g, x \cdot x'^{-1}) = 1$ iff $x = x'$. We obtain:

**Theorem 37.** *For TTAC-based protocols with Diffie-Hellman exponentiation,* 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 [7] and it extends the authentication protocol by [31] 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 [32].

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

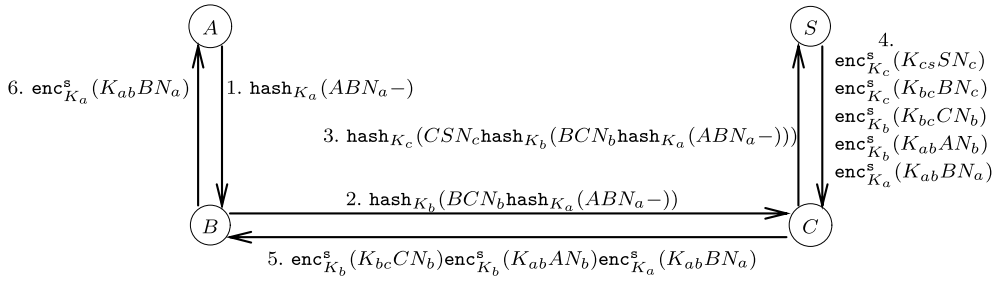


Fig. 1. A run of the recursive authentication protocol.

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 Fig. 1, $K_a$ (resp. $K_b, K_c$) denotes the long-term key shared between $A$ and $S$ (resp. $B$ and $S$, $C$ and $S$). 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 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 so-called 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. 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 $S$. We model static corruption and therefore partition the set of principals into honest $\mathcal{H}$ and dishonest $\mathcal{D}$ principals, i.e., we have $\mathcal{H} \cup \mathcal{D} = \{0, \ldots, n\}$ and $\mathcal{H} \cap \mathcal{D} = \emptyset$. We assume that $S$ is honest, and hence, $S \in \mathcal{H}$. The intruder will play the role of the dishonest principals and for this purpose his initial knowledge will contain $K_i$ for every $i \in \mathcal{D}$.

In the following, we first specify honest agents and the server. We then put everything together to formally specify the RA protocol as a TTAC-based protocol.

*Modeling the honest agents.* An honest agent $P_i, i < n$ and $i \in \mathcal{H}$, performs two receive–send actions and is given by the tuple $\Pi_i = \left(\boldsymbol{T}_{\mathbf{0}}^{i}, \boldsymbol{T}_{\mathbf{1}}^{i}, \mathcal{I}^i\right)$. The different components are defined next. The nonce sent by $P_i$ in the request message is denoted by the constant $N_i$.

The message transducer $\boldsymbol{T}_{\mathbf{0}}^{i}$ for sending the request message consists of two transitions. The first one is

$$(\mathsf{request}, \bot, P_{j'})(*, \mathsf{init}) \to \mathtt{hash}_{K_i}(P_i, P_{j'}, N_i, -),$$

and the second one is

$$(\mathsf{request}, P_j, P_{j'})(*, \mathtt{hash}_a(P_j, P_i, x_0, x_1)) \to \mathtt{hash}_{K_i}(P_i, P_{j'}, N_i, \mathtt{hash}_a(P_j, P_i, x_0, x_1)),$$

where $x_0$ and $x_1$ are variables, $j' \leqslant n$, $j < n$, $a \in \mathcal{A}$, and $\mathsf{init} \in \mathcal{A}$ 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 $\boldsymbol{T}_{\mathbf{0}}^{i}$ are $(\mathsf{request}, \bot, P_{j'})$ and $(\mathsf{request}, P_j, P_{j'})$ for every $j' \leqslant n$ and $j < n$.

The transducer $\boldsymbol{T}_{\mathbf{1}}^{i}$ is a challenge output action which receives a session key and sends it out to the intruder as a challenge in case the communication partner is honest. Note that the secrecy of a session key only needs to be guaranteed among honest principals. If one communication partner is dishonest it is clear that the intruder can derive the session key by simply following the protocol. Hence, in this case the challenge could always be met by the intruder. For every $j_1, j_2, j_3 \leqslant n$ and $j_4, j_5 < n$ with $j_1, j_3, j_4 \in \mathcal{H}$, $\boldsymbol{T}_{\mathbf{1}}^{i}$ contains the following transitions:

$$\begin{aligned}
(\mathsf{key}, \bot, P_{j_1})(*, \mathtt{enc}^{\mathtt{S}}_{K_i}(x_0, P_{j_1}, N_i)) &\to x_0\\
(\mathsf{key}, P_{j_2}, P_{j_3})(*, \langle \mathtt{enc}^{\mathtt{S}}_{K_i}(x_0, P_{j_3}, N_i), \mathtt{enc}^{\mathtt{S}}_{K_i}(x_1, P_{j_2}, N_i)\rangle) &\to x_0\\
(\mathsf{key}, P_{j_4}, P_{j_5})(*, \langle \mathtt{enc}^{\mathtt{S}}_{K_i}(x_0, P_{j_5}, N_i), \mathtt{enc}^{\mathtt{S}}_{K_i}(x_1, P_{j_4}, N_i)\rangle) &\to x_1
\end{aligned}$$

where $x_0$ and $x_1$ are variables. The first transition is applied if $P_i$ initiated the protocol run for communication with $P_{j_1}$. We require that $j_1 \in \mathcal{H}$ since if $P_{j_1}$ were dishonest, then, as already mentioned above, it is clear that the intruder could obtain $x_0$ by simply following the protocol. The other two transitions are applied if $P_i$ was called by $P_{j_2}$ and $P_{j_4}$, respectively, and called $P_{j_3}$ and $P_{j_5}$, respectively. For the same reason as above we require that $j_3, j_4 \in \mathcal{H}$. All states occurring in $\boldsymbol{T}_{\mathbf{1}}^{i}$ 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

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

This model of the agents is more precise than the one presented in [20] where word transducers have been used instead of tree transducers. While in [20], the nonces (the messages substituted for $x_0$ in $\boldsymbol{T}_{\mathbf{0}}^{i}$) needed to be typed since a word transducer can not parse arbitrary message, here any message can be substituted for $x_0$.

*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 $\boldsymbol{T}_n$. Formally, $P_n$ is defined by the tuple $\Pi_n = (\boldsymbol{T}_n, \{\mathsf{start}\})$ where start is the initial state of $\boldsymbol{T}_n$, with $\boldsymbol{T}_n$ defined next.

The transducer $\boldsymbol{T}_n$ has two states and works as described in Section 7.1.1. In state start, the initial state, $\boldsymbol{T}_n$ 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 $\boldsymbol{T}_n$ are specified as follows:

$$\mathsf{start}(*, \mathtt{hash}_{K_i}(P_i, P_n, x_0, x_1)) \to \mathsf{read}(v_N, \mathtt{hash}_{K_i}(P_i, P_n, x_0, x_1))$$
$$\mathsf{read}(v_R, \mathtt{hash}_{K_i}(P_i, P_j, x_0, -)) \to \mathtt{enc}^{\mathsf{S}}_{K_i}(v_R, P_j, x_0)$$
$$\mathsf{read}(v_R, \mathtt{hash}_{K_i}(P_i, P_j, x_0, \mathtt{hash}_{K_{i'}}(P_{i'}, P_i, x_1, x_2))) \to$$
$$\mathtt{enc}^{\mathsf{S}}_{K_i}(v_R, P_j, x_0), \mathtt{enc}^{\mathsf{S}}_{K_i}(v_N, P_{i'}, x_0), \mathsf{read}(v_N, \mathtt{hash}_{K_{i'}}(P_{i'}, P_i, x_1, x_2))$$

where $i, i', j \leqslant n$ 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 constant, respectively.

This model of the server is more precise than the one presented in [20]. 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, TTAC's allow to generate anonymous constants, and thus, provide a very natural way of modeling the creation of new session keys. The word transducers as considered in [20] did not have this capability. Therefore, in [20], 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 [34,26,5,2] the server cannot be modeled faithfully since these models do not allow to describe recursive processes.

*The specification of the RA protocol.* Given the specification of the honest agents and the server from above, the RA protocol is now formally specified by the tuple

$$(\{\Pi_i\}_{i \in \mathcal{H}}, \{P_0, \ldots, P_n\} \cup \{K_i \mid i \in \mathcal{D}\}),$$

i.e., we explicitly model the behavior of the honest agents, including the server, by the TTAC-based principals $\Pi_i$. The dishonest agents are subsumed by the intruder who has in his initial knowledge the names of all principals and the long-term keys the dishonest agents share with the server.

We note that while in an attack only at most one session of an honest agent is performed, the intruder can simulate an unbounded number of sessions of dishonest agents. In particular, one dishonest agent can be involved in many requests to the server, and hence, the length of the list of requests sent to the server in an attack is unbounded.

### 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., [11] 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, respectively, and $N_A$ and $N_B$ denote nonces generated by $A$ and $B$, respectively:

$$A \to B : \mathtt{enc}^{\mathsf{a}}_{K_B}(N_A, A)$$
$$B \to A : \mathtt{enc}^{\mathsf{a}}_{K_A}(N_A, N_B)$$

$$A \to B : \mathtt{enc}^{\mathtt{a}}_{K_B}(N_B)$$
$$B \to \quad : N_B$$

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 honest $A$ runs one instance of the protocol as initiator with the intruder $I$. We also model one instance of honest $B$ running in the role of a responder with $A$.

All receive–send actions can be modeled by TTAC's with only one state, which we call start, and one transition.

Principal $A$ is formally specified as a TTAC-based principal by the tuple $\Pi_A = ((\boldsymbol{T}_{\mathbf{0}}^{\boldsymbol{A}}, \boldsymbol{T}_{\mathbf{1}}^{\boldsymbol{A}}), \{\mathsf{start}\} \times \{\mathsf{start}\})$ where $\boldsymbol{T}_{\mathbf{0}}^{\boldsymbol{A}}$ and $\boldsymbol{T}_{\mathbf{1}}^{\boldsymbol{A}}$ specify the two receive–send actions performed by $A$. These TTAC's have the following transitions:

$$\boldsymbol{T}_{\mathbf{0}}^{\boldsymbol{A}}\ \mathsf{start}(*, \mathsf{init}) \to \mathtt{enc}^{\mathtt{a}}_{K_I}(N_A, A)$$
$$\boldsymbol{T}_{\mathbf{1}}^{\boldsymbol{A}}\ \mathsf{start}(*, \mathtt{enc}^{\mathtt{a}}_{K_A}(N_A, x)) \to \mathtt{enc}^{\mathtt{a}}_{K_I}(x)$$

Principal $B$ is formally specified as a TTAC-based principal by the tuple $\Pi_B = ((\boldsymbol{T}_{\mathbf{0}}^{\boldsymbol{B}}, \boldsymbol{T}_{\mathbf{1}}^{\boldsymbol{B}}), \{\mathsf{start}\} \times \{\mathsf{start}\})$ where $\boldsymbol{T}_{\mathbf{0}}^{\boldsymbol{B}}$ and $\boldsymbol{T}_{\mathbf{1}}^{\boldsymbol{B}}$ specify the two receive–send actions performed by $B$. These TTAC's have the following transitions:

$$\boldsymbol{T}_{\mathbf{0}}^{\boldsymbol{B}}\ \mathsf{start}(*, \mathtt{enc}^{\mathtt{a}}_{K_B}(x, A)) \to \mathtt{enc}^{\mathtt{a}}_{K_A}(x, N_B)$$
$$\boldsymbol{T}_{\mathbf{1}}^{\boldsymbol{B}}\ \mathsf{start}(*, \mathtt{enc}^{\mathtt{a}}_{K_B}(N_B)) \to N_B$$

Now, the Needham-Schroeder Public Key Protocol with honest $A$ running one instance of the protocol as initiator with the intruder $I$ and honest $B$ running one instance of the protocol as responder with $A$ is formally specified as a TTAC-based protocol by the tuple

$$(\{\Pi_A, \Pi_B\}, \{A, B, K_A, K_B, K_I, K_I^{-1}\}),$$

i.e., the instances of the honest principals $A$ and $B$ are explicitly specified by $\Pi_A$ and $\Pi_B$. The initial intruder knowledge contains the names of the principals and their public keys as well as the private key of the intruder.

The protocol specification above could of course be extended by instances of other principals or further instances of $A$ and $B$, e.g., those in which they talk to other principals. The one described above is sufficient to uncover the attack first found by [23].

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 [34,26,5,2]. Just as for the RA protocol, in [20] 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 (TAAC's) and transducers (TTAC's) over signatures with an infinite set of (anonymous) constants and proved that for TTAC's the iterated pre-image

word problem is decidable. Apart from the application to cryptographic protocols, we believe that the study of TAAC's and TTAC's started here is of independent interest.

One open problem is to establish tight complexity bounds for our decidability results. So far, our transducers allow for only one register. We believe that our results also hold even for the case of multiple registers. We have, however, not investigated this case, mainly because it was not necessary for our application and because it would have made the definitions and proofs more cumbersome. While here we do not allow anonymous constants as keys, this would be another interesting extension of our model. Our decision procedure for analyzing the security of recursive protocols has been implemented [30]. However, not much effort has been put into optimizations yet. So far, even on simplified versions of the Recursive Authentication Protocol and the Needham-Schroeder Public Key Protocol the implementation runs out of memory. This is due to the fact that for every pre-image automata the state space grows exponentially. It is not clear, from a complexity-theoretic point of view, whether this blowup can be avoided. From a practical point of view, much more effort has to be put into finding heuristics for cutting down the state space of the pre-image automata.

## Acknowledgment

We thank the anonymous reviewers for their helpful and detailed comments.

## References

[1] N. Alon, T. Milo, F. Neven, D. Suciu, V. Vianu, XML with data values: typechecking revisited, Journal of Computer and System Sciences 66 (4) (2003) 688–727.
[2] R. Amadio, D. Lugiez, V. Vanackere, On the symbolic reduction of processes with cryptographic functions, Theoretical Computer Science 290 (1) (2002) 695–740.
[3] G. Ateniese, M. Steiner, G. Tsudik, Authenticated group key agreement and friends, in: Proceedings of the 5th ACM Conference on Computer and Communication Security (CCS'98), ACM Press, San Francisco, CA, 1998, pp. 17–26.
[4] M. Bellare, 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), ACM, 1995, pp. 57–66.
[5] M. Boreale, Symbolic trace analysis of cryptographic protocols, in: F. Orejas, P. Spirakis, J. van Leeuwen (Eds.), Automata, Languages and Programming, 28th International Colloquium (ICALP 2001), Lecture Notes in Computer Science, vol. 2076, Springer-Verlag, 2001, pp. 667–681.
[6] J. Bryans, S. Schneider, CSP, PVS, and a Recursive Authentication Protocol, in: DIMACS Workshop on Formal Verification of Security Protocols, 1997.
[7] J. Bull, D. Otway, The authentication protocol, Technical Report DRA/CIS3/PROJ/CORBA/SC/1/CSM/436-04/03, Defence Research Agency, Malvern, UK, 1997.
[8] Y. Chevalier, R. Küsters, M. Rusinowitch, 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). IEEE, IEEE, Computer Society Press, 2003, pp. 261–270.
[9] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents, in: P. Pandya, J. Radhakrishnan (Eds.), FSTTCS 2003: Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science, vol. 2914, Springer, Berlin, 2003, pp. 124–135.
[10] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, Deciding the security of protocols with commuting public key encryption, Electronic Notes in Theoretical Computer Science 125 (1) (2005) 55–66.
[11] J. Clark, J. Jacob, A Survey of Authentication Protocol Literature, Web Draft Version 1.0, Available from: <http://citeseer.nj.nec.com/>, 1997.
[12] H. Comon, V. Cortier, J. Mitchell, Tree automata with one memory, set constraints, and ping-pong protocols, in: F. Orejas, P. Spirakis, J. van Leeuwen (Eds.), Automata, Languages and Programming, 28th International Colloquium (ICALP 2001), vol. 2076, Lecture Notes in Computer Science, 2001, pp. 682–693.
[13] H. Comon-Lundh, 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). IEEE, Computer Society Press, 2003, pp. 271–280.
[14] D. Dolev, A. Yao, On the security of public-key protocols, IEEE Transactions on Information Theory 29 (2) (1983) 198–208.
[15] N. Durgin, P. Lincoln, J. Mitchell, A. Scedrov, Multiset rewriting and the complexity of bounded security protocols, Journal of Computer Security 12 (2) (2004) 247–311.
[16] J. Engelfriet, Three hierarchies of transducers, Mathematical Systems Theory 15 (1982) 95–125.

[17] N. Ferguson, B. Schneier, A Cryptographic Evaluation of IPsec, Technical report, Available from: <http://www.counterpane.com/ipsec.pdf/>, 2000.
[18] T. Genet, F. Klay, Rewriting for cyptographic protocol verification, in: D. McAllester (Ed.), Proceedings of the 17th International Conference on Automated Deduction (CADE 2000), Lecture Notes in Computer Science, vol. 1831, Springer, 2000, pp. 271–290.
[19] J. Goubault-Larrecq, A method for automatic cryptographic protocol verification (extended abstract), in: Workshop on Formal Methods for Parallel Programming (FMPPTA 2000), vol. 1800, Lecture Notes in Computer Science, Springer, Berlin, 2000, pp. 977–984.
[20] R. Küsters, On the decidability of cryptographic protocols with open-ended data structures, in: L. Brim, P. Jancar, M. Kretinsky, A. Kucera (Eds.), 13th International Conference on Concurrency Theory (CONCUR 2002), Lecture Notes in Computer Science, vol. 2421, Springer-Verlag, 2002, pp. 515–530.
[21] R. Küsters, T. Truderung, On the atomatic analysis of recursive security protocols with XOR, in: W. Thomas, P. Weil (Eds.), Proceedings of the 24th Symposium on Theoretical Aspects of Computer Science (STACS 2007), Lecture Notes in Computer Science, vol. 4393, Springer, Berlin, 2007, pp. 646–657.
[22] R. Küsters, T. Wilke, Automata-based analysis of recursive cryptographic protocols, in: V. Diekert, M. Habib (Eds.), 21st Symposium on Theoretical Aspects of Computer Science (STACS 2004), Lecture Notes in Computer Science, vol. 2996, Springer-Verlag, 2004, pp. 382–393.
[23] G. Lowe, An attack on the Needham-Schroeder public-key authentication protocol, Information Processing Letters 56 (1995) 131–133.
[24] C. Meadows, Extending formal cryptographic protocol analysis techniques for group protocols and low-level cryptographic primitives, in: P. Degano (Ed.), Proceedings of the First Workshop on Issues in the Theory of Security (WITS'00), 2000, pp. 87–92.
[25] C. Meadows, Open issues in formal methods for cryptographic protocol analysis, in: Proceedings of DISCEX 2000, IEEE Computer Society Press, 2000, pp. 237–250.
[26] J.K. Millen, V. Shmatikov, Constraint solving for bounded-process cryptographic protocol analysis, in: Proceedings of the 8th ACM conference on Computer and Communications Security, ACM Press, 2001, pp. 166–175.
[27] D. Monniau, Abstracting cryptographic protocols with tree automata, in: A. Cortesi, G. Filé (Eds.), Proceedings of the 6th International Symposium on Static Analysis (SAS '99), Lecture Notes in Computer Science, vol. 1694, Springer, Berlin, 1999, pp. 149–163.
[28] D. Monniaux, Decision procedures for the analysis of cryptographic protocols by logics of belief, in: 12th Computer Security Foundations Workshop (CSFW 1999), IEEE Computer Society, 1999, pp. 44–54.
[29] F. Neven, T. Schwentick, V. Vianu, Towards regular languages over infinite alphabets, in: J. Sgall, A. Pultr, P. Kolman (Eds.), Mathematical Foundations of Computer Science (MFCS 2001), Lecture Notes in Computer Science, vol. 2136, Springer, Berlin, 2001, pp. 560–572.
[30] A. Obermann, Verifikation kryptographischer Protokolle mit Baumtransduktionen, Master's thesis, Fachbereich Informatik, TU Kaiserslautern, 2004.
[31] D. Otway, O. Rees, Efficient and timely mutual authentication, Operating Systems Review 21 (1) (1987) 8–10.
[32] L. Paulson, Mechanized proofs for a recursive authentication protocol, in: 10th IEEE Computer Security Foundations Workshop (CSFW-10), IEEE Computer Society Press, 1997, pp. 84–95.
[33] O. Pereira, J.-J. Quisquater, A security analysis of the cliques protocols suites, in: Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW-14), 2001, pp. 73–81.
[34] M. Rusinowitch, M. Turuani, Protocol insecurity with finite number of sessions is NP-complete, in: 14th IEEE Computer Security Foundations Workshop (CSFW-14), IEEE Computer Society, 2001, pp. 174–190.
[35] H. Seidl, Haskell overloading is DEXPTIME-complete, Information Processing Letters 52 (2) (1994.)
[36] V. Shmatikov, Decidable analysis of cryptographic protocols with products and modular exponentiation, in: D. Schmidt (Ed.), 13th European Symposium on Programming (ESOP 2004), Lecture Notes in Computer Science, vol. 2986, Springer, Berlin, 2004, pp. 355–369.
[37] T. Truderung, Regular protocols and attacks with regular knowledge, in: R. Nieuwenhuis (Ed.), Proceedings of the 20th International Conference on Automated Deduction (CADE 2005), Lecture Notes in Computer Science, vol. 3328, Springer-Verlag, 2005, pp. 377–391.
[38] T. Truderung, Selecting theories and recursive protocols, in: M. Abadi, L. de Alfaro (Eds.), Proceedings of the 16th International Conference on Concurrency Theory (CONCUR 2005), Lecture Notes in Computer Science, vol. 3653, Springer-Verlag, 2005, pp. 217–232.
[39] J. Zhou, Fixing a security flaw in IKE protocols, Electronic Letter 35 (13) (1999) 1072–1073.