Integrated Specification and Verification of Security Protocols and Policies

Integrated Specification and Verification of Security Protocols and Policies

Simone Frau
ETH Zürich

Mohammad Torabi-Dashti
ETH Zürich

ABSTRACT

We propose a language for formal specification of service-oriented architectures. The language supports the integrated specification of communication level events, policy level decisions, and the interaction between the two. We show that the reachability problem is decidable for a fragment of service-oriented architectures. The decidable fragment is well suited for specifying, and reasoning about, security-sensitive architectures. In the decidable fragment, the attacker controls the communication media. The policies of services are centered around the *trust application* and *trust delegation* rules, and can also express RBAC systems with role hierarchy. The fragment is of immediate practical relevance: we have specified and verified a number of security-sensitive architectures stemming from the e-government domain.

1. INTRODUCTION

Security protocols and authorization logics are two major techniques used in securing software systems. A central role of any (security) protocol is to give meaning to the messages that are exchanged in the course of the protocol [16]. For example, a signed X.509 certificate sent by a certificate authority is in many security protocols *meant* to imply that the authority endorses the public key and its owner, mentioned in the certificate. There are several ways to make the meanings of messages, and in general actions, of a protocol explicit, e.g. by associating epistemic effects to the actions [8]. In this paper, we propose a formal language for specifying service-oriented architectures, in which

- the messages received by a service are interpreted in terms of policy statements of the service, and
- the authorization policies of the service constrain the actions the service can perform.

The proposed language is well suited for integrated specification of security protocols and authorization policies in service-oriented architectures. We see a service-oriented architecture as a collection of finitely many services which communicate over insecure media. Each service consists of a number of processes that run in parallel and share a *policy engine*. Processes communicate by sending and receiving messages, as it is usual in asynchronous message passing environments. Each send event is constrained by a *guard*, and each receive event leads to an *update*. Guards and updates belong to the *policy level*, as opposed to send and receive events which constitute the *communication level*. In anthropomorphic terms, services "think" at the policy level, and "talk" at the communication level.

Technical Report #702
Dept. Computer Science, ETH Zürich
6 December 2010.

From an operational point of view, guards are predicates which, if derivable by the policy engine of a service, allow the service to perform a corresponding send action, cf. Dijkstra's guarded command language. Updates are also predicates at the policy level. When a service receives a message in one of its processes, it adds the corresponding update predicates to its policy engine. Intuitively, updates associate meanings to the messages a service receives in terms of predicates in the policy level. The notion of updates is similar to the *assumptions* which are relied upon after receiving a message, in the trust management model of Guttman et al. [13].

Motivations. The separation between the communication and policy levels is a useful abstraction for better understanding each of these levels. Indeed, distributed authorization logics, such as [7, 4, 11], typically abstract away the communication level events by assuming that all the policy statements exchanged among the participants are signed certificates. This frees the modelers from specifying the exact routes through which the statements travel, etc. The abstraction however obscures how each of the policy statements are represented (as messages) in a given application, how messages are interpreted as policy statements, whether there is a place for misinterpretation, etc. For instance, one would expect that the statement "Ann says employee(Piet)" is added to the (e.g. BINDER [7]) policy engine of a service, only after the service receives a message which is *meant* to indicate that Piet is Ann's employee. However, the meanings of messages (determined by their format, who has signed them, etc.) is often not specified in the policy level. Therefore, in a concrete environment, it is unclear whether or not the attacker can fake a message which would mean that Piet is Ann's employee, even though he is not. Similarly, formal specifications of security protocols, e.g. as in [15, 17], fully detail the format of the exchanged messages, while the meanings of messages in terms of policy statements are left unspecified.

While maintaining the separation between the communication and policy levels, we believe that, for a thorough security analysis, the interaction between the two levels must also be defined precisely. A typical specification in our proposed language thus consists of three components: communication level events, policy level decisions, and the the interface between the two. As the interface between the levels is explicitly present in the specifications, a more precise security analysis of service-oriented architectures becomes possible. This singles out our specification language from the formalisms which focus on either the communication or the policy level, and hence neglect their interactions.

Decision procedure. We assume that the attacker is in direct control of the communication media, i.e. messages are passed through the attacker. The message composition capabilities of the attacker may, for example, reflect the Dolev-Yao threat model [9]. The attacker can indirectly affect the policies of the participating services, by sending tampered messages which in turn affect the update predicates.

A generic *reachability* problem is defined for service-oriented architectures specified in the language. The reachability problem subsumes the secrecy problem for security protocols and the safety problem for authorization policies. The reachability problem turns out to be undecidable in general, even when assuming a finite bound on the number of participating services. We give a decision algorithm for the reachability problem under the following two conditions: (1) the message composition and decomposition capabilities of the attacker reflect the Dolev-Yao threat model, and (2) policy engines of (honest) services are centered around the *trust application* and *trust delegation* rules à la DKAL [11], besides *type-1* theories (formally defined in section 4. Type-1 theories are sufficiently expressive for modelling, e.g., RBAC systems with role hierarchy. The trust application and trust delegation rules, which are the core of many distributed authorization logics [7, 4, 11], intuitively state that

Trust application If Ann trusts Mike on statement f, and Mike says f, then Ann believes f holds.

Trust delegation If Ann trusts Mike on statement f, and Mike delegates the right to state f to, e.g., Piet, then Ann trusts Piet on statement f.

Trust delegation often contributes to the resilience and flexibility of access control systems. In practice, however, for a given application, trust delegation may, or may not, be allowed. Our formalism and decision algorithm can be adapted to exclude (transitive) trust delegation, if desired.

The decidable fragment is of practical interest: several industrial service-oriented architectures studied in the context of AVANTSSAR [2] (The EU Project on Automated Validation of Trust and Security of Service-oriented Architectures) fall into this fragment. As a comprehensive example, a case study on specifying and verifying an on-line car registration service [2], stemming from the European initiative for *points of single contact*, is reported in this paper.

To prove our decidability result, we encode the derivation of authorization predicates in the policy engine of a service into message inference trees induced by the Dolev-Yao deduction rules. The encoding benefits us in two ways: (1) it simplifies the decidability proof, and (2) it allows us to build upon existing tools which have been originally developed for verifying security protocols. In particular, we have extended the constraint solver of Millen and Shmatikov [15] in Prolog to validate service-oriented architectures.

Note that verification algorithms for correctness of security protocols and authorization logics have been mostly developed in isolation. For instance, it has been shown that the secrecy problem is decidable for security protocols with a bounded number of sessions [17, 15]. For these results the local computational power of the processes is limited to pattern matching, hence not fully accounting for authorization policies of the participants. Likewise, (un)decidability results for the safety problem in the HRU access control matrix model, and authorization logics such as [7, 4, 11] abstract away communication level events and their effects on policy level decisions. In contrast, our decision algorithm for reachability takes the communication and policy levels into account, and also covers the interface between them.

Related work. Our proposed language can be used to specify security protocols as it is common in the literature, e.g. see [15]. Authorization policies are modeled as logic programs in the language. Logic programs have been extensively used for specifying and reasoning about policies, e.g. see BINDER [7], SECPAL [4], and DKAL [11].

Recent progress in analyzing business processes, augmented with authorization policies, is related to our work, e.g. see [1, 18]. These studies focus on using specific formalisms and techniques for selected case studies, and thus do not consider decidability issues in general. A notable exception to this is [3], where workflows, policy level predicates and their interfaces are all formalized in first-order logic. Reachability is not considered in [3].

Road map. Section 2 describes an on-line car registration service: this serves as our running example. Section 3 introduces the syntax and semantics of the language we propose for specifying service-oriented architectures. There, we also formally define the reachability problem. A decidable fragment of architectures is identified in section 4. A constraint-solving algorithm for deciding reachability in the fragment is given in section 5. Section 6 concludes the paper. The proofs that are omitted in the text can be found in appendix B.

2. A RUNNING EXAMPLE: CRP

We give an informal description of a service-oriented architecture for online car registration procedure, originated from the European initiative for *points of single contact*, see [2]. We refer to the case study as CRP. A formal model of CRP and its verification results are given in subsequent sections.

CRP involves a number of parties: Mike the new owner of a car, Piet the employee of the car registration office, Ann the head of the car registration office, the human resources department of the office, called hr, and the central repository server, referred to as cr.

Mike buys a car, but before he is allowed to drive it, he must register the car at the car registration office. The office provides an online registration service. After producing a document containing all the necessary data for registering the car, Mike sends the document to one of the employees of the office, Piet. If the document is valid, Piet sends it to the central repository server to be permanently stored. The *cr* allows only the employees of the car registration office to write on the server, and Piet is not initially known to the *cr* as an employee. The *cr* trusts Ann, the head of the office, on who is an employee of the office. Ann, however, has decided to delegate this task (or, right) to the human resources department *hr*, and communicates this decision to the *cr* with a certificate. Consequently, the *cr* inquiries the *hr* on the status of Piet, to which the *hr* replies with a certificate stating that Piet is an employee of the office. Finally, the *cr* accepts Piet's request to store the document. After storing the document, the *cr* sends back an acknowledgement to Piet. Piet, in turn, sends a token of successful completion of the registration to Mike.

Below, we present the message exchange pattern for the CRP. The usual primitives for security protocols are employed: asymmetric encryptions $\{\cdot\}_\cdot$, signatures $sig(\cdot,\cdot)$, public key constructors $pk(\cdot)$, hash functions $h(\cdot)$ and pairing $(\cdot,\cdot)$. When confusion is unlikely, we simply write x, y for the pair (x, y), and write $[x]_a$ for $a, x, sig(a, x)$. We assume these cryptographic operators are ideal, à la Dolev and Yao [9]. Below, terms in sans-serif are flags, i.e. unique constants which denote the purpose of their accompanying messages.

$$\begin{aligned}
mike \to piet &: \{mike, doc\}_{pk(piet)}, [h(doc)]_{mike}\\
piet \to cr &: \{mike, doc\}_{pk(cr)}, [ann, h(mike, doc)]_{piet}\\
cr \to piet &: [h(mike, doc)]_{cr}\\
piet \to mike &: [h(mike, doc), \mathsf{success\_token}]_{piet}
\end{aligned}$$

The delegation of Ann's right to the *hr*, and the *hr*'s attestation that Piet is an employee, go over a separate exchange.

$$\begin{aligned}
cr \to ann &: piet, \mathsf{empl\_status}\\
ann \to cr &: [piet, \mathsf{is\_empl}, \mathsf{delegated\_to}, hr]_{ann}\\
cr \to hr &: piet, \mathsf{empl\_status}\\
hr \to cr &: [piet, \mathsf{is\_empl}]_{hr}
\end{aligned}$$

This specification falls short of capturing the internal reasonings of the participants involved, as described informally above. For instance, it is not clear how the participants must interpret the messages they receive, and how the *cr* ascertains Piet's right to store. These relations are often formalized in authorization logics, such as DKAL. However, specifying the internal reasoning in, e.g. SECPAL and DKAL, would fall short of determining the actual messages exchanged, or how the messages are related to policy statements.

In section 4.1, we specify CRP in our formal language. Then, we can answer questions such as: whether the attacker can take the CRP system into a state in which, Eve, who is not an employee of the office, can write into the repository. Such questions cannot be meaningfully posed when considering the communication or the policy levels of CRP in isolation.

3. A LANGUAGE FOR SPECIFYING SERVICE ORIENTED ARCHITECTURES

The syntax and semantics of the language are defined below. A typical architecture specified in the language consists of communication events of services, how received messages are interpreted, how services make logical decisions, and how these decisions affect the communication events.

3.1 Syntax

A *signature* is a tuple $(\Sigma, \mathcal{V}, \mathcal{P})$, where Σ is a countable set of functions, $\mathcal{V}$ is a countable set of variables, $\mathcal{P}$ is a nonempty finite set of predicates, and these sets are pairwise disjoint. We use the capital letters $A, B, \ldots$ to denote the elements of $\mathcal{V}$. The free term algebra induced by Σ, with variables $\mathcal{V}$, is denoted $\mathcal{T}_{\Sigma(\mathcal{V})}$. A *message* is an element of $\mathcal{T}_{\Sigma(\emptyset)}$, i.e. a ground term. The set of *atoms* $\mathcal{A}_{\Sigma(\mathcal{V})}$ is defined as $\{p(t_1, \cdots, t_n) \mid p \in \mathcal{P}, t_i \in \mathcal{T}_{\Sigma(\mathcal{V})},\ \text{arity of } p \text{ is } n\}$. The total function $var : \mathcal{T}_{\Sigma(\mathcal{V})} \cup \mathcal{A}_{\Sigma(\mathcal{V})} \to 2^{\mathcal{V}}$ gives the set of variables appearing in terms and atoms. A *fact* is an atom with no variables.

Fix a signature. An *event* is of either of the following forms:

- $g_1 \vee \cdots \vee g_k \blacktriangleright \mathsf{s}(t)$,
- $\mathsf{r}(t) \rhd u$

where $t \in \mathcal{T}_{\Sigma(\mathcal{V})}$, each g_i, with $1 \le i \le k$ and $k \ge 0$, is a finite set of atoms, and u is a finite set of atoms. Intuitively, an event of the form $g_1 \vee \cdots \vee g_k \blacktriangleright \mathsf{s}(t)$ denotes *guarded send*, where term t is sent to the network only if the *guard* $g_1 \vee \cdots \vee g_k$ is evaluated to "true" (for the exact definition, see section 3.2). An event of the form $\mathsf{r}(t) \rhd u$ denotes *receive* followed by *update*, where receiving term t results in adding the update u to the policy level. The function var extends to events as $var(g_1 \vee \cdots \vee g_k \blacktriangleright \mathsf{s}(t)) = \bigcup_{1 \le i \le k} var(g_i) \cup var(t)$ and $var(\mathsf{r}(t) \rhd u) = var(t) \cup var(u)$.

We write E for the set of all events, and E^* for the set of all finite sequences of events. A *process* is a finite sequence of events $\mathfrak{e} = e_1 \cdots e_n$ where

- If $e_i = g_1 \vee \cdots \vee g_k \blacktriangleright \mathsf{s}(t)$, then $var(e_i) \subseteq \bigcup_{1 \le j < i} var(e_j)$, for all $1 \le i \le n$.
- If $e_i = \mathsf{r}(t) \rhd u$, then $var(u) \subseteq var(t) \bigcup_{1 \le j < i} var(e_j)$, for all $1 \le i \le n$.

These conditions intuitively state that the behavior of any process depends deterministically on its input.

A *service* π is a tuple $(\eta_\pi, \Omega_\pi, \mathbf{I}_\pi)$, where η_π is a finite set of processes, Ω_π is a finite set of facts, called the *knowledge* of π, and $\mathbf{I}_\pi$ is a finite set of Horn clauses, called the *intensional knowledge* of π. A Horn clause is of the form $a \leftarrow a_1, \cdots, a_n$, with $n \ge 0$, and $a, a_1, \cdots, a_n$ being atoms.

An *attacker* model $\mathbb{A}$ is a service with no processes, i.e. it is a pair $(\Omega_\mathbb{A}, \mathbf{I}_\mathbb{A})$, where $\Omega_\mathbb{A}$ is a finite set of facts, called the knowledge of the attacker, and $\mathbf{I}_\mathbb{A}$ is a finite set of Horn clauses, referred to as the intensional knowledge of the attacker. The attacker is also able to send and receive messages; these capabilities are reflected in the execution model described in section 3.2.

A (service-oriented) *architecture* is a tuple $((\Sigma, \mathcal{V}, \mathcal{P}), \Pi, \mathbb{A})$, where $(\Sigma, \mathcal{V}, \mathcal{P})$ is a signature, Π is a finite nonempty set of services and $\mathbb{A}$ is an attacker model, where Π and $\mathbb{A}$ are defined using the signature $(\Sigma, \mathcal{V}, \mathcal{P})$. In order to avoid trivial name clashes, it is assumed that variables that appear in different processes of an architecture are distinct.

We assume that for any architecture $((\Sigma, \mathcal{V}, \mathcal{P}), \Pi, \mathbb{A})$, the predicate $\mathcal{K}$, of arity 1, belongs to $\mathcal{P}$. This predicate is in particular used to model the knowledge of the attacker $\mathbb{A}$.

3.2 Semantics

Let s be a finite set of facts, and $\mathbf{I}$ be a finite set of Horn clauses. The *closure* of s under $\mathbf{I}$, denoted $\lceil s \rceil^{\mathbf{I}}$, is the smallest set that contains s and moreover $\forall (a \leftarrow a_1, \cdots, a_n) \in \mathbf{I}. \forall \sigma.\ a_1\sigma, \cdots, a_n\sigma \in \lceil s \rceil^{\mathbf{I}} \implies a\sigma \in \lceil s \rceil^{\mathbf{I}}$, where σ is a total (grounding) substitution function for the Horn clause $a \leftarrow a_1, \cdots, a_n$; that is $\sigma : (var(a) \bigcup_{1 \le i \le n} var(a_i)) \rightarrow \mathcal{T}_{\Sigma(\emptyset)}$. [1]

Let $((\Sigma, \mathcal{V}, \mathcal{P}), \Pi, \mathbb{A})$ be an architecture, consisting of services $\Pi = \{1, \cdots, \ell\}$, with $\pi = (\eta^0_\pi, \Omega^0_\pi, \mathbf{I}_\pi)$ for $\pi \in \Pi$, and $\mathbb{A} = (\Omega^0_{\mathbb{A}}, \mathbf{I}_{\mathbb{A}})$. A *configuration* of the architecture is a tuple $((\eta_1, \Omega_1), \cdots, (\eta_\ell, \Omega_\ell), \Omega_{\mathbb{A}})$, where η_i is a finite set of processes, for $1 \le i \le \ell$, and $\Omega_{\mathbb{A}}$ and Ω_i are finite sets of facts. The *initial* configuration of the architecture is $z^0 = ((\eta^0_1, \Omega^0_1), \cdots, (\eta^0_\ell, \Omega^0_\ell), \Omega^0_{\mathbb{A}})$.

Each architecture $((\Sigma, \mathcal{V}, \mathcal{P}), \Pi, \mathcal{A})$ is attributed with a Kripke structure which represents all the executions of the architecture. This is explained informally in the following. Suppose a process belonging to service $\pi \in \Pi$ can perform a guarded send event $g_1 \vee \cdots \vee g_k \blacktriangleright \mathsf{s}(t)$. Then, the guard is evaluated against the policy engine of π. The guard is interpreted as the "disjunction" of the "conjunctions" of atoms in each g_i, with $1 \le i \le k$. If the guard can be derived in the policy engine (i.e. the guard evaluates to "true"), then the process sends the term t to the network; that is, t is immediately added to the knowledge of the attacker. We remark that variables appearing in a guarded send event originate in previous receive events in the process. Therefore, the variables in term t have already been instantiated with ground values. Therefore, only messages (i.e. terms with no variables) are sent to the network. Now, suppose a process belonging to service $\pi \in \Pi$ can perform a receive event $\mathsf{r}(t) \rhd u$. If there exists a grounding substitution σ such that $t\sigma$ can be derived from the attacker's knowledge, then the process receives $t\sigma$ and the predicates $u\sigma$ are added to the knowledge set of service π. The formal definition is given below.

An architecture $((\Sigma, \mathcal{V}, \mathcal{P}), \Pi, \mathcal{A})$, with $\Pi = \{1, \cdots, \ell\}$, induces a Kripke structure (S, S^0, T), where S and T are the smallest sets satisfying

- $S^0 = z^0$, $z^0 \in S$.
- If $z = ((\eta_1, \Omega_1), \cdots, (\eta_i, \Omega_i), \cdots, (\eta_\ell, \Omega_\ell), \Omega_{\mathbb{A}}) \in S$, then
  - If $\mathfrak{e} \in \eta_i$ with $\mathfrak{e} = (g_1 \vee \cdots \vee g_k \blacktriangleright \mathsf{s}(t))\mathfrak{e}'$, $\mathfrak{e}' \in E^*$, and $\exists j.\ 1 \le j \le k \wedge g_j \subseteq \lceil \Omega_i \rceil^{\mathbf{I}_i}$, then $z' \in S$ and $(z, z') \in T$, with $z' = ((\eta_1, \Omega_1), \cdots, (\eta'_i, \Omega_i), \cdots, (\eta_\ell, \Omega_\ell), \Omega_{\mathbb{A}} \cup \{\mathcal{K}(t)\})$, and $\eta'_i = \eta_i \setminus \{\mathfrak{e}\} \cup \{\mathfrak{e}'\}$.
  - If $\mathfrak{e} \in \eta_i$ with $\mathfrak{e} = (\mathsf{r}(t) \rhd u)\mathfrak{e}'$, $\mathfrak{e}' \in E^*$, and there exists a substitution σ where $\mathcal{K}(t\sigma) \in \lceil \Omega_{\mathbb{A}} \rceil^{\mathbf{I}_{\mathbb{A}}}$, then $z' \in S$ and $(z, z') \in T$, where the configuration z' is defined as $z' = ((\eta_1, \Omega_1), \cdots, (\eta'_i, \Omega_i \cup u\sigma), \cdots, (\eta_\ell, \Omega_\ell), \Omega_{\mathbb{A}})$ and $\eta'_i = \eta_i \setminus \{\mathfrak{e}\} \cup \{\mathfrak{e}'\sigma\}$.

Given a configuration Z and Kripke structure (S, S^0, T), we say Z is *reachable* in (S, S^0, T) iff $(S^0, Z) \in T^*$, where T^* is the reflexive transitive closure of T.

3.3 The reachability decision problem

Given an architecture $\mathsf{arch} = ((\Sigma, \mathcal{V}, \mathcal{P}), \Pi, \mathbb{A})$, with $\Pi = \{1, \cdots, \ell\}$, and a fact $f \in \mathcal{A}_{\Sigma(\emptyset)}$, the *reachability* problem $\text{REACH}\langle \mathsf{arch}, a, f \rangle$, with $a \in \Pi \cup \{\mathbb{A}\}$, asks whether there exists a reachable configuration $((\eta_1, \Omega_1), \cdots, (\eta_\ell, \Omega_\ell), \Omega_{\mathbb{A}})$ in the Kripke structure induced by the architecture such that $f \in \lceil \Omega_a \rceil^{\mathbf{I}_a}$, or not. These cases are respectively denoted by $\text{REACH}\langle \mathsf{arch}, a, f \rangle = \mathsf{T}$ and $\text{REACH}\langle \mathsf{arch}, a, f \rangle = \mathsf{F}$.

The decision problem REACH subsumes the secrecy problem for security protocols and the safety problem for authorization logics. The secrecy problem asks whether the attacker can learn a (supposedly secret) message m via interacting with other services. This can be represented as $\text{REACH}\langle \mathsf{arch}, \mathbb{A}, \mathcal{K}(m) \rangle$. The safety problem asks whether an authorization predicate f (e.g. $knows(can\_read(ann, file12))$) can be derived by a service, say π, with $\pi \in \Pi$. This corresponds to $\text{REACH}\langle \mathsf{arch}, \pi, f \rangle$.

Due to the computational power of logic programs, REACH is in general undecidable (even though any architecture consists of finitely many services). It is easy to observe that REACH is semi-decidable if the *ground deduction* problem is decidable for the intensional knowledge sets of the participants and the attacker. The ground deduction problem for a finite set of Horn clauses $\mathbf{I}$, asks whether $a \in \lceil s \rceil^{\mathbf{I}}$, for an arbitrary fact a and a finite set of facts s.

4. A DECIDABLE FRAGMENT

In this section, we identify a fragment of intensional knowledge (for the participants and the attacker) that admits decision algorithms for the reachability problem. In this decidable fragment, referred to as $\mathbf{A}_1$, the intensional knowledge of the attacker is fixed to the standard Dolev-Yao (DY) deduction rules, as formalized in, e.g., [15]. The policies (i.e. intensional knowledge) of honest services are centered around *trust application TA*, and (transitive) *trust delegation TD* rules, adopted and adapted from DKAL, and can also express typical RBAC systems with role hierarchy. Next, we introduce the notion of *infons*, cf. [11, 12].

Infons are pieces of information, e.g. $can\_read(piet, file12)$ stipulating that Piet can read a certain *file12*. An infon does not admit a truth value, i.e. it is never false or true. Instead, infons are the interfaces between the communication level and policy level for honest services. That is, if the policy engine of a service, say Ann, derives the predicate $knows(can\_read(piet, file12))$, then Ann "knows" that Piet may read this file, and may thus grant him read access to *file12*. Note that "knows" in this context, and also in DKAL,

[1]The existence of $\lceil s \rceil^{\mathbf{I}}$ follows immediately from Knaster-Tarski's fixed point theorem.

is a predicate symbol and not a modality as in logics of knowledge. In fact, "knows" here is closer to the notion of *belief* rather than *knowledge*, in epistemic terms.

Infons are different from predicates (i.e. policy statements) in that they can be *nested*, i.e. infons are constructed by applying *infon constructors* to message terms and other infons. We assume that in any signature $(\Sigma, \mathcal{V}, \mathcal{P})$, the set Σ can be partitioned into Σ_{msg} and Σ_{infon}, so that Σ_{infon} is the set of infon constructor functions, and Σ_{msg} is the set of message constructor functions. The set of infons *Infons* is formally defined as the smallest set satisfying the property: if $t_1, \cdots, t_n \in \mathcal{T}_{\Sigma_{msg}(\mathcal{V})} \cup Infons$ and $f \in \Sigma_{infon}$ with arity of f being n, then $f(t_1, \cdots, t_n) \in Infons$. Note that $Infons \cap \mathcal{T}_{\Sigma_{msg}(\mathcal{V})} = \emptyset$; in particular messages are not infons.

We assume that for honest services the policy statements (i.e. the inhabitants of the policy level) are predicates over *Infons*. That is, for these services, the knowledge ranges over pieces of information. In contrast, the knowledge of the attacker ranges directly over message terms. Below, disjoint union of two sets is denoted by $\sqcup$.

Definition 1. *Fragment* $\mathbf{A}_1$ *consists of all architectures* $\mathsf{arch} = ((\Sigma, \mathcal{V}, \mathcal{P}), \Pi, \mathbb{A})$, *which satisfy the following syntactical conditions:*

- $(\Sigma, \mathcal{V}, \mathcal{P})$ *is a signature, with* $\Sigma = \Sigma_{msg} \sqcup \Sigma_{infon}$, *and:*
  - *A finite subset of constants in* Σ_{msg}, *denoted by Agents, represents the set of the names of the services in* Π. [2]
  - *Apart from nullary functions (i.e. constants),* Σ_{msg} *only contains the functions* $\{\cdot\}_\cdot$, $\{\!|\cdot|\!\}_\cdot$, $sig(\cdot,\cdot)$, $pk(\cdot)$, $h(\cdot)$, $(\cdot,\cdot)$. *These represent respectively asymmetric and symmetric encryption, digital signature, public key constructor, hash and pairing functions, interpreted as usual.*
  - Σ_{infon} *contains in particular the functions* $\theta(\cdot,\cdot)$ *and* $\sigma(\cdot,\cdot)$. *These intuitively stand for* trusted on *and* said, *respectively, with* $\theta, \sigma : Agents \times Infons \to Infons$.
  - $\mathcal{P} = \{\mathcal{K}\}$, *with* $\mathcal{K}$ *being a unary predicate. Intuitively,* $\mathcal{K}$ *stands for "knows".*
- *Any service* $\pi = (\eta^0_\pi, \Omega^0_\pi, \mathbf{I}_\pi)$ *in* Π *meets the conditions:*
  - *For all processes in* η^0_π, *all the terms sent and received are elements of* $\mathcal{T}_{\Sigma_{msg}(\mathcal{V})}$.
  - Ω^0_π *is a finite set of ground atoms of the form* $\mathcal{K}(i)$, *with* $i \in Infons$.
  - $\mathbf{I}_\pi$ *includes the TA and TD rules, respectively represented by* $\mathcal{K}(X) \leftarrow \mathcal{K}(\theta(A,X)), \mathcal{K}(\sigma(A,X))$, *and* $\mathcal{K}(\theta(A,\theta(B,X))) \leftarrow \mathcal{K}(\theta(A,X))$. *The set of all the other rules in* $\mathbf{I}_\pi$ *constitutes a* type-1 theory, *as defined below, and* σ *and* θ *do not appear in this set.*
- $\mathbb{A} = (\Omega^0_{\mathbb{A}}, \mathbf{I}_{\mathbb{A}})$, *where* $\Omega^0_{\mathbb{A}}$ *is a finite subset of* $\{\mathcal{K}(t) \mid t \in \mathcal{T}_{\Sigma_{msg}(\emptyset)}\}$, *and* $\mathbf{I}_{\mathbb{A}}$ *consists of the rules that reflect the capabilities of the standard DY attacker as formalized in appendix A or [5]; for instance,* $\mathcal{K}(X) \leftarrow K((X,Y))$, *and* $\mathcal{K}(\{\!|X|\!\}_Y) \leftarrow \mathcal{K}(X), \mathcal{K}(Y)$.

[2]Intuitively, one (or more) public key is attributed to each element of *Agents*. The public keys are known to everyone, and to the attacker in particular. Using the public key of $a \in Agents$ one can encrypt messages for a and can verify the authenticity of the messages signed by a.

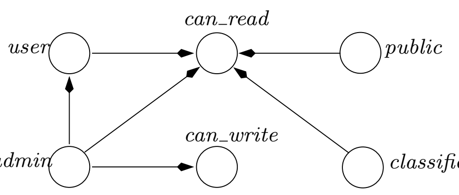

Figure 1: Dependency graph, example 1.

We define type-1 theories in order to extend the policies of (honest) services beyond *TA* and *TD*, e.g. to express typical RBAC systems with role hierarchy.

Definition 2. *A finite set of Horn clauses T, defined over signature* $(\Sigma, \mathcal{V}, \mathcal{P})$, *with* $\Sigma = \Sigma_{msg} \sqcup \Sigma_{infon}$, *is called a* type-1 *theory, iff*

(a) All clauses in T have the form $p(t) \leftarrow p_1(t_1), \cdots, p_\ell(t_\ell)$, *where* $p, p_1, \cdots, p_\ell \in \mathcal{P}$, *and* $t, t_1, \cdots, t_\ell \in Infons$.

(b) For all $a \leftarrow a_1, \cdots, a_\ell$ *in T*, $\bigcup_{i \in \{1,\cdots,\ell\}} var(a_i) \subseteq var(a)$.

(c) The infon dependency graph *of T is acyclic. The infon dependency graph of T is a directed graph defined by the pair* $(\Sigma_{infon}, Edges)$ *with* $(f,g) \in Edges$ *iff there exists a Horn clause in T using g in its head, and f in its body.*

We remark that neither *TA* nor *TD* fall into type-1 theories, due to conditions (b) and (c) in definition 2, respectively.

Example 1. Consider a file server which implements an RBAC system with two roles, *user* and *admin*. Users may read any *public* file, admins may read any *classified* file, and admins may also write to any file. Admins inherit all the rights attributed to users. Below, we give a type-1 theory which describes this RBAC system.

$$\begin{array}{lcl}
\mathcal{K}(user(A)) & \leftarrow & \mathcal{K}(admin(A)) \\
\mathcal{K}(can\_read(A,F)) & \leftarrow & \mathcal{K}(user(A)), \mathcal{K}(public(F)) \\
\mathcal{K}(can\_read(A,F)) & \leftarrow & \mathcal{K}(admin(A)), \mathcal{K}(classified(F)) \\
\mathcal{K}(can\_write(A,F)) & \leftarrow & \mathcal{K}(admin(A))
\end{array}$$

Here Σ_{msg} contains the set of identities of involved services, and names of files, while $\mathcal{P} = \{\mathcal{K}\}$, and $\Sigma_{infon} = \{user, admin, public, classified, can\_read, can\_write\}$ with obvious arities. The infon dependency graph for this theory, shown in figure 1, is indeed acyclic. •

The type-1 policy of examples 1 is *centralized* in the sense that it describe the policies of a single entity, i.e. the file server. In the context of an architecture, this type-1 theory may represent the policies of a single service, cf. section 4.1.

In section 5, we give a decision algorithm for the reachability problem for architectures in fragment $\mathbf{A}_1$. The decision algorithm is based upon encoding policy level computations of services into message derivation trees of the standard DY model. Let us now continue with a formal specification of CRP (section 2) as an $\mathbf{A}_1$ architecture.

4.1 A formal specification of CRP

We give a formal specification of CRP by defining the architecture $\mathsf{crp} = ((\Sigma, \mathcal{V}, \mathcal{P}), \Pi, \mathbb{A})$. The architecture, as we will see, indeed falls into the $\mathbf{A}_1$ fragment. The signature Σ contains the standard cryptographic primitives $\{\cdot\}_\cdot$, $\{\!|\cdot|\!\}_\cdot$, $pk(\cdot)$, $h(\cdot)$, $sig(\cdot,\cdot)$, $(\cdot,\cdot) \in \Sigma_{msg}$, and the infon constructors, namely $can\_store(\cdot), empl(\cdot), head(\cdot), \sigma(\cdot,\cdot), \theta(\cdot,\cdot) \in \Sigma_{infon}$. The set of constants $\Sigma_C \subset \Sigma_{msg}$ used in crp is defined as $\Sigma_C = \{mike, piet, ann, cr, hr, eve, doc, \mathsf{empl\_status}, \mathsf{is\_empl}, \mathsf{delegated\_to}, \mathsf{success\_token}\}$ with eve being the identity of the attacker.

The set $\mathcal{P}$ contains only one unary predicate, $\mathcal{K}$, used for modelling the knowledge of the services and the attacker. Note that the knowledge of each of the services and the attacker are stored separately. In order to avoid unnecessary cluttering, we suppress the predicate symbol $\mathcal{K}$.

The attacker $\mathbb{A} = (\Omega^0_{\mathbb{A}}, \mathbf{I}_{\mathbb{A}})$ has the initial knowledge $\Omega^0_{\mathbb{A}} = \Sigma_C \setminus \{doc\} \cup \{pk(A) \mid A \in \{mike, piet, ann, cr, hr, eve\}\}$, and her intensional knowledge reflects the usual DY message derivation rules. Despite the fact that the attacker's knowledge set is always finite, she can generate an infinite set of terms by pairing, hashing, signing and encrypting the terms that are known to her (cf. appendix A).

The intensional knowledge for $mike$, $piet$, and ann consists of the TA and TD rules only. The intensional knowledge of the hr contains the type-1 theory $\{empl(X) \leftarrow head(X)\}$, besides TA and TD. The intensional knowledge of the cr, in addition to TA and TD contains the type-1 theory $\{empl(X) \leftarrow head(X), can\_store(X) \leftarrow empl(X)\}$, which constitute a simple hierarchical RBAC system, where employees have the right to store documents in the cr, and heads of the office inherit all rights of employees.

Below, we describe the processes executed by the participating services, and their initial knowledge. Recall that capital letters denote variables. To avoid name clashes, variables appearing in different processes should be tagged with process names. The tagging is however omitted in the following to ease the presentation.

Citizen (*mike*). Mike's initial knowledge is empty.

$$\emptyset \blacktriangleright \mathsf{s}(\{mike, doc\}_{pk(piet)}, [h(doc)]_{mike})$$
$$\mathsf{r}([h(mike, doc), \mathsf{success\_token}]_{piet}) \triangleright \emptyset$$

Mike sends document doc to Piet, in order to be stored in the cr. He then waits to receive a "success" token from Piet. The names $mike$ and $piet$, and also doc are constants in Mike's specification.

Employee (*piet*). Piet's initial knowledge is empty.

$$\mathsf{r}(\{C, D\}_{pk(piet)}, [h(D)]_C) \triangleright \emptyset$$
$$\emptyset \blacktriangleright \mathsf{s}(\{C, D\}_{pk(cr)}, [ann, h(C, D)]_{piet})$$
$$\mathsf{r}([h(C, D)]_{cr}) \triangleright \emptyset$$
$$\emptyset \blacktriangleright \mathsf{s}([h(C, D), \mathsf{success\_token}]_{piet})$$

After Piet receives a request, from a citizen C, to store a document D, he sends a corresponding request to the cr. Then he waits for confirmation (of successful storing) from the cr, after which he notifies the citizen on the completed transaction.

Head (*ann*). Ann's initial knowledge is empty.

$$\mathsf{r}(E, \mathsf{empl\_status}) \triangleright \emptyset$$
$$\emptyset \blacktriangleright \mathsf{s}([E, \mathsf{is\_empl}, \mathsf{delegated\_to}, hr]_{ann})$$

When Ann receives a request for information on the status of a principal E, she replies that the task of providing such information has been delegated to the hr. Ann would typically execute a few instances of this process in parallel.

Human Resources (*hr*). The initial knowledge of hr contains all employees and heads of the office. That is, the hr's initial knowledge is $\{empl(piet), head(ann)\}$.

$$\mathsf{r}(E, \mathsf{empl\_status}) \triangleright \emptyset$$
$$\{empl(E)\} \blacktriangleright \mathsf{s}([E, \mathsf{is\_empl}]_{hr})$$

After receiving a request for the status of a principal E, the hr confirms that E is an employee of the office by sending the message $[E, \mathsf{is\_empl}]_{hr}$, only if $empl(E)$ can be derived in the policy engine of the hr.

Central Repository (*cr*). The cr service consists of two processes, executed in parallel. The initial knowledge of the cr is $\{head(ann)\}$.

Central Repository's main process.

$$\begin{array}{rcl} \mathsf{r}(\{C, D\}_{pk(cr)}, [H, h(C, D)]_E) & \triangleright & \{\theta(H, empl(E))\} \\ \{head(H)\} & \blacktriangleright & \mathsf{s}(E, \mathsf{empl\_status}) \\ \mathsf{r}([E, \mathsf{is\_empl}]_F) & \triangleright & \{\sigma(F, empl(E))\} \\ \{can\_store(E)\} & \blacktriangleright & \mathsf{s}([h(C, D)]_{cr}) \end{array}$$

After receiving the first message from E, requesting to store a document D as an employee of the office headed by H, the cr asserts that H is trusted on whether E is an employee of the office, or not. Next, if H is indeed the head of the office, then the cr asks H for clarifying the employment status of E. If the cr, after receiving the third message, ascertains that E has the right to store, then the document is stored (not formalized here), and E is notified.

Central Repository's delegation handler.

$$\mathsf{r}([E, \mathsf{is\_empl}, \mathsf{delegated\_to}, HR]_H) \triangleright \{\sigma(H, \theta(HR, empl(E)))\}$$

This process receives, independently of the other process,

$$\dfrac{\dfrac{\dfrac{\dfrac{\theta(ann, empl(piet))}{\theta(ann, \theta(hr, empl(piet)))}\,\text{TD} \quad \sigma(ann, \theta(hr, empl(piet)))}{\theta(hr, empl(piet))}\,\text{TA} \quad \sigma(hr, empl(piet))}{empl(piet)}\,\text{TA}}{can\_store(piet)}\,\text{type-1}$$

Figure 2: A derivation tree for the CRP case study

messages from an office head H to delegate to the HR the right to declare the employment status of E.

The formalization given above models the inquiries which the cr conducts via a "broadcast" send. Namely, when the cr sends the message $(E, \mathsf{empl\_status})$, the message is intended for H (i.e. ann), but it is received also by F (i.e. the hr). This is because ann replies to the message with a delegation certificate, which is consumed by the delegation handler process of the cr. It is thus F (i.e. the hr) who actually responds to the message $(E, \mathsf{empl\_status})$ in the cr's main process.

The derivation tree of figure 2 shows how the cr ascertains Piet's right to write into the repository; the correspondence between derivation trees and computing closures under Horn clauses is immediate. The rules (e.g. TA or type-1 theory of the cr) used in each derivation step are also shown in the figure.

5. DECIDING REACHABILITY

In this section, we give an algorithm for deciding Reach in $\mathbf{A}_1$ architectures. We start with presenting an encoding from policy statements into (message) terms. Then, we extend the constraint-solving algorithm of Millen and Shmatikov for deciding reachability in the "encoded" $\mathbf{A}_1$ architectures.

The purpose of the encoding is to replace the logic programs of the participating services in an $\mathbf{A}_1$ architecture with the derivation rules of the DY model. Then, intuitively, the attacker and all the services would be equipped with the reasoning power of the DY model, which is well understood and comes with decision algorithms for reachability.

5.1 Encoding policy level computations

Below, to simplify the presentation, we suppress the predicate symbol $\mathcal{K}$ from facts, and work directly with infons; indeed $\mathcal{K}$ is the only predicate symbol in $\mathbf{A}_1$ architectures. The encoding consists of two functions: ζ which maps infons to $\mathcal{T}_{\Sigma(\mathcal{V})}$, and $\mathcal{E}$ which maps infons to guards. We start with an initial encoding for trust application only. Then, we extend the encoding to trust delegation and type-1 theories.

TA only. We recursively define the encoding for infon i:

$$\zeta(i) = \begin{cases} \{\!|\zeta(X), sig(\bar{A}, \zeta(X))|\!\}_{\zeta(\theta(A,X))} & \text{if } i = \sigma(A, X) \\ \theta(\bar{A}, \zeta(X)) & \text{if } i = \theta(A, X) \\ i & \text{otherwise} \end{cases}$$

where $\bar{\cdot} : Agents \to \overline{Agents}$ is a bijection which associates a unique name to each element of $Agents$. Elements of $\overline{Agents}$ belong to $\mathcal{T}_{\Sigma_{msg}(\emptyset)}$, and are defined solely for the encoding function ζ, i.e. they do not appear in the specifications of architectures. In particular, the attacker does not have the private keys associated to the members of $\overline{Agents}$.

Here, the encoding of the infon $\sigma(a, x)$ is the ciphertext $\{\!| x, sig(\bar{a}, x)|\!\}_{\theta(\bar{a},x)}$ from which the infon x (i.e. what service a said) can be obtained using the *symmetric decryption* ($Sdec$) rule of DY only if the key $\theta(\bar{a}, x)$ (i.e. a is trusted on x) is obtained first. This indicates that if TA is applicable on a set of facts at the policy level, then the DY rule

$$\mathcal{K}(X) \leftarrow \mathcal{K}(\{\!|X|\!\}_K), \mathcal{K}(K) \quad Sdec$$

is applicable to the terms resulting from the encoding. Intuitively, the role of sig is to ensure that terms of the form $\{\!| x, sig(\bar{a}, x)|\!\}_{\theta(\bar{a},x)}$ can be constructed using the DY rules only if a corresponding $\sigma(a, x)$ can be derived in the policy level. Recall that the attacker does not know the private key of $\bar{a}$.

TA, TD and type-1 theories. In order to include TD and type-1 theories in the encoding, we define an *expansion* function $\mathcal{E}$ that for any atom returns a guard. We motivate the expansion function via a simple example. Suppose the fact $\theta(a, i)$ is present in the policy engine of a service, and the guard $\theta(a, \theta(b, i))$ is to be evaluated. The TD rule implies that the guard can be derived, while there is no corresponding inference tree (in the DY model) for $\zeta(\theta(a, \theta(b, i)))$, given $\zeta(\theta(a, i))$. The set of infons which yield $\theta(a, \theta(b, i))$ via applying only the TD rule is however finite. This finite set of infons can be seen as a guard, namely $\{\theta(a, \theta(b, i))\} \vee \{\theta(a, i)\}$. The fact that $\theta(a, i)$ yields $\theta(a, \theta(b, i))$ in the policy engine is reflected in the DY model by: either $\zeta(\theta(a, \theta(b, i)))$ or $\zeta(\theta(a, i))$, or both, are obtained from $\zeta(\theta(a, i))$.

The expansion function $\mathcal{E}_P(Q, i)$ is defined for finite sets of Horn clauses P and Q, and infon i:

$$\begin{aligned} &\mathcal{E}_P(\emptyset, i) = \{i\} \\ &\mathcal{E}_P(\{r \leftarrow r_1, \ldots, r_\ell\} \sqcup Q', i) = \\ &\quad \begin{cases} \mathcal{E}_P(Q', i) \vee & \\ \qquad \big(\mathcal{E}_P(P, r_1\rho) \cup \cdots \cup \mathcal{E}_P(P, r_\ell\rho)\big) & \text{if } i = r\rho \\ \mathcal{E}_P(Q', i) & \text{if } \neg\exists\rho.\ i = r\rho \end{cases} \end{aligned}$$

where $\cup$ distributes over $\vee$, i.e. $S \cup (S_1 \vee S_2) = (S_1 \vee S_2) \cup S = (S_1 \cup S) \vee (S_2 \cup S)$. Here Q is a support theory used only to ensure that the expansion of any infon results in a finite set; see lemma 1 below.

Lemma 1. *Let $P = P^1 \cup \{TD\}$, with P^1 being the type-1 theory in a service of an $\mathbf{A}_1$ architecture. Then, $\mathcal{E}_P(P, i)$ is a finite set for any infon i.*

PROOF. Immediate, since the dependency graph of P^1 is acyclic, P^1 does not contain θ, and the Horn clause which encodes TD strictly decreases the number of θ functions. $\square$

We write $\mathcal{E}(i)$ for $\mathcal{E}_P(P, i)$, when P is clear from the context. We write $g \Subset_\vee \mathcal{E}(i)$ if $\mathcal{E}(i) = g_1 \vee \cdots \vee g \vee \cdots \vee g_n$, with $n \geq 1$.

EXAMPLE 2. Consider the infon $i = \mathit{can\_read}(a, \mathit{file})$ along with the type-1 theory of example 1. Then,

$$\begin{aligned}\mathcal{E}(i) = \{&\mathit{user}(a), \mathit{public}(\mathit{file})\} \vee \{\mathit{admin}(a), \mathit{public}(\mathit{file})\} \vee \\ \{&\mathit{admin}(a), \mathit{classified}(\mathit{file})\} \vee \{\mathit{can\_read}(a, \mathit{file})\}\end{aligned}$$

Write $\mathcal{E}(i) = g_1 \vee g_2 \vee g_3 \vee g_4$, with $g_1 = \{\mathit{user}(a), \mathit{public}(\mathit{file})\}$, etc. The guard $\mathcal{E}(i)$ is interpreted as: $\mathit{can\_read}(a, \mathit{file})$ holds, i.e. a can read *file* in example 1, iff at least one of the following conditions holds: $[g_1]$ $\mathit{user}(a)$ and $\mathit{public}(\mathit{file})$ are known, or $[g_2]$ $\mathit{admin}(a)$ and $\mathit{public}(\mathit{file})$ are known, or $[g_3]$ $\mathit{admin}(a)$ and $\mathit{classified}(\mathit{file})$ are known, or $[g_4]$ $\mathit{can\_read}(a, \mathit{file})$ is known via an inference outside the RBAC system of example 1. Similarly, we get $\mathcal{E}(\mathit{can\_write}(a, \mathit{file})) = \{\mathit{admin}(a)\} \vee \{\mathit{can\_write}(a, \mathit{file})\}$. •

We refine the function ζ (introduced above) by incorporating the expansion function $\mathcal{E}$ into ζ. This intuitively ensures that $\zeta(i)$, for infon i, is obtainable from $\zeta(\sigma(a, i))$ if there exist at least one $g \Subset_\vee \mathcal{E}(\theta(a, i))$ such that $\zeta(g)$ can be obtained first. Hence, we define:

$$\zeta(i) = \begin{cases} \{\!|\zeta(X), \mathit{sig}(\bar{A}, \zeta(X))|\!\}_{\zeta(\mathcal{E}_P(P, \theta(A, X)))} & \text{if } i = \sigma(A, X) \\ \theta(\bar{A}, \zeta(X)) & \text{if } i = \theta(A, X) \\ i & \text{otherwise} \end{cases}$$

Here, $\{\!|\, x \,|\!\}_{k_1 \vee \cdots \vee k_\ell}$ stands for the tuple $\{\!|\, x \,|\!\}_{k_1}, \cdots, \{\!|\, x \,|\!\}_{k_\ell}$, function ζ distributes over $\vee$, and $P = P^1 \cup \{TD\}$ with P^1 being the type-1 theory at hand. For a finite set of infons g, $\zeta(g)$ is defined as the concatenation of $\zeta(i)$, for all $i \in g$. We remark that elements of $\mathcal{E}_P(P, \theta(A, X))$ in the definition of ζ are singletons. This is because in any $\mathbf{A}_1$ architecture, $\mathcal{E}_P(P, \theta(a, i)) = \mathcal{E}_{\{TD\}}(\{TD\}, \theta(a, i))$, as θ does not appear in P^1.

Correctness of the encoding. We remark that the purpose of the proposed encoding is to replace the logic programs of services with the derivation rules of the Dolev-Yao model.

The following theorem ensures that if a policy fact is derivable in the logic program of a service, then its corresponding encoded term can be derived using the DY inference rules, and vice versa. We consider the standard DY capabilities for the term algebra $\mathcal{T}_{\Sigma(\mathcal{V})}$, which comprises both infon and message constructors. That is, the infon constructors are seen as uninterpreted functions, while message constructors (e.g. $\{\!|\cdot|\!\}_\cdot$) have their standard meaning in the DY model.

In the rest of the paper, $T \vdash u$ denotes that u is derivable from a set of terms T with the standard DY inference rules as formalized, e.g., in appendix A and [15].

THEOREM 1. *Let P be the intensional knowledge of a service in an $\mathbf{A}_1$ architecture, with $P = \{TA\} \cup Q$, $Q = \{TD\} \cup P^1$ and P^1 being a type-1 theory. For any (ground) infon f and finite set of (ground) infons G,*

$$\mathcal{K}(f) \in \lceil \mathcal{K}(G) \rceil^P \iff \exists g \Subset_\vee \mathcal{E}_Q(Q, f). \quad \zeta(G) \vdash \zeta(g),$$

where $\mathcal{K}(G)$ stands for the set $\{\mathcal{K}(f_i) \mid f_i \in G\}$.

PROOF. Fix the policy set P. We write $G \Vdash f$ for $\mathcal{K}(f) \in \lceil \mathcal{K}(G) \rceil^P$, suppress $\mathcal{K}$ when confusion is unlikely, and write $\mathcal{E}(f)$ for $\mathcal{E}_Q(Q, f)$. Below, we talk about *proof trees* for f, given G. The correspondence between finding proof trees and computing closures is immediate. The proof is split into two directions.

$\Rightarrow$ We use structural induction on proof trees for f, given G. If $f \in G$, then the implication is trivial. Otherwise, consider the last rule applied in the proof tree:

- (TA) Then $G \Vdash \sigma(a, f), \theta(a, f)$, for some $a \in \mathit{Agents}$. By induction hypotheses,

  $\exists s \Subset_\vee \mathcal{E}(\sigma(a, f)), t \Subset_\vee \mathcal{E}(\theta(a, f)).\ \zeta(G) \vdash \zeta(s), \zeta(t).$

  Observe that $s = \sigma(a, f)$. The term $\zeta(\sigma(a, f))$ is the tuple $\{\!|\zeta(f), \mathit{sig}(\bar{a}, \zeta(f))|\!\}_{\mathcal{E}(\theta(a,f))}$. Since $t \Subset_\vee \mathcal{E}(\theta(a, f))$, through unpairing, we obtain the ciphertext $\{\!|\zeta(f), \mathit{sig}(\bar{a}, \zeta(f))|\!\}_{\zeta(t)}$ from $\zeta(\sigma(a, f))$. From $\zeta(G) \vdash \zeta(t)$, by applying the *Sdec* rule and unpairing we get $\zeta(G) \vdash \zeta(f)$. Clearly $f \Subset_\vee \mathcal{E}(f)$.
- (TD) Then $f = \theta(a, \theta(b, i))$ for some $a, b \in \mathit{Agents}$ and $i \in \mathit{Infons}$, and $G \Vdash \theta(a, i)$. By induction hypotheses, $\exists t \Subset_\vee \mathcal{E}(\theta(a, i))$. $\zeta(G) \vdash \zeta(t)$. Now, the claim follows since for any infon t, $t \Subset_\vee \mathcal{E}(\theta(a, i))$ implies $t \Subset_\vee \mathcal{E}(\theta(a, \theta(b, i)))$.
- (Type-1) Let $R = r \leftarrow r_1, \ldots, r_\ell \in P^1$ be the last rule applied. Then $f = r\rho$ and $G \Vdash r_1\rho, \cdots, r_\ell\rho$, for some grounding substitution ρ (cf. condition (b) in definition 2). By induction hypotheses, $\exists r'_1 \Subset_\vee \mathcal{E}(r_1\rho), \cdots, r'_\ell \Subset_\vee \mathcal{E}(r_\ell\rho)$. $\zeta(G) \vdash \zeta(r'_1), \cdots, \zeta(r'_\ell)$. By definition of $\mathcal{E}$, $\{r'_1, \cdots, r'_\ell\} \Subset_\vee \mathcal{E}(f)$, hence follows the claim.

$\Leftarrow$ First, we claim that $\zeta(G) \vdash \zeta(g)$ implies $G \Vdash g$. Notice that the $\zeta(g)$ is either of the form $\{\!|\zeta(x), \mathit{sig}(\bar{a}, \zeta(x))|\!\}_{\mathcal{E}(\theta(a,x))}$, or of the form $i(x)$, with i being an infon constructor. The claim follows by case analysis on the DY attacker's message (de)composition abilities. In particular, note that (1) to fabricate $\{\!|\zeta(x), \mathit{sig}(\bar{a}, \zeta(x))|\!\}_{\mathcal{E}(\theta(a,x))}$, the attacker needs to construct $\mathit{sig}(\bar{a}, \zeta(x))$, which is impossible as the attacker does not own the private key for any $\bar{a} \in \overline{\mathit{Agents}}$, and (2) infon constructors are uninterpreted functions in the DY model, i.e. they can neither be applied by the attacker, nor their application can be deconstructed. The other cases are straightforward; we thus omit them here. Finally, notice that if $G \Vdash g$ and $g \Subset_\vee \mathcal{E}(f)$, then $G \Vdash f$; hence follows the claim.

This completes our proof. $\square$

5.2 A constraint-solving algorithm for deciding reachability

We begin with a brief description of Millen and Shmatikov's constraint solving algorithm for deciding reachability in cryptographic protocols [15]. Recall that participants are specified as sequences of communication (i.e. send and receive) events in [15], and the reachability problem, given a message s, asks whether there exists a reachable configuration z of the protocol where $T(z) \vdash s$, with $T(z)$ being the attacker's knowledge in z.

The algorithm of [15] searches the finite set of interleavings of (symbolic) actions performed by the participants and a fictitious *test* process, which receives s and then sends stop to the search algorithm. For each interleaving, a sequence C of *attacker constraints* is constructed. An attacker constraint is a pair $\langle m : T\rangle$, where m is a term that the attacker should derive from the set of terms T, using her inference capabilities. The constraint sequence is built for each interleaving as: when a participant sends a message term, the term is added to the attacker term set, and when a receive action occurs, a constraint $\langle m : T\rangle$ is enqueued to C, with m being the term that is to be received and T is the current attacker term set.

A solution for constraint sequence $C = \langle m_1 : T_1\rangle \cdots \langle m_i : T_i\rangle \cdots \langle m_n : T_n\rangle$, with $m_i = s$, is a (grounding) substitution $\sigma : var(\langle m_1 : T_1\rangle \cdots \langle m_i : T_i\rangle) \to \mathcal{T}_{\Sigma(\emptyset)}$ such that $T_j\sigma \vdash m_j\sigma$, for $1 \le j \le i$; here, $var(c_1 \cdots c_i)$ is the set of variables appearing in the constraints $c_1, \cdots, c_i$. In our presentation, therefore, we account for *partial* executions as well, cf. [6]. Millen and Shmatikov's algorithm applies a number of reduction rules which reduce C to a sequence of immediately (un)satisfiable constraints. In the following, we refer to their reduction procedure as $\mathsf{MSReduce}$. If $\mathsf{MSReduce}$ does not succeed for C (i.e. C is unsatisfiable), next interleaving is considered, until all the interleavings are exhausted. If one of the constraint sequences is satisfiable, then the (supposedly) secret message s is revealed to the attacker. That is, an attack is found. Otherwise the protocol is correct, i.e. s is not revealed to the attacker, for the instantiation at hand.

The following two observations enable us to use Millen and Shmatikov's procedure (with minor extensions) for deciding reachability for $\mathbf{A}_1$ architectures:

1. Checking guards in, and making updates to, policy engines of the services can be emulated by communication actions. Namely, sending an infon to the knowledge set of a service reflects updating the policy engine of that service, while receiving an infon derived from the knowledge set reflects querying the policy engine of the service for evaluating a guard.

2. The encoding presented in section 5.1 entails that the same inference rules which are used for the attacker (namely the standard DY message derivation capabilities) can model the computations of the participants at their policy level. Therefore, the send and receive actions which would emulate checking guards and updating knowledge sets of services (mentioned above) can be treated with the constraint solving procedure that one would use for the attacker knowledge (here, $\mathsf{MSReduce}$).

Algorithm 1 Constraint solving for deciding reachability in $\mathbf{A}_1$ fragment

```
REQUIRES: $\textsc{Reach}\langle \mathsf{arch}, a, f\rangle$, $\mathsf{arch} = ((\Sigma, \mathcal{V}, \mathcal{P}), \Pi, \mathbb{A})$
  $expand(\Pi)$
  $I := interleavings(\Pi, a, f)$
  for all $\iota \in I$ do
    $flatten(\iota)$
    $C := constraint\_sequence(\iota)$
    $trace := \mathsf{MSReduce}^{\bowtie}(C)$
    if $trace \neq \emptyset$ then
      return (reach : $trace$)
  return (unreach)
```

Algorithm 1 details our constraint solving procedure for deciding reachability in $\mathbf{A}_1$ architectures. The input to the algorithm is a reachability problem $\textsc{Reach}\langle \mathsf{arch}, a, f\rangle$, with $\mathsf{arch} = ((\Sigma, \mathcal{V}, \mathcal{P}), \Pi, \mathbb{A})$ being an $\mathbf{A}_1$ architecture. The algorithm either returns a *trace* witnessing $\textsc{Reach}\langle \mathsf{arch}, a, f\rangle = \mathsf{T}$, or returns *unreach* if $\textsc{Reach}\langle \mathsf{arch}, a, f\rangle = \mathsf{F}$.

In algorithm 1, $expand(\Pi)$ expands the guards in each process, for all the services in Π. A guard $g_1 \vee \cdots \vee g_n$ in a service with policy P, is expanded to $\bigvee_{1\le j\le n} \mathcal{E}_Q(Q, g_j)$, where $Q = P \setminus \{TA\}$ and $\mathcal{E}_Q(Q, g)$ for a finite set of infons g is defined as the union of $\mathcal{E}_Q(Q, i)$, for all $i \in g$. Recall that $\cup$ distributes over $\vee$ (cf. section 5.1). The *expand* procedure thus rewrites guards into guards. For example, the guard $\{a, b\}$ in a service with intensional knowledge $\{TA, TD, a \leftarrow c, b \leftarrow d\}$ is expanded to $\{a, b\} \vee \{c, b\} \vee \{a, d\} \vee \{c, d\}$.

The procedure $interleavings(\Pi, a, f)$ computes the finite set of interleavings of events of the participants in Π. Furthermore, this procedure **(1)** adds a *testing* process whose sole purpose is to indicate that the search has reached a configuration in which $f \in \lceil \Omega(a) \rceil^{\mathbf{I}_a}$, given $\textsc{Reach}\langle \mathsf{arch}, a, f\rangle$; that is, if $a = \mathbb{A}$ and $f = \mathcal{K}(m)$, then the testing process simply receives m and then sends stop to the search algorithm. If $a \neq \mathbb{A}$, then the testing process $\{f\} \blacktriangleright \mathsf{s}(\mathsf{stop})$ is added to service a. **(2)** The *interleavings* procedure treats the disjunction operator $\vee$ inside guards as branching points, e.g. the interleaving $\mathfrak{e}_1 \cdot (g \vee g' \blacktriangleright \mathsf{s}(m)) \cdot \mathfrak{e}_2$, with $\mathfrak{e}_1, \mathfrak{e}_2 \in E^*$, gives rise to two interleavings $\mathfrak{e}_1 \cdot (g \blacktriangleright \mathsf{s}(m)) \cdot \mathfrak{e}_2$ and $\mathfrak{e}_1 \cdot (g' \blacktriangleright \mathsf{s}(m)) \cdot \mathfrak{e}_2$. Consequently, no $\vee$ appears in guards for any interleaving $\iota \in I$ in algorithm 1.

For each interleaving, the procedure *flatten* intuitively "flattens" the two levels of specification into one. That is, a sequence of events (i.e. guarded sends, and receives coupled with updates) is translated into a sequence of *annotated* send and receive actions. The annotations indicate the knowledge set with which the communication is carried out: $\mathbb{A}$ for network communications through the attacker, and $\pi \in \Pi$ for each service π.

$$\begin{array}{lll} (\{a_1, \ldots, a_\ell\} \blacktriangleright \mathsf{s}(m)) & \text{maps to} & \mathsf{r}^{\pi}(\zeta(a_1), \cdots, \zeta(a_\ell)) \cdot \mathsf{s}^{\mathbb{A}}(m) \\ (\mathsf{r}(m) \rhd \{a_1, \ldots, a_\ell\}) & \text{maps to} & \mathsf{r}^{\mathbb{A}}(m) \cdot \mathsf{s}^{\pi}(\zeta(a_1), \cdots, \zeta(a_\ell)) \end{array}$$

The annotated send and receive actions are used in the reduction procedure $\mathsf{MSReduce}^{\bowtie}$, which is a barely syntactic modification of $\mathsf{MSReduce}$. The modification consists in allocating one term set for each service in Π, and one set for the attacker. This is in contrast to $\mathsf{MSReduce}$ where only one term set, denoting the attacker knowledge, is considered. In $\mathsf{MSReduce}^{\bowtie}$, annotated sends and receives communicate with the term set that is determined by their annotation.

After *flatten*ing, the procedure *constraint_sequence* generates a constraint sequence for the interleaving (as it is done in [15]). The resulting constraint sequence is fed to the procedure $\mathsf{MSReduce}^{\bowtie}$.

EXAMPLE 3. Consider a file server service *fs*, with intensional knowledge $P = \{TA, TD\} \cup P^1$, where P^1 is the type-1 theory of example 1. A typical event of the main process of the *fs* service is $\{can\_read(A,F)\} \blacktriangleright \mathsf{s}(\{F\}_{pk(A)})$, where A and F denote, respectively, a client of the service and a file stored on *fs*. For this event the *expand* procedure returns $\{user(A), public(F)\} \vee \{admin(A), public(F)\} \vee \{admin(A), classified(F)\} \vee \{can\_read(A,F)\} \blacktriangleright \mathsf{s}(\{F\}_{pk(A)})$; see example 2.

The *interleavings* procedure then creates a branch for each $g \in_{\vee}$ the resulting guard, while interleaving this event with other events of the architecture. For each of the branches the *flatten* procedure maps the events into communication actions. For example, the guarded send $\{user(A), public(F)\} \blacktriangleright \mathsf{s}(\{F\}_{pk(A)})$ maps to $\mathsf{r}^{fs}(user(A), public(F)) \cdot \mathsf{s}^{\mathbb{A}}(\{F\}_{pk(A)})$. •

The following theorem states that algorithm 1 is terminating on decision problems for $\mathbf{A}_1$ architectures; moreover the algorithm is correct (i.e. sound and complete) w.r.t. the semantics of $\mathbf{A}_1$ architectures.

THEOREM 2. *Given decision problem* $\text{REACH}\langle \mathsf{arch}, a, f\rangle$, *with* arch *being an* $\mathbf{A}_1$ *architecture, algorithm 1 terminates, and returns a trace iff* $\text{REACH}\langle \mathsf{arch}, a, f\rangle = \mathsf{T}$.

The proof of theorem 2 is relegated to appendix B. Below, we sketch the main idea of the proof. Let $\mathbf{KS} = (S, S^0, T)$ be the Kripke structure induced by arch. A *trace* in $\mathbf{KS}$ is a sequence $z_0 e_1 z_1 \cdots e_n z_n$, where $z_0 = S^0$, $(z_{j-1}, z_j) \in T$, for $1 \le j \le n$, and the system evolves from z_{j-1} into z_j when event e_j occurs, as defined in section 3.2. A *symbolic trace* $\mathfrak{st}$ is a trace which may contain non-ground terms. We say $\mathfrak{st}$ is *realizable* in $\mathbf{KS}$ iff there exists a grounding substitution σ, such that $\mathfrak{st}\sigma$ is a trace in $\mathbf{KS}$. Given a symbolic trace $\mathfrak{st}$ and a sequence of constraints C we define their *correspondence* inductively: if $\mathfrak{st} = z_0$, then the empty sequence, i.e. $C = \mathsf{nil}$, corresponds to $\mathfrak{st}$. Now, let $\mathfrak{st} = z_0 e_1 \cdots z_n e z$, with $n \ge 0$, and C' be a sequence of constraints that corresponds to $z_0 \cdots z_n$. If $e = g \blacktriangleright \mathsf{s}(m)$, then any $C = C' \cdot \langle \zeta(g_i) : \zeta(\Omega_\pi^{@z_n})\rangle$ corresponds to $\mathfrak{st}$, where π is the service that performs e, $g_i \in_{\vee} \mathcal{E}(g)$ is calculated w.r.t. the intensional knowledge of π, and $\Omega_\pi^{@z_n}$ refers to the knowledge of π at symbolic configuration z_n. The correspondence between event e and constraint $\langle \zeta(g_i) : \zeta(\Omega_\pi^{@z_n})\rangle$ hinges upon theorem 1, which tells us that the encoding function ζ is such that $\zeta(G) \vdash \zeta(g_i)$ for at least one $g_i \in_{\vee} \mathcal{E}(g)$ if $\mathcal{K}(g) \in \lceil \mathcal{K}(G) \rceil^P$, given any $G \subseteq \mathcal{A}_{\Sigma(\emptyset)}$. If $e = \mathsf{r}(m) \rhd u$, then $C = C' \cdot \langle m : \Omega_{\mathbb{A}}^{@z_n}\rangle$ corresponds to $\mathfrak{st}$, where $\Omega_{\mathbb{A}}^{@z_n}$ refers to the attacker knowledge at symbolic configuration z_n. We remark that for the attacker knowledge the (suppressed) predicate $\mathcal{K}$ ranges directly over messages.

From the definition above, clearly there are finitely many constraint sequences (created due to the $\vee$ operator in guards) corresponding to any symbolic trace $\mathfrak{st}$. The proof of theorem 2 intuitively goes by showing that for each symbolic trace $\mathfrak{st}$, created by an interleaving of events of services in arch, $\mathfrak{st}$ is realizable in $\mathbf{KS}$ iff a constraint sequence corresponding to $\mathfrak{st}$ is satisfiable in algorithm 1.

We proceed with the CRP case study.

5.2.1 Verification results for CRP

We have developed a tool which implements algorithm 1. The tool is written in Prolog, and extends the constraint solver developed by Millen and Shmatikov [15]. We have used the tool to verify a number of properties on the CRP case study (specified in section 4.1), namely: (1) *Executability.* We replaced the last event in the specification of the citizen with

$$\mathsf{r}([h(mike, doc), \mathsf{success\_token}]_{piet}) \rhd \{stored(doc)\}$$

and checked if $\text{REACH}\langle \mathsf{crp}, mike, stored(doc)\rangle = \mathsf{T}$; that is, whether *mike* knows that his document has been successfully stored in the *cr*. The tool returns a trace showing that this property indeed holds in the CRP architecture. (2) *Secrecy.* We have checked if $\text{REACH}\langle \mathsf{crp}, eve, doc\rangle = \mathsf{T}$, i.e. whether the attacker *eve* can discover *doc*. No (attack) trace is found. (3) *Safety.* We have checked whether the attacker *eve* can obtain the right to store in the *cr*. That is, if $\text{REACH}\langle \mathsf{crp}, cr, can\_store(eve)\rangle = \mathsf{T}$. No (attack) trace is found.

It is worth mentioning that in $\mathbf{A}_1$ architectures, e.g. the formalization of CRP in section 4.1, only a finite number of services are allowed. Our decidability result would in fact fall apart if an unbounded number of services were considered in $\mathbf{A}_1$ architectures. This immediately follows from the undecidability of reachability in security protocols with an unbounded number of sessions.

6. CONCLUSION

We have presented a language for formal specification of service-oriented architectures. The language allows us to specify communication level events, policy level decisions, and the interface between the two. We have shown that the reachability problem is decidable for a fragment of architectures specified in the language. The decidable fragment is of immediate practical relevance.

In this decidable fragment, the policies of the (honest) services are limited to trust application and trust delegation rules, besides a finite set of Horn clauses of *type-1*. Type-1 Horn theories are characterized here by placing certain syntactic conditions on the form of the clauses. As we show in the paper, these conditions are indeed sufficient; they are however in general not necessary for deciding reachability. We intend to investigate how, without undermining our decidability result, the policies of honest services can be extended to theories beyond type-1 which are interesting in practice. This is left for future work.

Revocation of rights is not expressible in a natural way in authorization logics which are based on Horn theories, cf. [14]

for an overview. We are currently working towards accommodating rights revocation in our language. Investigating decidability issues in the presence of revocation is left for future work.

We (similar to, e.g., DKAL) see "know" as a predicate, as opposed to a modality as it is common in epistemic logics. This causes a discrepancy between the syntactic form of "knowledge", represented by predicates, and the semantic notion of knowledge. It must therefore be interesting to investigate how our proposed formalism (or, DKAL) can be connected to epistemic models, such as knowledge-based programs [10] and dynamic epistemic logic [8].

Acknowledgements

We are grateful to David Basin, Samuel Burri, Silvio Ranise and Eugen Zalinescu for their comments on this paper. The work has been supported by AVANTSSAR, FP7-ICT-2007-1 Project no. 216471.

7. REFERENCES

[1] A. Armando and S. Ponta. Model checking of security-sensitive business processes. In *FAST '09*, volume 5983 of *LNCS*, pages 66–80. Springer, 2010.

[2] AVANTSSAR. Deliverable D5.1: Problem cases and their trust and security requirements. Available at `http://www.avantssar.eu`, 2008.

[3] M. Barletta, S. Ranise, and L. Viganò. Verifying the interplay of authorization policies and workflow in service-oriented architectures. In *CSE (3)*, pages 289–296. IEEE Computer Society, 2009.

[4] M. Becker, C. Fournet, and A. Gordon. Design and semantics of a decentralized authorization language. In *CSF '07*, pages 3–15. IEEE Computer Society, 2007.

[5] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *CSFW '01*, pages 82–96. IEEE Computer Society, 2001.

[6] R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In *SAS '02*, volume 2477 of *LNCS*, pages 326–341. Springer, 2002.

[7] J. DeTreville. Binder, a logic-based security language. In *IEEE Symposium on Security and Privacy '02*, page 105, 2002.

[8] H. van Ditmarsch, W. van der Hoek, and B. Kooi. *Dynamic epistemic logic*. Springer, 1st edition, 2007.

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

[10] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning About Knowledge*. MIT, 2003.

[11] Y. Gurevich and I. Neeman. DKAL: Distributed-knowledge authorization language. In *CSF '08*, pages 149–162. IEEE Computer Society, 2008.

[12] Y. Gurevich and I. Neeman. The logic of infons. *Bulletin of the EATCS*, (98):150–178, 2009.

[13] J. Guttman, F. Thayer, J. Carlson, J. Herzog, J. Ramsdell, and B. Sniffen. Trust management in strand spaces. In *ESOP '04*, volume 2986 of *LNCS*, pages 325–339, 2004.

[14] J. Halpern and V. Weissman. Using first-order logic to reason about policies. *ACM Trans. Inf. Syst. Secur.*, 11(4), 2008.

[15] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *CCS '01*, pages 166–175. ACM Press, 2001.

[16] R. Parikh and R. Ramanujam. A knowledge based semantics of messages. *Journal of Logic, Language and Information*, 12(4), 2003.

[17] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *CSFW '01*, page 174. IEEE CS, 2001.

[18] A. Schaad, V. Lotz, and K. Sohr. A model-checking approach to analysing organisational controls in a loan origination process. In *SACMAT '06*, pages 139–149, 2006.

APPENDIX

A. ATTACKER MODEL

Below, we list the capabilities of the Dolev-Yao attacker model. The following rules describe the intensional knowledge (i.e. the deduction capabilities) of the attacker, cf. [5]. The attacker is called *eve* here.

$$\begin{array}{rll}
 & \textit{Analysis} & \\
x & \leftarrow (x, y) & \phi_{proj1} \\
y & \leftarrow (x, y) & \phi_{proj2} \\
x & \leftarrow \{x\}_{pk(eve)} & \phi_{pdec} \\
x & \leftarrow \{\!|x|\!\}_y, y & \phi_{sdec} \\
 & \textit{Synthesis} & \\
(x, y) & \leftarrow x, y & \phi_{pair} \\
\{x\}_y & \leftarrow x, y & \phi_{penc} \\
\{\!|x|\!\}_y & \leftarrow x, y & \phi_{senc} \\
h(x) & \leftarrow x & \phi_{hash} \\
sig(pk(eve), x) & \leftarrow x & \phi_{sig}
\end{array}$$

B. PROOF OF THEOREM 2

Below, we present the proof of theorem 2. The proof relies upon a lemma on correctness and termination of $\mathsf{MSReduce}^{\bowtie}$.

PROOF OF THEOREM 2. Termination of algorithm 1 is immediate, as the procedure *interleavings* produces a finite number of (symbolic) interleavings, and all the functions applied to interleavings, in particular $\mathsf{MSReduce}^{\bowtie}$ (see lemma 2, below) and *expand* (due to lemma 1), terminate in finitely many steps.

Notice that algorithm 1 exhaustively considers all possible symbolic traces of arch, and for each symbolic trace $\mathfrak{st}$, the algorithm considers all constraint sequences corresponding to $\mathfrak{st}$. We write $\widehat{\mathfrak{st}}$ for the set of all constraint sequences which corresponds to $\mathfrak{st}$.

The proof is split into two directions.

$\Rightarrow$ Assume $\textsc{Reach}\langle \mathsf{arch}, a, f\rangle = \mathsf{T}$. Then, there exists a symbolic trace $\mathfrak{st} = z_0 e_1 \cdots e_n z_n$, and a nonempty set of substitutions $\mathfrak{S} = \{\sigma_1, \sigma_2, \ldots\}$ where $\mathfrak{st}\sigma_j$ is a trace in **KS**, for any $\sigma_j \in \mathfrak{S}$. Then, there exists at least one constraint sequence $C \in \widehat{\mathfrak{st}}$, which is satisfiable by any substitution $\sigma_j \in \mathfrak{S}$, and that is considered by algorithm 1. It now remains to show that $\mathsf{MSReduce}^{\bowtie}$ returns a witness trace for C. This holds due to lemma 2, below.

$\Leftarrow$ Let C be a constraint sequence generated in algorithm 1. Then, $C \in \widehat{\mathfrak{st}}$, for some symbolic trace $\mathfrak{st}$. Suppose $\mathsf{MSReduce}^{\bowtie}$ returns a witness trace, showing that C is satisfiable under (non-ground) substitution ρ (see lemma 2 on applicability of $\mathsf{MSReduce}^{\bowtie}$, below). By theorem 1 and the fact that solving constraint $\langle m : T\rangle$ implies $T \vdash m$, it follows that $\mathfrak{st}\sigma$ is a trace in **KS**, for any ground substitution σ that refines ρ. That is, $\textsc{Reach}\langle \mathsf{arch}, a, f\rangle = \mathsf{T}$.

This completes the proof. $\square$

Lemma 2, below, concerns the applicability of $\mathsf{MSReduce}^{\bowtie}$. Given a reduction system for constraint sequences, we write $C \to C'$ iff C is reduced, using a rule in the reduction system, to C'. A reduction system is

- *terminating* iff it admits no infinite reductions;
- *sound* iff $C \to C'$ implies that for any σ that is a solution of C', σ is also a solution of C;
- *complete* iff for any σ that is a solution of C, there exists a reduction step $C \to C'$ such that σ is a solution of C'.

Lemma 2. $\mathsf{MSReduce}^{\bowtie}$ *is terminating, sound and complete for the constraint sequences generated in algorithm 1.*

We prove lemma 2 by revising the proofs of termination and correctness (i.e. soundness and completeness) of $\mathsf{MSReduce}$, originally presented in [15]. We start by giving an overview of $\mathsf{MSReduce}$ in appendix B.1. There, we also single out the differences between $\mathsf{MSReduce}$ and $\mathsf{MSReduce}^{\bowtie}$. Then, the proofs of termination and correctness of $\mathsf{MSReduce}^{\bowtie}$ are given, respectively, in appendices B.2 and B.3.

B.1 Overview of MSReduce

Extended attacker model. The attacker model considered in [15] is based on the standard Dolev-Yao attacker model, as formalized in appendix A. In addition, Millen and Shmatikov consider an *encryption hiding* operator, here denoted $\|\cdot\|_\cdot$, that serves as a technical means to avoid non-termination of the constraint-solving algorithm. Consequently, the following attacker capabilities are also considered:

$$\begin{array}{lll} & \textit{Encryption hiding} & \\ \{\!|x|\!\}_y & \leftarrow \|x\|_y & \phi_{open} \\ \|x\|_y & \leftarrow \{\!|x|\!\}_y & \phi_{hide} \end{array}$$

Given a set of messages T we denote by $\mathcal{F}(T)$ the set of messages that the attacker can derive from T using her capabilities. In terms of the formalization introduced in the paper, $\mathcal{F}(T) = \lceil T \rceil^{\mathbf{I}_\mathbb{A}}$.

Reduction procedure. In the following we give a short description of the $\mathsf{MSReduce}$ algorithm (see algorithm 2). The algorithm takes an initial constraint sequence IC as input, and builds the tree of all possible reductions for IC. Each node is labelled by a pair (C, σ) whose first element is a constraint sequence, and the second element is a substitution for the variables in C. Starting from the root $(IC, \emptyset)$, the tree is explored with a depth first search. Notice that the algorithm 2 uses a stack structure, rather than a tree structure, as it is usual in the depth first search. The exploration terminates successfully (i.e. node (C, σ) corresponds to an attack) when C is *simple*; that is, every constraint in C is of the form $\langle V : T\rangle$ where V is a variable. Simple constraint sequences (under the monotonicity and origination assumptions, see below) are immediately solvable.

Algorithm 2 MSReduce

```
REQUIRES: initial constraint sequence $IC$
  $stack := \emptyset$
  $stack.push((IC, \emptyset))$
  repeat
    $(C, \sigma) :=$ stack.pop
    let $c = \langle m : t\rangle$ be the first constraint in $C$
                              s.t. $m$ is not a variable
    if $c$ not found then
      return (satisfiable : $(C, \sigma)$)
    apply rule ($elim$) to $c$ until no longer applicable
    for all $r \in R$ do
      if $r$ is applicable to C then
        stack.push($r(C, \sigma)$)
  until stack.empty
  return (unsatisfiable)
```

Initially, the root of the tree is set to the pair $(IC, \emptyset)$ (i.e. $(IC, \emptyset)$ is the only element of the stack). Then, the tree is explored as follows. A node (C, σ) is popped from the stack, and the *active* constraint (that is, the first constraint $c = \langle m : T\rangle \in C$ with a non-variable m) is picked. If no such constraint exists in C, then C is simple hence the node is returned as a solution for the constraint sequence. If c is found, then all occurrences of stand-alone variables are removed from the term set T. Subsequently, for any applicable reduction rule r (see paragraph Reduction rules, below), the new node resulting from the application of r to the current node is pushed on top of the stack. When no nodes are left in the stack, all possible reductions have been considered, so the algorithm returns "unsatisfiable" and terminates.

Reduction rules. In the following we present the reduction rules used by $\mathsf{MSReduce}$. The rules read from top to bottom, that is, a rule r

$$\frac{(C_< \cdot c \cdot C_>, \sigma)}{(C'_< \cdot c' \cdot C'_>, \sigma')}\, r$$

says that constraint sequence $C_< \cdot c \cdot C_>$, where c is the active constraint and $C_<$ and $C_>$ are respectively the constraints preceding and following c, is reduced to $C'_< \cdot c' \cdot C'_>$, and substitution σ is refined by σ'.

$$\frac{(C_< \cdot \langle m : T \cup V\rangle \cdot C_>, \sigma)}{(C'_< \cdot \langle m : T\rangle \cdot C'_>, \sigma')}\, elim$$
where v is a variable

$$\frac{(C_< \cdot \langle m\!:\!T\rangle \cdot C_>, \sigma)}{(C_<\tau \cdot C_>\tau, \sigma \cup \tau)}\ \mathit{un}$$
$$\text{where } \tau = \mathit{mgu}(m,t), t \in T$$

$$\frac{(C_< \cdot \langle (m_1,m_2)\!:\!T\rangle \cdot C_>, \sigma)}{(C_< \cdot \langle m_1\!:\!T\rangle \cdot \langle m_2\!:\!T\rangle \cdot C_>, \sigma)}\ \mathit{pair}$$

$$\frac{(C_< \cdot \langle h(m)\!:\!T\rangle \cdot C_>, \sigma)}{(C_< \cdot \langle m\!:\!T\rangle \cdot C_>, \sigma)}\ \mathit{hash}$$

$$\frac{(C_< \cdot \langle \{m\}_k\!:\!T\rangle \cdot C_>, \sigma)}{(C_< \cdot \langle k\!:\!T\rangle \cdot \langle m\!:\!T\rangle \cdot C_>, \sigma)}\ \mathit{penc}$$

$$\frac{(C_< \cdot \langle \{\!|m|\!\}_k\!:\!T\rangle \cdot C_>, \sigma)}{(C_< \cdot \langle k\!:\!T\rangle \cdot \langle m\!:\!T\rangle \cdot C_>, \sigma)}\ \mathit{senc}$$

$$\frac{(C_< \cdot \langle \mathit{sig}(pk(\mathit{eve}))\!:\!T\rangle \cdot C_>, \sigma)}{(C_< \cdot \langle m\!:\!T\rangle \cdot C_>, \sigma)}\ \mathit{sig}$$

$$\frac{(C_< \cdot \langle m\!:\!T \cup \{(t_1,t_2)\}\rangle \cdot C_>, \sigma)}{(C_< \cdot \langle m\!:\!T \cup \{t_1,t_2\}\rangle \cdot C_>, \sigma)}\ \mathit{split}$$

$$\frac{(C_< \cdot \langle m\!:\!T \cup \{\{t\}_{pk(\mathit{eve})}\}\rangle \cdot C_>, \sigma)}{(C_< \cdot \langle m\!:\!T \cup \{t\}\rangle \cdot C_>, \sigma)}\ \mathit{pdec}$$

$$\frac{(C_< \cdot \langle m\!:\!T \cup \{\{t\}_k\}\rangle \cdot C_>, \sigma)}{(C_<\tau \cdot \langle m\tau\!:\!T\tau \cup \{\{t\tau\}_{k\tau}\}\rangle \cdot C_>\tau, \sigma\tau)}\ \mathit{ksub}$$
$$\text{where } \tau = \mathit{mgu}(k, pk(\mathit{eve})), k \neq pk(\mathit{eve})$$

$$\frac{(C_< \cdot \langle m\!:\!T \cup \{\{\!|t|\!\}_k\}\rangle \cdot C_>, \sigma)}{(C_< \cdot \langle k\!:\!T\rangle \cdot \langle m\!:\!T \cup \{k,t\}\rangle \cdot C_>, \sigma)}\ \mathit{pdec}$$

Properties of MSReduce. For applying MSReduce to a sequence of constraints C, it is required that [15]:

- (Monotonicity) If constraint $\langle m\!:\!T\rangle$ precedes $\langle m'\!:\!T'\rangle$ in C, then $\mathcal{F}(T) \subseteq \mathcal{F}(T')$.
- (Origination) For any constraint of the form $\langle m : T\rangle$ in C, with V being a variable appearing in T, there exists a preceding constraint $\langle m'\!:\!T'\rangle$ in C, where V is a subterm of m' and $\mathcal{F}(T') \subset \mathcal{F}(T)$.

MSReduce preserves the aforementioned properties at each reduction step. Notice that the monotonicity and the origination property are necessary to grant solvability of simple constraint sequences.

Overview of MSReduce$^{\bowtie}$. MSReduce and MSReduce$^{\bowtie}$ differ only in the properties of the constraint sequences that are passed to them as input.

Let C be a constraint sequence generated by algorithm 1. C can always be partitioned into two disjoint sequences of constraints C_A and C_P, that we call respectively "attacker constraints" and "policy constraints".

Formally, an attacker constraint $\langle m : T\rangle$ models a receive event at the communication level, where m is a term that an attacker should construct from a finite set of terms T. Here, both m and the terms in T are message terms, i.e. $m \in \mathcal{T}_{\Sigma(\mathcal{V})}$ and $T \subseteq \mathcal{T}_{\Sigma(\mathcal{V})}$.

A policy constraint models the evaluation of a query at the policy level. Differently from an attacker constraint, for a policy constraint $\langle q\!:\!K\rangle$ the term q and the terms in K are the result of the application of function ζ to *Infons*. In particular, all terms in policy constraints contain constructors belonging to Σ_{Infons} (cf. sections 4 and 5.1), while terms in attacker constraints never contain infon constructors. It is hence possible to distinguish between them with a simple syntactical check.

We note that C_A satisfies the monotonicity and origination properties. This is immediate from the syntax and semantics of our language; see section 3. The origination and monotonicity properties do not hold for C_P, in general. However, for any constraint of the form $\langle q : K\rangle$ in C_P, with V being a variable appearing in K, there exists a preceding constraint $\langle m' : T'\rangle$ in C_A, where V is a subterm of m'. This is because all variables in specifications of service-oriented architectures are originally instantiated at a receive event in an honest process.

We remark that the monotonicity and the origination properties hold for simple constraint sequences obtained by applying MSReduce$^{\bowtie}$ to constraint sequences generated by algorithm 1. The resulting simple constraint sequences are thus immediately solvable. This is because:

- policy constraints, for which the monotonicity property does not hold, are eliminated by the reduction procedure. This is due to the fact that the variables in policy constraints are always subterms of applications of infon constructors. Infon constructors are uninterpreted functions for the attacker's inference rules; hence they are never constructed or deconstructed by the reduction system. If such constraints are satisfiable, then they will be eliminated using the unification rule *un*.
- the reduction procedure preserves the monotonicity of the attacker constraints, as shown in [15].

B.2 Proof of termination

The proof of termination of MSReduce$^{\bowtie}$ follows closely the termination proof of MSReduce. The proof is based on a termination measure (N_v, N_s) of a constraint sequence C. Here, N_v and N_s are naturals. In particular, N_v is the number of distinct variables occurring in C and N_s is a special *expansion* measure. Tuples are ordered lexicographically.

The expansion measure hinges upon another measure, the size $|m|$ of a term m, that is the number of operator applications plus the number of constants and variable in m. Then the expansion measure N_s of constraint sequence C is the sum of the expansion measures of its constraints; in turn, the expansion measure of a constraint $\langle m : T\rangle$ is $|m| \cdot \chi(T)$, where χ is defined as follows:

$$\begin{array}{l}
\chi(t) = 2 \quad \text{if } t \text{ is a variable or constant} \\
\chi(\{t_1, \ldots, t_n\}) = \chi(t_1) \cdots \chi(t_n) \\
\chi((t_1, t_2)) = \chi(t_1)\chi(t_2) + 1 \\
\chi(\{t\}_k) = \chi(t) \\
\chi(\|t\|_k) = 1 \\
\chi(sig(k, t)) = \chi(t) + 1 \\
\chi(h(t)) = \chi(t) + 1 \\
\chi(\{\!|t|\!\}_k) = \chi(t)\chi(k) + |k| + 1
\end{array}$$

We show now that the termination measure decreases strictly at each application of a reduction rule. Rule *elim* removes a stand-alone variable, hence reduces N_s; rule *un* either substitutes a variable, hence decreasing N_v, or decreases N_s by removing the constraint; rules *sig*, *pair*, *hash*, *penc* and *senc* reduce N_s by splitting the constraint in constraints whose sum of expansion measures is smaller; rules *split* and *pdec* decrease N_s by replacing a term with terms whose product of expansion measures is smaller; rule *ksub* substitutes a variable, hence decreases N_v. Finally rule *sdec* replaces a constraint $c = \langle m : T \cup \{\!|t|\!\}_k\rangle$ with constraints $c' = \langle k : T \cup \|t\|_k\rangle$ and $c'' = \langle m : T \cup \{t, k\}\rangle$. The expansion measure of c is $|m|\chi(T)(\chi(t)\chi(k) + |k| + 1)$, while the product of the expansion measures of c' and c'' is $|k|\chi(T) + |m|\chi(T)\chi(t)\chi(k)$. Since $|k| < |m|(|k| + 1)$, the expansion is strictly decreased in each *sdec* reduction step. $\mathsf{MSReduce}^{\bowtie}$ thus terminates.

B.3 Proof of correctness

In the following, for ease of presentation, we ignore the encryption hiding operator and related attacker capabilities. The encryption hiding operator is used in [15] merely to avoid non-termination, as mentioned above.

B.3.1 Proof of soundness

We show here that if a rule of $\mathsf{MSReduce}^{\bowtie}$ reduces constraint sequence C to constraint sequence C', then any solution σ of C' is also a solution of C. In other words, $\mathsf{MSReduce}^{\bowtie}$ does not introduce new solutions.

We condition on the rule applied:

- Rule *elim* removes a stand-alone variable V from $T \cup \{V\}$, for the active constraint being $c = \langle m : T \cup \{V\}\rangle$. We need to distinguish two cases:
  - c is an attacker constraint. We show that $\mathcal{F}(T \cup V) = \mathcal{F}(T)$. It is obvious that $\mathcal{F}(T) \subseteq \mathcal{F}(T \cup \{V\})$, we show then that $\mathcal{F}(T \cup \{V\}) \subseteq \mathcal{F}(T)$. By origination property, there exists an earlier constraint $c' = \langle m' : T'\rangle$, such that V does not appear in T' and V appears in m'. In particular, $m' = V$ (all constraints earlier than c are simple). Due to idempotency of closure, it suffices to show that $T \cup \{V\} \subseteq \mathcal{F}(T)$, and we only need to show that $V \subseteq \mathcal{F}(T)$, since $T \subseteq \mathcal{F}(T)$. By the monotonicity property $T' \subseteq T \cup \{V\}$, and since $V \notin T'$ then $T' \subseteq T$. But $V \in \mathcal{F}(T')$, therefore also $V \in \mathcal{F}(T)$.
  - c is a policy constraint. Then rule *elim* is never applied, because all variables in T appear under the application of an infon constructor. Furthermore, infon constructors are uninterpreted functions, hence can not be be deconstructed to yield the variables they contain.
- Rules *split* and *pdec* are sound since $\mathcal{F}$ is closed under ϕ_{pair} and ϕ_{penc}.
- Rule *un* removes $c = \langle m : T\rangle$ when m is unifiable with some term $t \in T$. Let $\tau = mgu(m, t)$. If σ is a solution for C', then $\sigma\tau$ is a solution for C provided that $T\sigma\tau \vdash m\sigma\tau$. This is obvious since $T\tau \vdash m\tau$.
- Rules *pair*, *hash*, *penc*, *senc* and *sig* are sound since $\mathcal{F}$ is closed under the corresponding ϕ rules of the attacker.
- Rule *sdec* replaces the active constraint $c = \langle m : T\rangle$, with $\{\!|t|\!\}_k \in T$, with the constraints $\langle k : T\rangle$ and $\langle m : T \cup \{k, t\}\rangle$. This rule is sound because if $k \in \mathcal{F}(T)$, then $\mathcal{F}(T) = \mathcal{F}(T \cup \{k\})$; moreover, since $\mathcal{F}$ is closed under ϕ_{sdec}, we have $\mathcal{F}(T) = \mathcal{F}(T \cup \{k, t\})$ given $k \in \mathcal{F}(T)$.

B.3.2 Proof of completeness

We show here that for every constraint sequence C and solution σ of C, there exists a rule r that reduces C to C', and σ is a solution of C'. In other words, all solutions for C are preserved in at least one reduction path.

Let $\langle m : T\rangle$ be the active constraint in C. The proof relies on the existence of a *normal* proof of $T\sigma \vdash m\sigma$, that is, a proof tree such that no label appears more than once in any path from the root to a leaf [15]:

Proposition 1. *Let t be a ground term and T a set of ground terms. If $t \notin T$ and $t \in \mathcal{F}(T)$ then there exists a normal sequence $\phi_1, \cdots, \phi_n$ such that $t \in \phi_n(\cdots \phi_1(T))$ and one of the following conditions holds:*

- *ϕ_n is a synthesis rule*
- *ϕ_1 is an analysis rule*
- *ϕ_i, for some $1 \leq i \leq n$, is ϕ_{sdec} and $\phi_1, \cdots, \phi_{i-1}$ are synthesis rules*

The proposition intuitively states that any normal sequence $\phi_1, \cdots, \phi_n$ such that $t \in \phi_n(\cdots, \phi_1(T))$ can be reordered so that analysis rules always appear earlier than synthesis rules, except in the case where ϕ_{sdec} is used (i.e. synthesis rules appear before ϕ_{sdec} to construct a non-atomic key).

Finding an applicable rule. Let $c = \langle m : T\rangle$ be the active constraint in C and σ a solution of C. Then $m\sigma \in \mathcal{F}(T\sigma)$. Intuitively, $\mathsf{MSReduce}^{\bowtie}$ tries to apply a sequence of reduction rules that reflects the order of a normal proof of $T\sigma \vdash m\sigma$ as indicated by proposition 1.

Recall that by definition c does not contain stand-alone variables neither on the left hand side (it is chosen as the first constraint whose left hand side is not a variable) nor on the right hand side (as all stand-alone variables are removed by application of the *elim* rule). It follows that every term in $\{m\} \cup T$ has a well-defined *top level structure*, i.e. outermost function application, against which applicability of a rule can be checked.

If $m\sigma \in T\sigma$, then rule *un* is applicable. If $m\sigma \notin T\sigma$ then we can assume a normal sequence of operators $\phi_1, \cdots, \phi_n$ as described in proposition 1. If ϕ_1 is an analysis rule then there is a term $t \in T$ with corresponding top level structure, hence the corresponding analysis rule can be applied. Similarly for ϕ_n being a synthesis rule, since m has a well-defined top structure the corresponding synthesis rule can be applied. Finally, if ϕ_i is ϕ_{sdec} then there is a term $\{\!|x|\!\}_y \in T$ that enables rule *sdec*.

Preserving the solution. Proving that each applicable rule preserves the solution proceeds by cases, on ϕ_1 if the rule is an analysis rule, or on ϕ_n if the rule is a synthesis rule. For brevity, we omit the details and explain only the proof for the case of *sdec*, i.e. in which ϕ_i is ϕ_{sdec} and $\phi_1 \cdots \phi_{i-1}$ are all synthesis rules. Rule *sdec* replaces the active constraint $c = \langle m\!:\!T\rangle$ with constraints $c' = \langle k\!:\!T\rangle$ and $c'' = \langle m\!:\!T \cup \{k, t\}\rangle$, for term $\{\!|t|\!\}_k \in T$. Observe that $k\sigma \in \phi_{i-1}(\cdots \phi_1(T\sigma))$, otherwise rule *sdec* would not be applicable, and consequently σ is also a solution for c'. Also, since $k\sigma \in \mathcal{F}(T\sigma)$ and $\mathcal{F}$ is idempotent, then $\mathcal{F}(T\sigma) = \mathcal{F}(T\sigma \cup \{k\sigma, t\sigma\})$, which shows that σ is a solution for c''.