Understanding SPKI/SDSI Using First-Order Logic

Ninghui Li John C. Mitchell
Department of Computer Science, Stanford University
Gates 4B, Stanford, CA 94305-9045
{ninghui.li, jcm}@cs.stanford.edu

Abstract

SPKI/SDSI is a language for expressing distributed access control policy, derived from SPKI and SDSI. We provide a first-order logic (FOL) semantics for SDSI, and show that it has several advantages over previous semantics. For example, the FOL semantics is easily extended to additional policy concepts and gives meaning to a larger class of access control and other policy analysis queries. We prove that the FOL semantics is equivalent to the string rewriting semantics used by SDSI designers, for all queries associated with the rewriting semantics. We also provide a FOL semantics for SPKI/SDSI and use it to analyze the design of SPKI/SDSI. This reveals some problems. For example, the standard proof procedure in RFC 2693 is semantically incomplete. In addition, as noted before by other authors, authorization tags in SPKI/SDSI are algorithmically problematic, making a complete proof procedure unlikely. We compare SPKI/SDSI with RT_1^C, which is a language in the RT Role-based Trust-management framework that can be viewed as an extension of SDSI. The constraint feature of RT_1^C, based on Constraint Datalog, provides an alternative mechanism that is expressively similar to SPKI/SDSI tags, semantically natural, and algorithmically tractable.

1 Introduction

In 1996, Rivest and Lampson [26] proposed a new public-key infrastructure, called the Simple Distributed Security Infrastructure (SDSI), featuring the use of linked local names. Concurrently, Ellison et al. developed the Simple Public Key Infrastructure (SPKI), which emphasizes delegation of authorization. In 1997, the two efforts were merged, leading to a system called SPKI/SDSI. The standard reference on SPKI/SDSI is RFC 2693 [6], with a later paper [4], whose authors include several designers of SPKI/SDSI, providing certificate chain reduction algorithms for SPKI/SDSI, clearer descriptions of many features, and certain minor design changes. SPKI/SDSI can be viewed as a trust-management (TM) language. Trust management is an approach to access control in decentralized distributed systems with access control decisions based on policy statements made by multiple principals. In general, a TM language has a syntax for specifying *policy statements* and *queries*, together with a semantic relation $\vdash$. Given a set $\mathcal{P}$ of policy statements and a query $\mathcal{Q}$, the relation $\mathcal{P} \vdash \mathcal{Q}$ means that $\mathcal{Q}$ follows from $\mathcal{P}$. When $\mathcal{Q}$ arises from an access request, $\mathcal{P} \vdash \mathcal{Q}$ means that access $\mathcal{Q}$ is allowed in $\mathcal{P}$.

In SPKI/SDSI, policy statements take the form of name-definition certificates (name certs), authorization certificates (auth certs), and Access Control List (ACL) entries. *Principals* are identified with public keys, and each principal has its own name spaces for names. A local name, which is identified by a principal and an identifier, is bound to a set of principals that we call the members of the local name. Only the principal K can issue name certs that determine members of the local name K A. Principal K can define K A to include a principal, a local name, or a linked local name (also called an extended name). We use SDSI to refer to the sub-language of SPKI/SDSI that just has name certs, also called 4-tuples. Auth certs and ACL entries, also called 5-tuples, originally came from SPKI. In a 5-tuple, the issuing principal grants certain authorization to a subject, which can be a principal, a local name, an extended name, or a threshold subject. The issuing principal also specifies whether the subject can further delegate the authorization it receives in a 5-tuple.

A set of SPKI/SDSI statements defines a policy, and many properties of a policy are of interest to its authors and users. The most basic query is whether a policy allows a principal to access a resource. However, it may also be important to determine safety and availability properties of a policy [21], such as whether a resource owner still has some guarantees about who can access their resources after delegating limited authority to other principals. RFC 2693 does not explicitly specify a class of queries that can be made against a policy, but provides operational rules for producing new 5-tuples from existing 5-tuples. The 5-tuple reductions implicitly define a class of access queries for

SPKI/SDSI as well as a proof procedure for answering these queries.

The semantics of SPKI/SDSI has attracted a lot of interest in the security research community. Beginning with Abadi [1], significant effort has gone into finding a logic-based semantics for both SDSI naming alone and SPKI/SDSI. A logic developed by Howell and Kotz [11] extends the ABLP logic [2, 15] with a restricted form of delegation and provides a semantics for SPKI/SDSI authorization. Halpern and van der Meyden subsequently developed the Logic of Local Name Containment (LLNC) [8] and then extend the logic to deal with 5-tuples in SPKI [9]. These studies all use specialized propositional modal logics. Li [16] provides a logic programming based reading for SPKI/SDSI, which has several drawbacks. In the present paper, we use standard first-order logic (FOL) and explicitly consider the class of queries supported by the semantics.

In our study of the SDSI naming portion of SPKI/SDSI (Section 2 of this paper), we present a FOL semantics based on translating each name cert into a Datalog clause. Datalog is a limited form of logic programming that does not have function symbols (except for zero-ary functions). Datalog is also a query language for relational databases. Since Datalog is a subset of first-order logic, this leads to a FOL semantics; since Datalog has a computational interpretation, a set of name certs induces a Datalog program. Since the semantics of a policy is defined by classical logical implication, this approach allows any FOL formula to be understood as a meaningful query against a policy. We prove that the FOL semantics is equivalent to the string rewriting semantics used by SDSI designers, for all queries associated with the rewriting semantics. The advantages of our approach over previous logics are the following. It captures the set-based semantic intuition of SDSI, which is used in Clarke et al. [4] and Halpern and van der Meyden [8]. It uses classical first-order logic rather than more complex modal logics. The FOL semantics contains more information in the sense that a larger class of queries can be formulated and understood in the semantics. The semantics is easily extended to support useful extensions to SDSI; and, finally, the relationship between the FOL semantics and logic programming provides an efficient method to answer a large class of queries.

In our study of full SPKI/SDSI, we first describe a trust management language RT_1^C [18], which uses Constraint Datalog as its semantic foundation. We then give a FOL semantics for SPKI/SDSI by treating authorization tags and validity specification as constraints, and examine several design issues of SPKI/SDSI, using RT_1^C for comparison. The language RT_1^C [18], which may be viewed as an alternate extension of SDSI, is a language from the RT family of Role-based Trust-management languages [20, 22]. One characteristic of RT_1^C is the use of various constraint domains to express policies about structured resources such as file hierarchies and standard concepts such as time of day or days of the week. After developing a semantics for SPKI/SDSI that treats authorization tags and validity specification as constraints, we observe that SPKI's 5-tuple reduction procedure is semantically incomplete. One reason is that reduction does not handle union of tags. For example, if K_1 grants to K_2 in two certificates with two authorization tags `(* range numeric ge 1 le 5)` and `(* numeric range ge 4 le 10)`, it is not possible to derive that K_1 grants to K_2 `(* range numeric ge 2 le 7)`, which is a logical implication of the two 5-tuples. Another problem, pointed out by Howell [10], is that the intersection of two authorization tags may be a set that is not representable by any finite set of authorization tags. This suggests that no rewriting algorithm dealing with SPKI/SDSI tags can be semantically complete. We then discuss several design issues of SPKI/SDSI, referring to RT_1^C for comparison. Among other benefits of a logical point of view, the constraint feature of RT_1^C is expressively similar to SPKI/SDSI tags, but semantically natural and algorithmically tractable.

We give FOL semantics and equivalence results for the SDSI naming portion of SPKI/SDSI in Section 2, with trust management language RT_1^C [18] and Constraint Datalog in Section 3, FOL semantics and analysis of SPKI/SDSI in Section 4. We conclude in Section 5.

2 Understanding SDSI Using First-Order Logic (FOL)

In SDSI, principals are identified with public keys. We use $\mathcal{K}$ to denote the set of all principals and use K, often with subscripts or superscripts, to denote a principal. An *identifier* is a word over some given standard alphabet. The set of all identifiers is denoted by $\mathcal{A}$, and an identifier is denoted by A or B (often with subscripts). We assume that both $\mathcal{K}$ and $\mathcal{A}$ are countable. We do not consider SDSI 1.1 [26] *special roots*, which are identifiers that are bound to the same principal in every name space. Although it would be straightforward to treat them, they do not seem to add any special interest and do not appear in SPKI/SDSI.

A *name string*, called a term in [4, 12], is a principal followed by zero or more identifiers. In other words, the set $\mathcal{S} \subseteq (\mathcal{K} \cup \mathcal{A})^*$ of name strings contains strings over $\mathcal{K} \cup \mathcal{A}$. A *local name* has the form $K\ A$, where $K \in \mathcal{K}$ and $A \in \mathcal{A}$, and an *extended name* is a principal followed by more than one identifier. Using $\mathcal{N}_{\mathcal{L}}$ for the set of all local names and $\mathcal{N}_{\mathcal{E}}$ for extended names, we have $\mathcal{S} = \mathcal{K} \cup \mathcal{N}_{\mathcal{L}} \cup \mathcal{N}_{\mathcal{E}}$, *i.e.*, a name string is either a principal, a local name, or and extended names.

A *name-definition certificate* (*name cert*) C is a signed 4-tuple (K, A, S, V), where

- $K \in \mathcal{K}$ is a principal (public key) called the *issuer*; the certificate is signed by K.
- $A \in \mathcal{A}$ is an identifier.
- $S \in \mathcal{S}$ is a name string, called the subject.
- The validity specification V provides information regarding the validity of the certificate.

We say the 4-tuple (K, A, S, V) *defines* the local name $K\,A$ and S is the *definition*. In this section, we ignore the validity specification V. We assume that the validity of name certs are checked, and only valid certificates are considered. We will discuss how to handle the validity specification in the logical semantics in Section 4.2. When ignoring the validity specification, we write $(K, A, S, \cdot)$ for a 4-tuple.

In the rest of this section, we describe three semantics for SDSI that have appeared (sometimes implicitly) in the literature, then introduce the FOL semantics and prove its equivalence to other semantics; finally, we compare the FOL semantics with other semantics for SDSI.

2.1 A String Rewriting Semantics for SDSI

RFC 2693 defines a 4-tuple-reduction mechanism, which Clarke et al. [4] explicitly describes as string rewriting. The main idea is to replace a local name in a name string by its definition. Regarding a 4-tuple $(K, A, S, \cdot)$ a rewriting rule $K\,A \mapsto S$, the two 4-tuple-reductions in Section 6.4 of RFC 2693 operate on the left end of a name string as follows:.

Using $K_1\ A_1 \mapsto K_2$, rewrite
$K_1\ A_1\ A_2\ \cdots\ A_\ell \rightarrowtail K_2\ A_2\ \cdots\ A_\ell$

Using $K_1\ A_1 \mapsto K_2\ B_1\ \cdots\ B_k$, rewrite
$K_1\ A_1\ A_2\ \cdots\ A_\ell \rightarrowtail K_2\ B_1\ \cdots\ B_k\ A_2\ \cdots\ A_\ell$

In the rest of this paper, we often write $K\ A \mapsto S$ instead of $(K, A, S, \cdot)$ to represent a 4-tuple. The following definition shows how set of 4-tuples defines a binary relation $\overset{*}{\rightarrowtail}$ on name strings.

Definition 1 (Rewriting Semantics for SDSI: $\mathsf{RS}[\mathcal{P}]$**)** If $\mathcal{P}$ is a set of 4-tuples, let $\mathsf{RS}[\mathcal{P}]$ be the rewriting system that rewrites $S_1 \rightarrowtail S_2$ whenever there is a corresponding rule $K_1\ A_1 \mapsto K_2$ or $K_1\ A_1 \mapsto K_2\ B_1\ \cdots\ B_k$ given by a 4-tuple in $\mathcal{P}$. We write $\mathsf{RS}[\mathcal{P}] \rhd S_1 \overset{*}{\rightarrowtail} S_2$ if it is possible to rewrite S_1 to S_2 using zero or more steps from $\mathsf{RS}[\mathcal{P}]$.

The class of string rewriting systems generated by sets of 4-tuples have the following characteristics: when rewriting a name string, the rewriting replaces the first two symbols, which form a local name, with another name string, and the result is also a name string. Jha and Reps [12] pointed out that SDSI string rewriting systems correspond exactly to the class of string rewriting systems modelled using push-down systems. This connection reduces certain computation problems in SDSI to reachability analysis and model checking in push-down systems, for which efficient algorithms exist.

Given a set of 4-tuples, for any name string S, there exists a set of principals that S can rewrite into. This determines a valuation for every name string, and leads to the set-theoretic semantics for SDSI.

2.2 A Set-theoretic Semantics

In the set-theoretic semantics for SDSI, every name string has a valuation that is a set of principals. Clarke et al. [4] use an informal description of this semantics as the semantic intuition for SDSI. Halpern and van der Meyden [8] use the same spirit to provide a semantics for their Logic of Local Name Containment (LLNC). Li et al. [22] use the same idea to provide a semantics for RT_0, which can be viewed as SDSI enhanced with the intersection operator in the subject. The following presentation of the semantics follows [22].

Definition 2 (Set-theoretic Semantics for SDSI: $\mathcal{M}[\mathcal{P}]$**)** Given any function $f : \mathcal{N_L} \to \wp(\mathcal{K})$ mapping local names to sets of principals, where $\wp(\mathcal{K})$ is the power set of $\mathcal{K}$, we extend f to a valuation $\mathcal{V}_f : \mathcal{S} \to \wp(\mathcal{K})$ on all name strings as follows:

$$\begin{array}{rcl} \mathcal{V}_f(K) & = & \{K\} \\ \mathcal{V}_f(K\ A) & = & f(K\ A) \\ \mathcal{V}_f(K_1\ A_1\ \cdots\ A_\ell) & = & \bigcup_{K_2 \in f(K_1\ A_1)} \mathcal{V}_f(K_2\ A_2\ \cdots\ A_\ell) \end{array}$$

where $\ell > 1$.

The semantics $\mathcal{M}[\mathcal{P}]$ of a set $\mathcal{P}$ of 4-tuples is the least[1] function $f : \mathcal{N_L} \to \wp(\mathcal{K})$ that satisfies the system of set containments

$$\mathsf{SC}[\mathcal{P}] = \{\ f(K\ A) \supseteq \mathcal{V}_f(S)\ \mid\ K\ A \mapsto S \in \mathcal{P}\ \}$$

We use a least-solution definition for $\mathcal{M}[\mathcal{P}]$ because 4-tuples may define local names recursively. Note that $\mathcal{V}$ naturally extends $\mathcal{M}[\mathcal{P}]$ and gives a valuation for each name string. ∎

We now show that the function $\mathcal{M}[\mathcal{P}] : \mathcal{N_L} \to \wp(\mathcal{K})$ is well defined and present a straightforward way to construct it finitely. We define the following sequence of functions $f^i : \mathcal{N_L} \to \wp(\mathcal{K}), i \in \mathbb{N}$

$$\begin{array}{rcl} f^0(K\ A) & = & \emptyset \\ f^{i+1}(K\ A) & = & \displaystyle\bigcup_{K\ A \mapsto S \in \mathcal{P}} \mathcal{V}_{f^i}(S) \end{array}$$

[1]One function $f : \mathcal{N_L} \to \wp(\mathcal{K})$ is less than another $g : \mathcal{N_L} \to \wp(\mathcal{K})$ if $f(K\ A) \subseteq g(K\ A)$ for every local name $K\ A \in \mathcal{N_L}$.

It is easy to show by induction that, for any local name $K\ A$ not defined in $\mathcal{P}$, $f^i(K\ A) = \emptyset$ for any $i \in \mathbb{N}$, and that, for any local name $K\ A$ and any $i \in \mathbb{N}$, $f^i(K\ A)$ only contains principals occurring in $\mathcal{P}$. It follows that the set of all functions that could occur in the sequence f^i forms a finite lattice. In addition, the sequence f^i is nondecreasing. Therefore, the least fixed point f^ω of the sequence, given by $f^\omega = \bigcup_{i \geq 0} f^i(K\ A)$, exists; in addition, $f^\omega = \mathcal{M}[\mathcal{P}]$. The function f^ω clearly satisfies $\mathsf{SC}[\mathcal{P}]$, and one can show by induction that any function that satisfies $\mathsf{SC}[\mathcal{P}]$ is greater than or equal to f^ω.

This set-theoretic semantics contains less information than the rewriting semantics, in the following sense. We will show that, if $\mathsf{RS}[\mathcal{P}] \rhd S_1 \overset{*}{\rightarrowtail} S_2$, then $\mathcal{V}_{M[P]}(S_1) \supseteq \mathcal{V}_{M[P]}(S_2)$; this follows from Proposition 1 in Section 2.3 and Theorem 3 in Section 2.5. However, the converse is only guaranteed to be true when S_2 is a principal, *i.e.*, $\mathcal{V}_{M[P]}(S) \ni K$ if and only if $\mathsf{RS}[\mathcal{P}] \rhd S \overset{*}{\rightarrowtail} K$. This follows from Proposition 1 in Section 2.3 and Theorem 3 in Section 2.5; see also Proposition 2 in Section 2.5. It may well be the case that $\mathcal{V}_{M[P]}(S_1) \supseteq \mathcal{V}_{M[P]}(S_2)$, but there is no rewriting relationship between S_1 and S_2. For example, given $\mathcal{P}$ that contains the two four tuples $K\ A_1 \mapsto K_2$ and $K\ A_2 \mapsto K_2$, the valuation of $K\,A_1$ clearly contains the valuation of $K\ A_2$; however, using $\mathcal{P}$, one cannot rewrite $K\ A_1$ into $K\ A_2$. In other words, $\mathsf{RS}[\mathcal{P}] \rhd S_1 \overset{*}{\rightarrowtail} S_2$ is not equivalent to $\forall K((\mathsf{RS}[\mathcal{P}] \rhd S_1 \overset{*}{\rightarrowtail} K) \Leftarrow (\mathsf{RS}[\mathcal{P}] \rhd S_2 \overset{*}{\rightarrowtail} K))$; the former is stronger than the latter.

When making access control decisions, the valuations of name strings are necessary, but the rewriting relationship between two name strings are not directly useful. However, this relation is very helpful for understanding the effect of 4-tuples. If S_1 rewrites into S_2 from $\mathcal{P}$, then S_1 rewrites into S_2 from any $\mathcal{P}'$ such that $\mathcal{P}' \supseteq \mathcal{P}$; therefore, the valuation of S_1 is always a superset of the valuation of S_2 no matter what new policy statements are added. This is especially useful for studying the availability properties of policies when policies may change, *e.g.*, statements may be added and/or removed [21].

2.3 A Logic Programming (LP) Semantics for SDSI

The set-theoretic semantics can be captured naturally using logic programs. This observation was made by Halpern and van der Meyden [8]. It was also used implicitly in the semantics of the RT framework [22, 20]. The semantics of RT_0, the basic component of the RT framework, was first defined using sets [22]. When additional features, such as internal structures of role terms (role terms correspond to identifiers in SDSI) are added, a logic programming based semantics is used [20]. The advantage of the LP approach over the set approach is that it is easily extended to additional forms of policy statements.

Definition 3 (LP Semantics for SDSI) We use one ternary predicate m; intuitively, $m(K, A, K')$ means that K' is in the valuation of the local name $K\ A$. We define a macro $\mathsf{contains}$, which takes a name string and a logical variable as parameters, and defines a first-order logic formula.

$$\begin{array}{ll}
\mathsf{contains}[K][z] & \text{is} \quad (K = z) \\
\mathsf{contains}[K\ A][z] & \text{is} \quad m(K, A, z) \\
\mathsf{contains}[K\ A_1\ A_2\ \cdots\ A_\ell][z] & \text{is} \\
\quad \exists y_1\,(m(K, A_1, y_1) \land \mathsf{contains}[y_1\ A_2\ \cdots\ A_\ell][z]) & \\
\quad \text{where } \ell > 1 &
\end{array}$$

Given a set $\mathcal{P}$ of 4-tuples, we define $\mathsf{LP}[\mathcal{P}]$ to be the following set of LP clauses:

$$\{\ \forall z(\mathsf{contains}[K\ A][z] \Leftarrow \mathsf{contains}[S][z])\ \mid\ K\ A \mapsto S\ \}$$

To see that $\forall z(\mathsf{contains}[K\ A][z] \Leftarrow \mathsf{contains}[S][z])$ is a LP clause (*i.e.*, Horn clause), observe that $\mathsf{contains}[K\ A_1\ A_2\ \cdots\ A_\ell][z]$ is logically equivalent to $\exists y_1 \exists y_2 \cdots \exists y_{\ell-1}\ m(K, A_1, y_1) \land m(y_1, A_2, y_2) \land \cdots \land m(y_{\ell-1}, A_\ell, z)$. Further observe that the above definition of $\mathsf{LP}[\mathcal{P}]$ is equivalent to defining $\mathsf{LP}[\mathcal{P}]$ to contain:

$$\begin{array}{ll}
m(K, A, K_1) & \text{for each } K\ A \mapsto K_1 \in \mathcal{P} \\
m(K, A, z) \mathrel{:-} m(K_1, A_1, z) & \\
\qquad \text{for each } K\ A \mapsto K_1\ A_1 \in \mathcal{P} & \\
m(K, A, z) \mathrel{:-} m(K_1, A_1, y_1),\ \cdots,\ m(y_{\ell-1}, A_\ell, z) & \\
\qquad \text{for each } K\ A \mapsto K_1\ A_1\ \cdots\ A_\ell \in \mathcal{P}, \ell > 1 &
\end{array}$$

The semantics of $\mathcal{P}$ is defined to be the minimal Herbrand model of $\mathsf{LP}[\mathcal{P}]$. If an atom $m(K_1, A, K_2)$ is in the minimal Herbrand model of $\mathsf{LP}[\mathcal{P}]$, we write $\mathsf{LP}[\mathcal{P}] \models m(K_1, A, K_2)$.

We now show that the LP semantics is equivalent to the set-theoretic semantics. Given a set $\mathcal{P}$ of 4-tuples, consider the set G of all functions $g : \mathcal{N}_\mathcal{L} \rightarrow \wp(\mathcal{K})$ that satisfies the following two conditions: (1) $g(K\ A) = \emptyset$ if either K or A does not appear in $\mathcal{P}$; (2) for any local name $K\ A$, $g(K\ A)$ contains only principals in $\mathcal{P}$. There exists a bijection between G and the set of all the Herbrand interpretations of $\mathsf{LP}[\mathcal{P}]$. Given a function $g \in G$, the corresponding interpretation is obtained by including $m(K, A, K')$ in the interpretation if and only if $g(K\ A) \ni K'$. Similarly, one obtains a function from each Herbrand interpretation, by assigning to $K\ A$ the smallest set that contains each K' such that $m(K, A, K')$ is in the interpretation. Furthermore, the definition of $\mathsf{contains}$ extends the predicate m to determine members of name strings, in exactly the same way in which $\mathcal{V}$ extends f in Definition 2. Therefore, a function g satisfies $\mathsf{SC}[\mathcal{P}]$ if and only if the interpretation corresponding to g is a model of $\mathsf{LP}[\mathcal{P}]$, and so the least solution $\mathcal{M}[\mathcal{P}]$ corresponds to the least Herbrand model of $\mathsf{LP}[\mathcal{P}]$. These establish the following equivalence.

Proposition 1 (Equivalence of LP semantics and set-theoretic semantics) *Given a set $\mathcal{P}$ of 4-tuples, a name string S and a principal K', $\mathsf{LP}[\mathcal{P}] \models \mathsf{contains}[S][K']$ if and only if $\mathcal{V}_{M[P]}(S) \ni K'$. In particular, for any local name K A, $\mathsf{LP}[\mathcal{P}] \models m(K, A, K')$ if and only if $\mathcal{M}[\mathcal{P}](K\ A) \ni K'$.*

The LP semantics is an attractive logical semantics. It is natural in the sense that it directly captures the set-based semantics intuition. It can also be used for computation purposes. Observe that $\mathsf{LP}[\mathcal{P}]$ is a Datalog program, that is, it does not have any function symbol other than constants. In addition, $\mathsf{LP}[\mathcal{P}]$ can be transformed to an equivalent logic program with at most two variables per rule. For example, the clause "$m(K, A, z) :- \ m(K_1, A_1, y_1),\ \ m(y_1, A_2, y_2),$" "$m(y_2, A_3, z)$" is equivalent to the two clauses "$m(K, A, z) \ :- \ m(K_1, A', y_2),\ \ m(y_2, A_3, z)$" and "$m(K_1, A', y_2) :- \ m(K_1, A_1, y_1),\ m(y_1, A_2, y_2)$" where A' is an identifier not appearing in $\mathcal{P}$.

Given a set $\mathcal{P}$ of 4-tuples with total size N, let L be the length of the longest extended name, one can first transform $\mathsf{LP}[\mathcal{P}]$ to contain only clauses that have at most two variables and then instantiate the clauses with principals in $\mathcal{P}$, obtaining a ground program with size $O(N^3L)$. It has been shown that the minimal Herbrand model of a ground logic program can be computed in time linear in the size of the program [5]. Therefore, any Horn query against $\mathsf{LP}[\mathcal{P}]$ can be answered in time $O(N^3L)$. This complexity is the same as the complexity bound derived in three papers [4, 22, 12] using algorithms based on string rewriting, graph searching, and pushdown systems.

2.4 A FOL Semantics

The LP semantics has the same limitation as the set-theoretic semantics, it cannot be used to directly determine whether S_1 rewrites into S_2. Such a query cannot be expressed using Horn queries. We now propose a first-order logic semantics to address this issue. The idea is very simple. Each Horn clause can be viewed as a first-order sentence; and the logic program $\mathsf{LP}[\mathcal{P}]$ can be viewed as a first-order theory. The rewriting query can be viewed as a first-order formula, and logical implication defines the semantics.

Definition 4 Given a set $\mathcal{P}$ of 4-tuples, we define $\mathsf{Th}[\mathcal{P}]$ to be the following first-order theory:

$$\{\forall z(\mathsf{contains}[KA][z] \Leftarrow \mathsf{contains}[S][z]) \mid K\ A \mapsto S \in \mathcal{P})\}$$

A query whether $\mathsf{RS}[\mathcal{P}] \rhd S_1 \overset{*}{\rightarrowtail} S_2$ can be answered by checking whether $\mathsf{Th}[\mathcal{P}] \models \forall z(\mathsf{contains}[S_1][z] \Leftarrow \mathsf{contains}[S_2][z])$. Note that $\mathsf{Th}[\mathcal{P}]$ is the same as $\mathsf{LP}[\mathcal{P}]$, here we just view it as a FOL theory.

Other first-order logic formulas can also be used as meaningful queries. For example, the formula $\exists z(m(K_1, A_1, z) \Leftarrow m(K_1, A_2, z)) \Leftarrow \exists z(m(K_2, A_1, z) \wedge m(K_2, A_2, z))$ means that if $K_2 A_1$ and $K_2 A_2$ shares a common member, then $K_1\, A_1$ and $K_1\, A_2$ also shares a common member.

2.5 Equivalence Among SDSI's Semantics

Proposition 1 says that the set-theoretic semantics is equivalent to the LP semantics for membership queries. The LP semantics can be viewed as a special case of the FOL semantics where only Horn queries are allowed. However, to the best of our knowledge, the relationship between the rewriting semantics and the other three semantics have not been established and proved in literature before.

In this section, we prove the following equivalence between the rewriting semantics and the FOL semantics: given any set $\mathcal{P}$ of 4-tuples, $\mathsf{RS}[\mathcal{P}] \rhd S_1 \overset{*}{\rightarrowtail} S_2$ if and only if $\mathsf{Th}[\mathcal{P}] \models \forall z(\mathsf{contains}[S_1][z] \Leftarrow \mathsf{contains}[S_2][z])$. In addition, we establish a way to use logic programs to efficiently determine whether $\mathsf{Th}[\mathcal{P}] \models \forall z(\mathsf{contains}[S_1][z] \Leftarrow \mathsf{contains}[S_2][z])$, and prove that this approach is correct.

First, we prove a proposition that will be useful in proving the main theorem.

Proposition 2 *Given a set $\mathcal{P}$ of policy statements, if $\mathsf{Th}[\mathcal{P}] \models m(K, A, K')$, then $RS(\mathcal{P}) \rhd K\ A \overset{*}{\rightarrowtail} K'$.*

See Appendix A for the proof.

We need the following definition, which helps in transforming a query $\mathsf{Th}[\mathcal{P}] \models \forall z(\mathsf{contains}[S_1][z] \Leftarrow \mathsf{contains}[S_2][z])$ into a Horn query. This definition gives a canonical way for placing a principal K' in the valuation of a name string S.

Definition 5 Given a set $\mathcal{P}$ of 4-tuples, a name string $S = K_1\ A_1\ A_2\ \cdots\ A_\ell$ where $\ell \geq 1$, and a principal K', define $add(\mathcal{P}, S, K')$ to be

$$\begin{cases} \mathcal{P} \cup \{K_1\ A_1 \mapsto K'\} & \text{when } l = 1 \\ \mathcal{P} \cup \{K_1\ A_1 \mapsto K'_1,\ K'_1\ A_2 \mapsto K'_2, & \\ \qquad \cdots,\ K'_{\ell-1}\ A_\ell \mapsto K'\} & \text{when } l > 1 \end{cases}$$

where $K'_1, \cdots, K'_{\ell-1}$ are principals not in $\mathcal{K}[\mathcal{P}] \cup \{K_1, K'\}$.

It is easy to see that $\mathsf{RS}[add(\mathcal{P}, K_1\, A_1\, A_2\, \cdots\, A_\ell, K')] \rhd K_1\, A_1\, A_2 \cdots A_\ell \overset{*}{\rightarrowtail} K'$. We are now ready to state the main theorem of this section.

Theorem 3 *Given a set $\mathcal{P}$ of 4-tuples, and two name strings S_1 and S_2, the following three statements are equivalent.*

1. $\mathsf{RS}[\mathcal{P}] \rhd S_1 \overset{*}{\rightarrowtail} S_2$.
2. $\mathsf{Th}[\mathcal{P}] \models \forall z(\mathsf{contains}[S_1][z] \Leftarrow \mathsf{contains}[S_2][z])$.
3. $\mathsf{Th}[\mathcal{P}'] \models m(K, A, K')$*, where $K\ A$ is any local name not defined in $\mathcal{P}$, and $\mathcal{P}'$ and K' depend on the form of S_2:*
   - *when S_2 is a principal, $K' = S_2$ and $\mathcal{P}' = \mathcal{P} \cup \{K\ A \mapsto S_1\}$*
   - *when $S_2 = K_1\ A_1\ \cdots\ A_\ell$ where $\ell \geq 1$, we set K' to be a principal not appearing in $\mathcal{P}$ and $\mathcal{P}' = add(\mathcal{P}, S_2, K') \cup \{K\ A \mapsto S_1\}$.*

See Appendix A for the proof. The equivalence of 2 and 3 in Theorem 3 says that to determine whether $\mathsf{Th}[\mathcal{P}] \models \forall z(\mathsf{contains}[S_1][z] \Leftarrow \mathsf{contains}[S_2][z])$, one can do the following: create a new principal K', find a local name $K\ A$ not defined in $\mathcal{P}$, construct $\mathcal{P}$' by adding to $\mathcal{P}$ the 4-tuple $K\ A \mapsto S_1$ and additional 4-tuples to make sure that S_2 rewrites into K', then check whether $m(K, A, K')$ follows from $\mathsf{LP}[\mathcal{P}']$.

The proof of the equivalence of 2 and 3 in Appendix A uses their relationships with the rewriting semantics. However, this equivalence also follows from general results in proof theory. Horn clauses and queries of the form $\forall z(\mathsf{contains}[S_1][z] \Leftarrow \mathsf{contains}[S_2][z])$ fall into a subclass of Harrop Hereditary formulas. Nadathur [25] showed that, for this subclass of formulas, classical provability, intuitionistic provability, and uniform provability are equivalent. Uniform proofs [24] are a restricted form of intuitionistic proofs that embody a special form of goal-directedness. Using uniform proofs, in order to determine whether $\mathsf{Th}[\mathcal{P}] \models \forall z \forall y_1 \forall y_2 (m(K, A, z) \Leftarrow m(K_1, A_1, y_1) \wedge m(y_1, A_2, y_2) \wedge m(y_2, A_3, z))$, one creates new constants K', K_1', K_2' that do not appear in $\mathcal{P}$, add to $\mathcal{P}$ three facts $m(K_1, A_1, K_1')$, $m(K_1', A_2, K_2')$, and $m(K_2', A_3, K')$, and then try to prove $m(K, A, K')$ from it. This is essentially what we are doing in Definition 5 and Theorem 3. The equivalence of classical provability and uniform provability for this class of queries says that this proof method is sound and complete with respect to classical first order logic. The equivalence of classical provability and intuitionistic provability for this class of queries says that one can also view the semantics as defined in intuitionistic logic rather than classical logic.

2.6 Comparing FOL Semantics with Other Semantics

Other logical semantics for SDSI exist. Both Abadi's logic [1] and Halpern and van der Meyden's Logic of Local Name Containment (LLNC) [8] are propositional modal logics. The core relation in their logic is the rewriting into relation. The focus of their study is to get a set of axioms characterizing their logic.

SDSI 1.1 [26] (the version of SDSI before merging with SPKI) gives pseudo-code of a nondeterministic algorithm for resolving a name string into a principal given a set of 4-tuples. Like the set-theoretic semantics, this algorithmic semantics defines a mapping from name strings to sets of principals. This semantics was used as the reference semantic in the logics by Abadi [1] and by Halpern and van der Meyden [8]. Li [16] showed that this semantics is equivalent to the string rewriting semantics when mapping name strings to principals. Li [16] gave another LP-based semantics for SDSI; the logic program there uses lists and is not Datalog; furthermore, it is not as natural as the LP semantics in Definition 3, and is not as easy to extend to FOL.

Comparing with the previous logic-based semantics for SDSI, such as LLNC, the FOL semantics has the following advantages:

- The FOL semantics directly captures the set-based semantic of SDSI, which is used widely as the underlying semantic intuition of SDSI, *e.g.*, both by SDSI designers in [4] and by Halpern and van der Meyden [8]. In addition, it uses classical first-order logic, instead of more complex logics with modal operators.
- It contains more information in the sense that a larger class of meaningful queries can be formulated and defined in this semantics.
- It is easily extended to support useful extensions of SDSI. There are two important ways of extending the SDSI 4-tuples. The first one is to add an intersection operator, *e.g.*, $K\ A \mapsto K\ A_1 \cap K\ A_2$. The meaning of such a statement is immediate given the set-based semantic intuition. Supporting intersection in the FOL semantics is straightforwardly done using the logical conjunction operator. Another way to extend SDSI is to allow identifiers to be logical terms, *e.g.*, K student(univ='Stanford') $\mapsto K_{\mathrm{John\_Smith}}$. In FOL semantics, this can be supported either by making the predicate m take more parameters or by having a predicate for each identifier function symbol, *e.g.*, using "student" as a predicate symbol. Extending logics such as LLNC to support these extensions may be possible, but seems less obvious.
- The FOL semantics has computational significance. The equivalence of 2 and 3 in Theorem 3 gives an efficient (in time polynomial in the size of $\mathcal{P}$) way to check whether $\mathsf{RS}[\mathcal{P}] \rhd S_1 \overset{*}{\rightarrowtail} S_2$ for any name strings S_1 and S_2. In LLNC, although a complete axiom system was given, the complexity of using the axiom system to determine whether $\mathsf{RS}[\mathcal{P}] \rhd S_1 \overset{*}{\rightarrowtail} S_2$ is not clear.

Some of these advantages also apply when comparing with the string rewriting semantics, *e.g.*, the FOL semantics contains more information and is more extendable. Because of this, we believe that the FOL semantics should be viewed as the reference semantics. Any semantics for SDSI should be equivalent to the FOL semantics for the class of queries it handles. Some other semantics are useful for computational or understanding purposes, of course. For example, the string rewriting semantics is both helpful for understanding SDSI and for computation, because of its relationship with pushdown systems.

3 RT_1^C: A TM Language Extending SDSI

In Section 2, we have seen that SDSI's semantics can be defined in several equivalent ways and can be computed efficiently. This shows that SDSI's design is natural and elegant. SDSI was originally designed as a scheme for naming. As such, it lacks important features when used as a full-fledged trust-management language. First, SDSI lacks intersection; one cannot define a local name to contain the intersection of two or more name strings. Many natural security policy statements have the form that a principal has a certain permission (or an attribute) if it has two (or more) other attributes at the same time. Second, SDSI identifiers do not have internal structures. For example, instead of saying "K student $\mapsto K_{\mathrm{John\_Smith}}$", it would be more useful to say "K student(dept='CS', program='PhD', year='2002', name='John Smith', $\cdots$) $\mapsto K_{\mathrm{John\_Smith}}$", because one can then use, for example, "K perm $\mapsto K$ student(dept='CS')" to grant "perm" to all principals that are CS students. Having internal structures in identifiers also enables one to represent relationships among principals. For example, one can then have local names such as "K managerOf(Alice)" and "K_1 physicianOf(Bob)" and issue statements such as "K accessRecord(?X) $\mapsto$ K_1 physicianOf(?X)", which means that the physician of any patient can access the record for that patient. ("?X" represents a logical variable.) Another use of having internal structures in identifiers is to represent access permissions that take parameters which identify resources and access modes.

SPKI/SDSI intends to be a full-fledged TM language; the approach taken in designing SPKI/SDSI can be viewed as adding SDSI features to SPKI, *e.g.*, allowing name strings to be used in subjects of auth certs. In Section 4, we will analyze SPKI/SDSI and argue that the design of SPKI/SDSI is problematic in several ways. In particular, authorization tags, which are used to qualify permissions in SPKI/SDSI, are quirky and algorithmically problematic. To make our analysis easier to follow, in this section we first look at another way of extending SDSI. Specifically, we look at RT_1^C [18], which is a language in the RT Role-based Trust-management framework [18, 20, 22]. RT consists of a family of TM languages. As pointed out in [20, 22], the design of RT is heavily influenced by SDSI and Delegation Logic [17]. RT_0, the most basic language in the RT family, can be viewed as adding intersections to SDSI. RT_1 adds to RT_0 parameters. RT_1^C further extends RT_1 with constraints. The semantics of the RT languages are based on translating statements into logical sentences, in the same style as SDSI's logic-based semantics.

3.1 Syntax of Policy Statements in RT_1^C

Before describing the syntax of policy statements in RT_1^C, we first explain the terminology differences between SDSI and RT. Identifiers in SDSI are called role terms in RT, and local names in SDSI are called roles. We use R, often with subscripts, to denote role terms, and we add a dot between a principal and a role term. For example, $K.R$ represents a role in RT; it corresponds to the local name K R in SDSI. Policy statements in RT_0, RT_1, and RT_1^C have the same structures. They only differ in how role terms are formed. In the following statements, the directions of the arrows are the opposite of that used in SDSI rewriting rules. We use this direction because it is the same as the direction of logical implication in the logic based semantics.

- *Type-1*: $K.R \longleftarrow K_1$
- *Type-2*: $K.R \longleftarrow K_1.R_1$
- *Type-3*: $K.R \longleftarrow K.R_1.R_2$
- *Type-4*: $K.R \longleftarrow K_1.R_1 \cap K_2.R_2 \cap \cdots \cap K_\ell.R_\ell$

A type-3 statement requires the same K to be used in both the role being defined, $K.R$, and the definition $K.R_1.R_2$. This design is motivated by the need to handle deduction with policy statements that are stored in a distributed manner. Observe that 4-tuples that use long extended names can still be equivalently represented in RT, by introducing new role terms and statements. For more details on this, see [22].

In RT_0, a role term is simply a role name, which is just like an identifier in SDSI. In RT_1, a role term may also contain parameters. These parameters may be constants or variables, and may use constraints in some limited ways. RT_1^C allows more general forms of role terms, which we now describe. In RT_1^C, each role name takes the form of $r(h_1, \ldots, h_n)$, in which r is a role name, and for each i such that $1 \leq i \leq n$, h_i takes one of the following three forms: $p = c$, $p \in S$, and $p = ref$, in which p is the name of one of r's parameters that has type τ, c is a constant of type τ, S is a value set of type τ, and ref is a reference to another parameter in the same statement, also of type τ. Intuitively, a value set is a constraint-based representation of a set of values, *e.g.*, $[10..800]$ may be a value set of an integer type. Parameters in role terms are specified by name, rather than by position; and they are strongly typed. For example, the following RT_1^C statement "$K_{\rm SA}$.socketPerm(host ∈ descendants('stanford.edu'), port∈ [8000..8443])⟵$K_{\rm Alice}$" means that $K_{\rm SA}$ grants to $K_{\rm Alice}$ the permission to connect to any host in the domain 'stanford.edu' at any port between 8000 and 8443.

3.2 Semantics of RT_1^C

Without constraints, policy statements in RT_1^C are translated into Datalog clauses (which can be viewed as first-order sentences), in ways very similar to SDSI. RT_1^C statements with constraints are translated into clauses in multi-sorted Constraint Datalog; these clauses can also be viewed as first-order sentences. In the following, we give an brief overview of Constraint Datalog. See [18] for more details of Constraint Datalog and for the translation from RT_1^C statements to clauses in Constraint Datalog.

Constraint Datalog (or DATALOG$^{\mathcal{C}}$) is a restricted form of Constraint Logic Programming (CLP), and is also a class of query languages for Constraint Databases (CDB) [13, 14]. The notion of constraint databases, which was introduced by Kanellakis, Kuper, and Revesz [13], grew out of the research on Datalog and CLP and generalizes the relational model of data by allowing infinite relations that are finitely representable using constraints. DATALOG$^{\mathcal{C}}$ allows first-order formulas in one or more constraint domains, which may describe file hierarchies, time intervals, and so on, to be used in the body of a rule. Intuitively, a constraint domain is a domain of objects, such as numbers, points in a plane, or files in a file hierarchy, together with a language for speaking about these objects. A constraint domain has a first-order language defined by a set of constants, function symbols, and relation symbols, and a class of quantifier-free formulas in the langauge, called primitive constraints. The following are some example constraint domains that are used when translating RT_1^C into DATALOG$^{\mathcal{C}}$.

Tree domains Each constant of a tree domain takes the form $\langle a_1, \ldots, a_k\rangle$. Imagine a tree in which every node is labelled with a string value. The constant $\langle a_1, \ldots, a_k\rangle$ represents the node for which $a_1, \ldots, a_k$ are the strings on the path from root to this node. A primitive constraint is of the form $x = y$ or $x\theta\langle a_1, \ldots, a_k\rangle$, in which $\theta \in \{=, <, \leq, \prec, \preceq\}$. The constraint $x < \langle a_1, \ldots, a_k\rangle$ means that x is a child of the node $\langle a_1, \ldots, a_k\rangle$, and $x \prec \langle a_1, \ldots, a_k\rangle$ means that x is a descendant of $\langle a_1, \ldots, a_k\rangle$.

Range domains The set of all constants in a range domain is linearly ordered. A primitive constraint has the form $x = y$, $x = c$ or $x \in (c_1, c_2)$, in which c is a constant, each of c_1 and c_2 is either a constant or a special symbol "$*$", meaning unbounded. And when c_1 is not $*$, "(" can also be "["; similarly, ")" can be "]" when c_2 is not $*$.

Discrete domains with sets A basic constraint has the form $x \in \{c_1, \ldots, c_\ell\}$, in which $c_1, \ldots, c_\ell$ are constants.

We say that these constraint domains are *unary*, because each primitive constraint either has the form $x = y$, where x and y are variables, or contains only one variable. We call a primitive constraint that contains just one variable a *basic constraint*. The three classes of unary constraint domains described above support the following three operations.

Conjunction Given two basic constraints $\phi_1(x)$ and $\phi_2(x)$, determine whether $\phi_1(x) \wedge \phi_2(x)$ is satisfiable, and if it is, compute $\psi_1(x) \vee \psi_2(x) \vee \cdots \vee \psi_k(x)$ such that $k \geq 1$, each $\psi_i(x)$ is a basic constraint, for $1 \leq i \leq k$, and the disjunction is logically equivalent to $\phi_1(x) \wedge \phi_2(x)$.

Constraint Projection Given any basic constraint, determine whether it is satisfiable.

Constraint Subsumption Given the disjunction of a set of basic constraints, determine whether another basic constraint is implied by the disjunction, *e.g.*, determine whether $x \in [2, 8]$ is implied by the disjunction of $x \in [1, 5]$ and $x \in [3, 10]$.

These operations make it possible to evaluate DATALOG$^{\mathcal{C}}$ programs using constraints in these domains. Li and Mitchell [18] have shown that tree domains, range domains, and discrete domains with sets have additional properties that make such evaluation tractable.

3.3 Types

Role terms in RT_1^C are strongly typed. One declares what parameters a role takes and the names and data types of these parameters. One also declares new data types using the mechanisms provided by RT. Role parameters and data types are declared in *application domain specification documents (ADSDs)*. Each ADSD is globally uniquely identified. One way to uniquely identify an ADSD is to use a collision-free hash of the document as the identifier; one may also include in the identifier an URI pointing to the document so that it can be easily retrieved. An ADSD declares a suite of related data types and role names, called a *vocabulary*. Policy statements, when using a role name, refer to the ADSD in which the role name is declared. This enables RT to have strongly typed policy statements. ADSDs also provide solutions to the following vocabulary agreement problem. For a statement K.student ⟵ K.university.student to make sense; every principal K' that is a member of the role K.university must agree with K on what "student" means, *e.g.*, whether it is a student registered in any class, or a student enrolled in a degree program. The use of ADSDs ensure that everyone is talking about the student role declared in one specific ADSD. The use of ADSDs and strongly typed statements helps reduce the possibility of errors in writing policy statements and unintended interaction among policy statements.

The notion of vocabularies is complementary to the notion of localized name spaces for roles. Each addresses a distinct name space issue. For example, an accrediting board might issue an ADSD that declares the role name "student". This defines the names and data types of the role's parameters. Such parameters may include university name, student name, program enrolled in, and so on. The ADSD may also contain description of the conditions under which a principal should be made a member of the student role, *e.g.*, it may require a principal be registered in a degree program. Then a university StateU can use this ADSD to issue credentials defining $K_{\mathsf{StateU}}.student$. Although using a vocabulary created by another principal, StateU is still the authority over who is a member of the role $K_{\mathsf{StateU}}.student$.

RT_1^C has several categories of types: integer types, float types, enumeration types, string types, tree types. Some system-defined data types of these categories exist, and one can declare new data types of these categories. Each type category has a syntax for defining *value sets*. Unordered enumeration types and string types correspond to discrete domains with sets; a value set takes the form of a set of constants. Integer types, float types, and ordered enumeration types correspond to range domains; a value set takes the form of a range, *e.g.*, $[10, 200]$, or $(10, *)$. Tree types correspond to tree domains; and are used to represent hierarchical structured resources such as file hierarchies and DNS names; a value set takes the form of a predefined function symbol applied to a node, *e.g.*, $\mathsf{descendants}(\langle \mathrm{edu}, \mathrm{stanford}\rangle)$ and $\mathsf{currentAndChildren}(\langle \mathrm{edu}, \mathrm{stanford}, \mathrm{cs}\rangle)$.

4 Analyzing SPKI/SDSI using First-Order-Logic (FOL)

In this section, we extend the FOL semantics for SDSI to SPKI/SDSI, which has authorization certificates (or auth certs) and ACL entries, in addition to name certs. Both auth certs and ACL entries are 5-tuples. An auth cert is a signed five tuple (K, H, D, T, V), where

- $K \in \mathcal{K}$ is the issuer principal, which signs the cert. The issuer grants a specific authorization through this 5-tuple.
- $H \in \mathcal{H}$ is called the subject, where $\mathcal{H}$ is defined to be the least set satisfying the following two conditions: (1) $\mathcal{S} \subseteq \mathcal{H}$, where S is the set of all name strings, and (2) $\theta_k(H_1, H_2, \cdots, H_n) \in \mathcal{H}$, where $H_1, H_2, \cdots, H_n \in \mathcal{H}$. The subject specifies principals that receive authorization from this 5-tuple.
- $D \in \{0, 1\}$ is called the delegation bit. When $D = 1$, the subject may further delegate the authorization it receives from this 5-tuple.
- T is the authorization tag, each tag represents a (potentially infinite set of) byte strings. T specifies the authorization that is granted by this 5-tuple.
- V is the validity specification, which is the same as in the case of a name cert.

An ACL entry is a locally stored 5-tuple $(\mathtt{Self}, H, D, T, V)$. It is very similar to an auth cert, except that the issuer is a special symbol `Self` instead of a key and that the ACL entry is not signed. We will treat `Self` as a special principal in $\mathcal{K}$.

The 5-tuple reduction rule in Section 6.3 of RFC 2693 is as follows: the two 5-tuples $(K_1, S_1, D_1, T_1, V_1)$ and $(K_2, S_2, D_2, T_2, V_2)$ yield $(K_1, S_2, D_2, \mathsf{AIntersect}(T_1, T_2), \mathsf{VIntersect}(V_1, V_2))$, provided that $S_1 = K_2$, $D_1 = 1$, and $\mathsf{AIntersect}(T_1, T_2)$ succeeds.

Observe that the 5-tuple reduction rule only applies when the subject is a principal. When the subject is a name string, the way to use the 5-tuple reduction rule is to first replace a 5-tuple that has a name string S as the subject with a set of 5-tuples, each of which has one principal in the valuation of the S as the subject. The procedure for handling threshold is described in details in Clarke et al. [4].

4.1 Handling Basic Delegation Relationships in 5-tuples

To understand 5-tuples, we first make some simplifying assumptions. These assumptions will be removed later one by one.

1. We assume that T does not have any internal structure, and can be viewed as an identifier. As a result, $\mathsf{AIntersect}(T_1, T_2)$ either fails (when $T_1 \neq T_2$) or equals $T_1 = T_2$.
2. We ignore the validity specification V, and write $(K, S, T, D, \cdot)$ for a 5-tuple. We assume that only valid certificates are considered.
3. We assume that the subject does not contain thresholds; in other words, the subject is a name string S.

When $D = 0$, a 5-tuple $(K, S, 0, T, \cdot)$ simply means that K grants the authorization T to any principal who is in the valuation of S. If we allow identifiers that correspond to authorization specifications to be used in 4-tuples, we can use $K\ T$ to denote the set of all principals that K grants the authorization T to, and view this 5-tuple as a 4-tuple $K\ T \mapsto S$.

When $D = 1$, a 5-tuple $(K, S, 1, T, \cdot)$ means that K grants the authorization to any principal that is in the valuation of S and to any principal such a principal grants T to. In other words, $(K, S, 1, T, \cdot)$ can be represented using two 4-tuples $K\ T \mapsto S$ and $K\ T \mapsto S\ T$. For example, a 5-tuple $(K, K_1\ A_1\ A_2, 1, T, \cdot)$ is represented using $K\ T \mapsto K_1\ A_1\ A_2$ and $K\ T \mapsto K_1\ A_1\ A_2\ T$.

Using 5-tuple reduction, one can use $(K_1, K_2, 1, T, \cdot)$ and $(K_2, K_3, 1, T, \cdot)$ to derive $(K_1, K_3, 1, T, \cdot)$. This can be viewed as deriving new rewriting rules from existing ones:

- rewrite $K_1\ T \mapsto K_2\ T$ using $K_2\ T \mapsto K_3$ and derive $K_1\ T \mapsto K_3$
- rewrite $K_1\ T \mapsto K_2 T$ using $K_2\ T \mapsto K_3 T$ and derive $K_1\ T \mapsto K_3\ T$

These observations show that, under these simplifying assumptions, 5-tuples can be viewed as 4-tuples, and we can use the FOL semantics for SDSI to provide a semantics for the simplified version of SPKI.

In the version of SPKI that existed before merging with SDSI, the subject of a 5-tuple cannot contain names. In this case a 5-tuple $(K, K_1, 1, T, V)$ can be represented using two 4-tuples $K\ T \mapsto K_1$ and $K\ T \mapsto K_1\ T$. This represents very limited delegation relationships, one cannot express $K\ T_1 \mapsto K_1\ T_2$ or the tuple $K\ T \mapsto K\ T_1\ T$, which represents a delegation about T from K to members of $K\ T_1$.

4.2 Using Constraints to Handle Authorization Tags and Validity Specifications

Having understood the delegation semantics of SPKI, we remove the first simplifying assumption and deal with authorization tags. Howell ([10] Chapter 6) provided a detailed analysis of authorization tags in his PhD thesis. In this analysis, an authorization tag T is viewed as representing an (often infinite) set of strings. Here, we take the same view and use constraints to specify the authorization represented as a tag. We use a unary constraint domain in which a basic constraint has the form $t \in T$, where t is a logical variable and T is an authorization tag. The mapping from tags to sets of strings then determines the truthfulness of formulas in this constraint domain. We will look at details of tags and this constraint domain in Section 4.3.

Definition 6 Given a set $\mathcal{P}$ of 4-tuples and 5-tuples, we define $\mathsf{Th}[\mathcal{P}]$ to be the following Constraint Datalog program (which is also a first-order theory). In addition to the predicate m used for 4-tuples, we also use another ternary predicate g; intuitively, $g(K, T, K')$ says that K grants the authorization T to K'. $\mathsf{Th}[\mathcal{P}]$ contains

$$\forall z(\mathsf{contains}[K\ A][z] \Leftarrow \mathsf{contains}[S][z]) \quad \text{for each} \quad (K, A, S, V) \in \mathcal{P}$$
$$\forall z \forall t(g(K, t, z) \Leftarrow \mathsf{contains}[S][z] \wedge t \in T) \quad \text{for each} \quad (K, S, 0, T, V) \in \mathcal{P}$$
$$\forall z \forall t(g(K, t, z) \Leftarrow \mathsf{contains}[S][z] \wedge t \in T) \quad \text{for each} \quad (K, S, 1, T, V) \in \mathcal{P}$$
$$\forall z \forall y \forall t(g(K, t, z) \Leftarrow \mathsf{contains}[S][y] \wedge g(y, t, z) \wedge t \in T) \quad \text{for each} \quad (K, S, 1, T, V) \in \mathcal{P}$$

RFC 2693 does not explicitly specify the class of queries in SPKI. However, because the 5-tuple reduction rule deduces new 5-tuples from existing ones, SPKI determines whether a 5-tuple follows from a set of statements. This is a reasonable "semantics" because authorization requests are also represented as 5-tuples. Because 5-tuples are represented using first-order formulas in our semantics, first-order logic naturally defines a semantic relation. Given a set $\mathcal{P}$ of 5-tuples and 4-tuples, if one wants to know whether a 5-tuple $(K, S, 0, T, V)$ follows from $\mathcal{P}$, one can ask whether $\mathsf{Th}[\mathcal{P}] \models \forall z \forall t(g(K, t, z) \Leftarrow \mathsf{contains}[S][z] \wedge t \in T)$. If one wants to know whether a 5-tuple $(K, S, 1, T, V)$ follows from $\mathcal{P}$, one should also check whether $\mathsf{Th}[\mathcal{P}] \models \forall z \forall y \forall t(g(K, t, z) \Leftarrow \mathsf{contains}[S][y] \wedge g(y, t, z) \wedge t \in T)$.

When $\mathsf{AIntersect}(T_1, T_2)$ is sound, *i.e.*, when $t \in \mathsf{AIntersect}(T_1, T_2) \Rightarrow t \in T_1 \wedge t \in T_2$ is a tautology for any pair of tags T_1 and T_2, the 5-tuple reduction rule is sound with respect to the logical semantic relation. For example, the 5-tuple reduction that uses $(K_1, K_2, 1, T_1, \cdot)$ and $(K_2, K_3, 0, T_2, \cdot)$ to derive $(K_1, K_3, 0, \mathsf{AIntersect}(T_1, T_2), \cdot)$ is essentially the following sound logical deduction: from $\forall z \forall t(g(K_1, t, z) \Leftarrow g(K_2, t, z) \wedge t \in T_1)$ and $\forall t(g(K_2, t, K_3) \Leftarrow t \in T_2)$, deduce $\forall t(g(K_1, t, K_3) \Leftarrow t \in \mathsf{AIntersect}(T_1, T_2))$.

Validity specifications can be handled by extending the two ternary predicates m and g to take an additional parameter v, which denotes the time during which this tuple is valid and viewing validity specifications as constraints on v. For example, validity specifications that are validity periods are straightforwardly represented using range constraints.

4.3 Structure of Tags

In this section, we look at the internal structure of tags. The main point of this section is that SPKI's 5-tuple reduction is an incomplete proof procedure with respect to the FOL semantics of SPKI. In addition, as pointed out before by Howell [10], authorization tags are not well behaved.

Specifically, the constraint domain for tags does not support operations that we need to compute the meaning of $\mathsf{Th}[\mathcal{P}]$.

The following descriptions of authorization tags follow RFC 2693. An authorization is a list of byte-strings or sublists. Two authorizations intersect by matching, element for element. If one list is longer than the other but matches at all elements where both lists have elements, then the longer list is the result of the intersection. This means that additional elements of a list must restrict the permission granted. For example, `(ftp (host ftp.clark.net))` may represent the permission of ftp access to every file and every directory on the host `ftp.clarke.net`. This is more general than `(ftp (host ftp.clark.net) (dir /pub/cme))`, and the intersection of the two tags results in the latter. SPKI also has a small number of special expressions.

`(*)` stands for the set of all tags and byte-strings. In other words, it will match anything. When intersected with another tag, the result is that other tag.

`(* set <tag-expr>*)` stands for the set of elements listed in the *-form.

`(* prefix <byte-string>)` stands for the set of all byte strings that start with the one given in the *-form.

`(* range <ordering> <lower-limit>? <upper-limit>?)` stands for the set of all byte strings lexically (or numerically) between the two limits. The ordering parameter (alpha, numeric, time, binary, date) specifies ordering.

We now show that 5-tuple reduction is incomplete with respect to the logical semantics, when interpreting tags as representing a set of strings. For example, given two 5-tuples $(K, K_1, 0,$ `(* set read write)`$, V)$ and $(K, K_1, 0,$ `(* set delete)`$, V)$, then the query $(K, K_1, 0,$ `(* set read delete)`$, V)$ should be true; but it cannot be inferred from the two 5-tuples by reduction. Intuitively, one should be able to combine authorizations received from multiple 5-tuples. However, the reduction rule uses only tag intersection and does not consider tag union. From this observation and the discussion in Section 4.2, we have the following theorem.

Theorem 4 *The 5-tuple reduction rule is a sound but incomplete procedure with respect to the first-order logic semantics, when tags are viewed as representation of sets of strings and* $\mathsf{AIntersect}(T_1, T_2)$ *is sound.*

A natural question to ask is whether there exists a sound and complete proof procedure for determining whether one 5-tuple follows from a set of 5-tuples. To do this, we have to look at the internal structures of tags and the operations on them. The short answer is that the constraint domain for representing tags is ill-behaved, and normal Constraint Datalog evaluation procedure does not apply. Howell [10] pointed out that intersections between some tags result in sets that may not be finitely represented using tags. This happens when intersecting a `(* prefix)` expression and a `(* range)` expression, or intersecting two two `(* range)` expressions that use different ordering. The example Howell gave is intersecting `(tag (* range numeric ge 0.5 le 0.5))` with `(tag (* prefix 000))`. Howell suggested artificially defining intersections in these potentially problematic cases to be empty, for lack of better solutions. This suggests that the constraint domain for authorization tags does not support the conjunction operation. It is also unclear how it can support the subsumption operation.

Bandmann and Dam [3] pointed out another problem with authorization tags: computing $\mathsf{AIntersect}$ of two tags that use `(* set)` may take time exponential in the size of the two tags.

4.4 Authorization Tags in SPKI vs. Named and Typed Parameters in RT_1^C

We have seen that, by viewing tags as representing sets of strings and viewing 4-tuples and 5-tuples as logical sentences, 5-tuple reduction is incomplete, and it seems unlikely that a complete proof procedure exists. This is clearly unsatisfactory. We now look for ways to remedy this problem.

SPKI/SDSI defines tags and their intersection in a syntactical way. Howell [10] provided a semantics for tags by viewing them as representations of sets of strings and found that "special tags cause havoc" (6.5.3 of [10]). We feel that the string semantics for tags is still syntactical in that it does not consider what the strings intend to represent. The `(* prefix)` tag, for example, is used for representing tree-like file hierarchies. If we consider the motivation and application for this construct, it seems more natural to represent statements about file hierarchies using multi-sorted first-order logic with a special sort devoted to files and directories. This approach leads to a constraint domain for tree-like hierarchies, namely, a tree domain, which we discussed in Section 3.2. The move to constraints over tree domains not only gives us a natural semantics, but leads to more expressive policy options. For example, `(* prefix)` is not helpful for referring to the DNS hierarchy in an access policy; we need a `(* postfix)` tag instead. Nor can one use `(* prefix)` to represent the set of all files and directories that are direct children of a directory, where as tree domains support these constraints. Similarly, `(* range)` tags are more naturally viewed as a form of constraints over range domains than sets of strings.

We believe that instead of viewing tags as defining sets of strings, and allowing combinations of string operations that do not meaningfully refer to any controlled resources, it is more informative and semantically appealing to use one special purpose constraint domain for each concept. The language RT_1^C does exactly this, allowing many choices of constraint domains. For example, in RT_1^C, one can declare one tree domain for file hierarchies, another tree domain for DNS names, one range domain for time of the day, and another range domain for port numbers, and so on. Each constraint domain has its standard semantic meaning, given by a first-order structure, and is easy to understand and handle algorithmically. In RT_1^C, a role term may have multiple named and typed parameters, each parameter from one constraint domain. This syntactic requirement allows different constraint domains to be combined in a single policy, without introducing meaningless statements that, for example, apply range predicates to tree expressions. A statement in RT_1^C can be translated into a Constraint Datalog clause with constraints from tree domains, range domains, or discrete domains with sets. In [18], we have proved that these domains are tractable and that evaluating the minimal models of Datalog programs with any multi-sorted combination of tractable domains remains tractable.

While we believe the multi-sorted language with multiple constraint domains is more appealing, there are combinations of SPKI/SDSI tags that cannot be translated into RT_1^C in any straightforward way. While many of these combinations do not seem meaningful or desirable, there are some aspects of SPKI/SDSI tag intersection that do seem useful and are not easily expressed in RT_1^C without special provision. The SPKI/SDSI AIntersect operator treats a longer tag as a more specialized tag. The design rationale for this to enable one to further specialize an authorization tag by appending new fields at the end. This flexibility seems to be missing in the strongly typed approach. The designers of the initial tags may not be able to foresee all necessary parameters; therefore, the initial type specification may not have all the parameters one wants. In RT_1^C, this flexibility is brought back by a feature that we call restrictive inheritance. A role name can be declared to restrictively inherit another role name and adds more parameters. Statements about the original role also implies statements about the extended role. For more details on this, see [19].

4.5 Threshold Subjects in SPKI vs. the Intersection Operator in RT_1^C

SPKI/SDSI lacks support for conjunction. As a result, it is not possible to directly grant a permission to any principal who has two or more attributes at the same time. Another use of conjunction is to allow a principal to manage a permission (by further delegating to other principals) without being able to use the permission. In RT_1^C, which has an intersection operator providing logical conjunction, $K.\mathsf{perm} \longleftarrow K_1.\mathsf{perm} \cap K.\mathsf{student}$ can be read as saying that K allows K_1 to delegate the authorization perm, but only to members of K.student. If K_1 is not a member of K.student, then K_1 cannot make itself a member of K.perm, even if it issues $K_1.\mathsf{perm} \longleftarrow K_1$.

Threshold subjects are part of SPKI/SDSI. If threshold subjects, which SPKI/SDSI only allows in 5-tuples, are also allowed in 4-tuples, they can be used to implement conjunction. For example, $\theta_2(K_1\ A_1,\ K_2\ A_2)$ can represent $K_1\ A_1 \cap K_2\ A_2$. Threshold subjects in 4-tuples are allowed in [6] and earlier versions of [7], but are not allowed in [4, 7], because they are viewed as "too convoluted to be useful in practice" [4]. As observed by Li [16], the meaning of threshold subjects in 5-tuples is different from that in 4-tuples. In a 4-tuple, k of the n subjects in a threshold subject must be resolved to a single principal. In a 5-tuple, k subjects can be resolved to different principals, which can further delegate to a single principal.

Coming up with a declarative semantics for threshold subjects in 5-tuples is not an easy task. Clarke et al. ([4], Section 10) uses a highly operational approach to handle thresholds. Halpern and van der Meyden [9] avoids threshold subjects in their logical reconstruction of SPKI. Intuitively, with threshold subjects, it is not a single principal that has some attributes (*e.g.*, being granted an authorization); instead, a set of principals together have some attributes. This set of principals can all delegate to one principal; they can also submit a joint access request.

In order to provide a logical approach to threshold subjects, the RT framework has the RT^T component, which handles threshold using manifold roles. *Manifold roles* generalize normal roles to allow each member to be a set of principals rather than a single principal. RT^T supports more expressive threshold structures and separation-of-duties policies, and has a logical semantics. See [20] for details on RT^T. An example policy that one can easily express in RT^T is requiring two different cashiers together to complete a transaction. To express this in SPKI, one has to explicitly list all the cashier principals in the threshold subject, and so the policy statement needs to be changed each time a new cashier is added or removed. Note that delegating to $\theta_2(K\,\mathsf{cashier},\ K\,\mathsf{cashier})$ is incorrect, since one single cashier principal satisfies the threshold in SPKI's semantics.

5 Conclusion

SPKI/SDSI is a language for expressing distributed access control policy, derived from SPKI and SDSI. Significant effort has gone into finding a logic-based semantics for both SDSI naming alone and SPKI/SDSI. We presented a FOL semantics for SDSI based on translating each name cert into a Datalog clause, viewed as a first-order logic sentence. We also proved that the FOL semantics is equivalent to the string rewriting semantics used by SDSI designers, for all queries associated with the rewriting semantics. The advantages of our approach over previous logics are the following. First-order logic captures the set-based semantic intuition of SDSI, and requires only classical first-order logic rather than more complex modal logics. The FOL semantics contains more information than rewriting semantics in the sense that a larger class of queries can be formulated and understood. This opens the door to more possibilities of safety and availability analysis, along the lines developed in [21]. The FOL semantics is easily extended to support useful extensions to SDSI and, finally, the relationship between the FOL semantics and logic programming provides an efficient method to answer a large class of queries.

In our study of full SPKI/SDSI, we use as comparison a trust management language RT_1^C [18], which has Constraint Datalog as its semantic foundation. RT_1^C is a language in the RT family of Role-based Trust-management languages [20, 22], and can be viewed as extending SDSI by adding several features. By adopting specific constraint domains tailored to SPKI/SDSI, we provided a FOL semantics for SPKI/SDSI in which authorization tags and validity specification are interpreted as logical constraints. This interpretation of SPKI/SDSI helped us examine several design issues, using RT_1^C for comparison. Our analysis shows that SPKI's 5-tuple reduction procedure is semantically incomplete. One reason is that reduction does not handle union of tags. In addition, authorization tags in SPKI/SDSI are algorithmically problematic, making a complete proof procedure unlikely. The constraint feature of RT_1^C provides an alternative mechanism that is often more expressive than SPKI/SDSI tags, semantically natural, and algorithmically tractable. The translation of SPKI/SDSI into logic with constraints, and experience with RT_1^C, suggests that Constraint Datalog is an appropriate foundation for trust management languages, subsuming SPKI/SDSI and possessing several algorithmic and expressiveness advantages.

Acknowledgement

This work is supported by DARPA through SPAWAR contract N66001-00-C-8015, by DOD MURI "Semantics Consistency in Information Exchange" as ONR Grant N00014-97-1-0505, and by DOD University Research Initiative (URI) program administered by the Office of Naval Research under Grant N00014-01-1-0795. We also thank the anonymous reviewers for their helpful comments.

A Proofs

Proposition 2 *Given a set $\mathcal{P}$ of policy statements, if $\mathsf{Th}[\mathcal{P}] \models m(K, A, K_1)$, then $RS(\mathcal{P}) \rhd K\ A \overset{*}{\rightarrowtail} K_1$.*

Proof. From standard result in logic programming, $\mathsf{Th}[\mathcal{P}] \models m(K, A, K_1)$ if and only if $m(K, A, K_1)$ is in the minimal Herbrand model of $\mathsf{Th}[\mathcal{P}]$. We now summarize a standard fixpoint characterization of the minimal Herbrand model, which we will use in this proof. For a Datalog program $\mathcal{DP}$, let $\mathcal{DP}^{inst}$ be the ground instantiation of $\mathcal{DP}$ using constants in $\mathcal{DP}$, the *immediate consequence operator*, $T_{\mathcal{DP}}$, is defined as follows. Given a set of ground logical atoms K, $T_{\mathcal{DP}}(K)$ consists of all logical atoms, a, such that $a{:-}\ b_1, \ldots, b_n \in \mathcal{DP}^{inst}$ and $b_j \in K$ for $1 \leq j \leq n$. The least fixpoint of $T_{\mathcal{DP}}$ can be constructed as follows. Define $T_{\mathcal{DP}}{\uparrow}^0 = \emptyset$ and $T_{\mathcal{DP}}{\uparrow}^{i+1} = T_{\mathcal{DP}}(T_{\mathcal{DP}}{\uparrow}^i)$ for $i \geq 0$. This defines an increasing sequence of subsets of a finite set. Thus there exists an N such that $T_{\mathcal{DP}}(T_{\mathcal{DP}}{\uparrow}^N) = T_{\mathcal{DP}}{\uparrow}^N$. $T_{\mathcal{DP}}{\uparrow}^N$ is easily shown to be the least fixpoint of $T_{\mathcal{DP}}$, which we denote by $T_{\mathcal{DP}}{\uparrow}^\omega$. $T_{\mathcal{DP}}{\uparrow}^\omega$ is identical to the minimal Herbrand model of $\mathcal{DP}$ [23]; therefore, $\mathsf{LP}[\mathcal{P}] \models m(X, u, Z)$ if and only if $m(X, u, Z) \in T_{\mathsf{LP}[\mathcal{P}]}{\uparrow}^\omega$.

We prove this proposition by using induction on i to show that if $m(K, A, K') \in T_{\mathsf{LP}[\mathcal{P}]}{\uparrow}^i$, then $RS(\mathcal{P}) \rhd K\ A \overset{*}{\rightarrowtail} K'$. The basis is trivially satisfied because $T_{\mathsf{LP}[\mathcal{P}]}{\uparrow}^0 = \emptyset$. In the step, $m(K, A, K') \in T_{\mathsf{LP}[\mathcal{P}]}{\uparrow}^{i+1}$, one of the following three cases apply.

Case one: $m(K, A, K') \in \mathsf{LP}[\mathcal{P}]$, this means that $K\ A \mapsto K' \in \mathcal{P}$. Clearly, $\mathsf{RS}[\mathcal{P}] \rhd K\ A \overset{*}{\rightarrowtail} K'$.

Case two: $m(K, A, z){:-}\ m(K_1, A_1, z) \in \mathsf{LP}[\mathcal{P}]$, and $m(K_1, A_1, K') \in T_{\mathsf{LP}[\mathcal{P}]}{\uparrow}^i$. In this case, $K\ A \mapsto K_1\ A_1 \in \mathcal{P}$, and by induction hypothesis, $\mathsf{RS}[\mathcal{P}] \rhd K_1\ A_1 \overset{*}{\rightarrowtail} K'$. Using rewriting rules in $\mathsf{RS}[\mathcal{P}]$, one can rewrite $K\ A$ first to $K_1\ A_1$, and then to K'; so $\mathsf{RS}[\mathcal{P}] \rhd K\ A \overset{*}{\rightarrowtail} K'$.

Case three: $m(K, A, z) \ :-\ m(K_1, A_1, y_1), m(y_1, A_2, y_2), \cdots, m(y_{\ell-1}, A_\ell, z) \in \mathsf{LP}[\mathcal{P}]$, $\ell > 1$, and $m(K_1, A_1, K_1'), m(K_1', A_2, K_2'), \cdots, m(K_{\ell-1}', A_\ell, K') \in T_{\mathsf{LP}[\mathcal{P}]}{\uparrow}^i$. In this case, $K\ A \mapsto K_1\ A_1 \cdots A_\ell \in \mathcal{P}$ and by induction hypothesis, $\mathsf{RS}[\mathcal{P}] \rhd K_1\ A_1 \overset{*}{\rightarrowtail} K_1',\ K_1'\ A_2 \overset{*}{\rightarrowtail} K_2', \cdots, K_{\ell-1}'\ A_\ell \overset{*}{\rightarrowtail} K'$. Using rewriting rules in $\mathsf{RS}[\mathcal{P}]$, one can rewrite $K\ A$ first to $K_1\ A_1 \cdots A_\ell$, then into $K_1'\ A_2 \cdots A_\ell$, and so on, and finally into K'.

∎

Theorem 3 *Given a set $\mathcal{P}$ of 4-tuples, and two name strings S_1 and S_2, the following three statements are equivalent.*

1. $\mathsf{RS}[\mathcal{P}] \rhd S_1 \overset{*}{\rightarrowtail} S_2$.
2. $\mathsf{Th}[\mathcal{P}] \models \forall z(\mathsf{contains}[S_1][z] \Leftarrow \mathsf{contains}[S_2][z])$.

3. $\mathsf{Th}[\mathcal{P}'] \models m(K,A,K')$, *where* $K\ A$ *is any local name not defined in* $\mathcal{P}$, *and* $\mathcal{P}'$ *and* K' *depend on the form of* S_2*:*

   - *when* S_2 *is a principal,* $K' = S_2$ *and* $\mathcal{P}' = \mathcal{P} \cup \{K\ A \mapsto S_1\}$
   - *when* $S_2 = K_1\ A_1\ \cdots\ A_\ell$ *where* $\ell \geq 1$, *we set* K' *to be a principal not appearing in* $\mathcal{P}$ *and* $\mathcal{P}' = add(\mathcal{P}, S_2, K') \cup \{K\ A \mapsto S_1\}$.

Proof. **1 implies 2**: We use induction on the number of rewriting steps. Base case, $S_1 = S_2$, the formula $\forall z(\mathsf{contains}[S_1][z] \Leftarrow \mathsf{contains}[S_2][z])$ is a tautology. Consider the step, suppose that $\mathsf{RS}[\mathcal{P}] \rhd S_1 \rightarrowtail S' \overset{*}{\rightarrowtail} S_2$. We will prove that $\mathsf{Th}[\mathcal{P}] \models \forall z(\mathsf{contains}[S_1][z] \Leftarrow \mathsf{contains}[S'][z])$. Induction hypothesis gives us $\mathsf{Th}[\mathcal{P}] \models \forall z(\mathsf{contains}[S'][z] \Leftarrow \mathsf{contains}[S_2][z])$. Combining them, we have $\mathsf{Th}[\mathcal{P}] \models \forall z(\mathsf{contains}[S_1][z] \Leftarrow \mathsf{contains}[S_2][z])$.

We now prove that $\mathsf{Th}[\mathcal{P}] \models \forall z(\mathsf{contains}[S_1][z] \Leftarrow \mathsf{contains}[S'][z])$ when $\mathsf{RS}[\mathcal{P}] \rhd S_1 \overset{*}{\rightarrowtail} S'$. S_1 is either a principal, a local name, or an extended name. S_1 cannot be a principal, since then the rewriting from S_1 to S' is impossible. If S_1 is a local name, then $S_1 \mapsto S' \in \mathcal{P}$, and so $\forall z(\mathsf{contains}[S_1][z] \Leftarrow \mathsf{contains}[S'][z]) \in \mathsf{Th}[\mathcal{P}]$. If S_1 is an extended name $K\ A_1\ \cdots\ A_\ell$ for some $\ell \geq 2$, we assume that the 4-tuple used to rewrite S_1 into S' is $K\ A_1 \mapsto K_1\ B_1\ \cdots\ B_k$ for some $k \geq 0$, then $S' = K_1\ B_1\ \cdots\ B_k\ A_2\ \cdots\ A_\ell$. Consider any model of $\mathsf{Th}[\mathcal{P}]$ and any principal z in the model, if $\mathsf{contains}[S'][z]$ is true in the model, then there exists principals $y'_1, \cdots, y'_k$ and $y_2, \ldots, y_\ell$ such that $m(K_1, B_1, y'_1)$, $m(y'_1, B_2, y'_2)$, $m(y'_{k-1}, B_k, y'_k)$, $m(y'_k, A_2, y_2)$, $\cdots$, $m(y_{\ell-1}, A_\ell, z)$ are true in the model. Since $K\ A_1 \mapsto K_1\ B_1\ \cdots\ B_k$, then $\mathsf{Th}[\mathcal{P}] \models \forall z(\mathsf{contains}[K\ A_1][z] \Leftarrow \mathsf{contains}[K_1\ B_1\ \cdots\ B_k][z])$, therefore $m(K, A_1, y'_k)$ is true in the model. It then follows that $\mathsf{contains}[K\ A_1\ \cdots\ A_\ell][z]$ is true in the model. Therefore, $\mathsf{Th}[\mathcal{P}] \models \forall z(\mathsf{contains}[S_1][z] \Leftarrow \mathsf{contains}[S'][z])$.

2 implies 3: Assume, without loss of generality, that S_1 is $K\ A_1\ \cdots\ A_\ell$ for some $\ell \geq 0$ and S_2 is $K_1\ B_1\ \cdots\ B_k$ from some $k \geq 0$. Observe that by definition of $\mathcal{P}'$, $\mathsf{Th}[\mathcal{P}'] \models \mathsf{contains}[S_2][K']$. When S_2 is a principal, $S_2 = K'$. When S_2 is a local name or an extended name, $\mathsf{Th}[\mathcal{P}']$ contains the following k atoms $m(K_1, B_1, K'_1)$, $m(K'_1, B_2, K'_2)$, $\cdots$, $m(K'_{k-1}, B_k, K')$. If $\mathsf{Th}[\mathcal{P}] \models \forall z(\mathsf{contains}[S_1][z] \Leftarrow \mathsf{contains}[S_2][z])$, then $\mathsf{Th}[\mathcal{P}'] \models \mathsf{contains}[S_1][K']$. Also observe that $\mathsf{Th}[\mathcal{P}'] \models \forall z(\mathsf{contains}[K\ A][z] \Leftarrow \mathsf{contains}[S_1][z])$; therefore, $\mathsf{Th}[\mathcal{P}'] \models m(K, A, K')$.

3 implies 1 From Proposition 2, we know that if $\mathsf{Th}[\mathcal{P}'] \models m(K, A, K')$, then $\mathsf{RS}[\mathcal{P}'] \rhd K\ A \overset{*}{\rightarrowtail} K'$. Consider the rewriting sequence, the first rule applied has to be $K\ A \mapsto S_1$, since it is the only rule that can apply. (Recall that by definition $K\ A$ is not defined in $\mathcal{P}$.) Consider the last step, the rule applied has to be $K'_{\ell-1}\ A_\ell \mapsto K'$, since that is the only rule having a K' on its right hand side. The rule applied in the second to last step has to be $K'_{\ell-2}\ A_{\ell-1} \mapsto K'_{\ell-1}$, since that is the only rule having $K'_{\ell-1}$ on its right-hand side, and so on. Therefore, the rewriting sequence must contains in its middle a sequence rewriting from S_1 to S_2. Further observe that the rules in $\mathcal{P}'$ but not in $\mathcal{P}$ cannot be applied in this middle sequence. Therefore, $\mathsf{RS}[\mathcal{P}] \rhd S_1 \overset{*}{\rightarrowtail} S_2$. ∎

References

[1] Martín Abadi. On SDSI's linked local name spaces. *Journal of Computer Security*, 6(1–2):3–21, 1998.

[2] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, October 1993.

[3] Olav Bandmann and Mads Dam. A note on SPKI's authorization syntax. In *Pre-Proceedings of 1st Annual PKI Research Workshop*, April 2002. Available from http://www.cs.dartmouth.edu/∼pki02/.

[4] Dwaine Clarke, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Morcos, and Ronald L. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.

[5] William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.

[6] Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. SPKI certificate theory. IETF RFC 2693, September 1999.

[7] Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. Simple public key certificates. Internet Draft (work in progress), July 1999. Available at http://world.std.com/∼cme/spki.txt.

[8] Joseph Halpern and Ron van der Meyden. A logic for SDSI's linked local name spaces. *Journal of Computer Security*, 9(1-2):47–74, 2001.

[9] Joseph Halpern and Ron van der Meyden. A logic reconstruction of SPKI. In *Proceedings of the 14th IEEE*

Computer Security Foundations Workshop, pages 59–70. IEEE Computer Society Press, June 2001.

[10] Jonathan R. Howell. *Naming and sharing resources acroos administrative boundaries*. PhD thesis, Dartmouth College, May 2000.

[11] Jonathan R. Howell and David Kotz. A formal semantics for SPKI. In *Proceedings of the Sixth European Symposium on Research in Computer Security (ESORICS 2000)*, pages 140–158. Springer, October 2000.

[12] Somesh Jha and Thomas Reps. Analysis of SPKI/SDSI certificates using model checking. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 129–144. IEEE Computer Society Press, June 2002.

[13] Paris C. Kanellakis, Gabriel M. Kuper, and Peter Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):26–52, August 1995.

[14] Gabriel Kuper, Leonid Libkin, and Jan Paredaens, editors. *Constraint Databases*. Springer, 2000.

[15] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, November 1992.

[16] Ninghui Li. Local names in SPKI/SDSI. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 2–15. IEEE Computer Society Press, July 2000.

[17] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. Delegation Logic: A logic-based approach to distributed authorization. *ACM Transaction on Information and System Security (TISSEC)*, 6(1):128–171, February 2003.

[18] Ninghui Li and John C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages (PADL 2003)*, pages 58–73. Springer, January 2003.

[19] Ninghui Li, John C. Mitchell, Yu Qiu, William H. Winsborough, Kent E. Seamons, Michael Halcrow, and Jared Jacobson. RTML: A Role-based Trust-management Markup Language, August 2002. Unpublished manuscript. Available at http://crypto.stanford.edu/∼ninghui/papers/rtml.pdf.

[20] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.

[21] Ninghui Li, William H. Winsborough, and John C. Mitchell. Beyond proof-of-compliance: Safety and availability analysis in trust management. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2003. To appear.

[22] Ninghui Li, William H. Winsborough, and John C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, February 2003.

[23] John W. Lloyd. *Foundations of Logic Programming, Second Edition*. Springer, 1987.

[24] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. *Uniform Proofs as a Foundation for Logic Programming*, volume 51 of *Annals of Pure and Applied Logic*, pages 125–157. Elsevier Science, 1991.

[25] Gopalan Nadathur. Correspondences between classical, intuitionistic and uniform provability. *Theoretical Computer Science*, 232(1-2):273–298, 2000.

[26] Ronald L. Rivest and Bulter Lampson. SDSI — a simple distributed security infrastructure, October 1996. Available at http://theory.lcs.mit.edu/∼rivest/sdsi11.html.