Weighted Pushdown Systems and Trust-Management Systems

Somesh Jha[1], Stefan Schwoon[2], Hao Wang[1], and Thomas Reps[1]

[1] *Computer Science Department, University of Wisconsin, Madison, WI 53706*
{hbwang, jha, reps}@cs.wisc.edu
[2] *Institut für Formale Methoden der Informatik, Universität Stuttgart,*
Universitätsstr. 38, 70569 Stuttgart, Germany
schwoosn@fmi.uni-stuttgart.de

Abstract. The authorization problem is to decide whether, according to a security policy, some principal should be allowed access to a resource. In the trust-management system SPKI/SDSI, the security policy is given by a set of certificates, and proofs of authorization take the form of certificate chains. The certificate-chain-discovery problem is to discover a proof of authorization for a given request. Certificate-chain-discovery algorithms for SPKI/SDSI have been investigated by several researchers. We consider a variant of the certificate-chain discovery problem where the certificates are distributed over a number of servers, which then have to cooperate to identify the proof of authorization for a given request. We propose two protocols for this purpose. These protocols are based on distributed model-checking algorithms for weighted pushdown systems (WPDSs). These protocols can also handle cases where certificates are labeled with weights and where multiple certificate chains must be combined to form a proof of authorization. We have implemented these protocols in a prototype and report preliminary results of our evaluation.

1 Introduction

In access control of shared computing resources, the *authorization problem* addresses the following question: "Given a security policy, should a principal be allowed access to a specific resource?" In trust-management systems [4, 5, 25], such as SPKI/SDSI [9], the security policy is given by a set of signed certificates, and a proof of authorization consists of a set of certificate chains. In SPKI/SDSI, the *principals are the public keys*, i.e., the identity of a principal is established by checking the validity of the corresponding public key. In SPKI/SDSI, *name certificates* define the names available in an issuer's local name space; *authorization certificates* grant authorizations, or delegate the ability to grant authorizations. The *certificate-chain-discovery problem* is to discover a set of certificate chains that provides a proof of authorization for a request by a principal to access a resource.

An efficient certificate-chain-discovery algorithm for SPKI/SDSI was presented by Clarke et al. [8]. An improved algorithm was presented by Jha and Reps [14]. The latter algorithm is based on translating SPKI/SDSI certificates to rules in a pushdown system [10, 11]. In [14] it was also demonstrated how this translation enables many other questions to be answered about a security policy expressed as a set of certificates. Algorithms presented in [8] and [14] assume that the proof of authorization consists of a *single* certificate chain. In general, however, a proof of authorization in SPKI/SDSI requires a *set* of certificate chains, each of which proves some *part* of the required authorization. Hence, the certificate-chain-discovery algorithms presented in [8, 14] are incomplete. This observation is also the basis for the observation by Li and Mitchell [19] that the "5-tuple reduction rule" of [9] is incomplete.

Schwoon et al. [24] introduced a new algorithm for certificate-chain discovery that translates SPKI/SDSI certificates to rules in a weighted pushdown system (WPDS) [22]. The algorithm presented by Schwoon et al. [24] can discover proofs of authorization that consist of multiple certificate chains. Moreover, the algorithm presented in [24] addresses such issues as trust, privacy, and recency in the context of authorization in SPKI/SDSI. As in [24], in this paper we translate SPKI/SDSI certificates into rules in a WPDS, where the authorization specifications of the certificates are translated to weights on rules. This translation to a WPDS yields a complete certificate-chain-discovery algorithm and is described in Section 5.

The algorithms of [8, 14, 24] assume that the set of all certificates relevant to a given request are known to a single site, which can then compute the answer to the authorization problem for a given principal and a given resource. In practice, however, there may be no such central authority. Certificates may be held by a number of different sites, each of which knows only a subset of the certificates. If a principal K from site S_1 wants to access a resource at site S_2, the certificate chain authorizing K to do so may involve certificates from both S_1 and S_2 (and possibly a number of other sites in between). For instance, consider the following example: The Computer Sciences department (CS) at the University of Wisconsin (UW) is part of the College of Letters and Sciences (LS). The department, the college, and the university could be different sites in the sense above. UW might grant access to some resource R to all of its faculty members by issuing a corresponding authorization certificate. The actual principals authorized to access R would be specified by name certificates, e.g., UW would declare that its faculty members are (among others) those of LS, LS would declare that its faculty members are (among others) those of CS, and CS would have a list of its faculty members. If members of CS want to access R, they need a chain of certificates from UW, LS, and CS, and none of these sites may know all of the certificates involved.

This paper makes two major contributions. First, we present a distributed model-checking algorithm for WPDSs. Second, using this algorithm we develop a distributed certificate-chain-discovery algorithm for SPKI/SDSI where the certificates are distributed across various sites. Background on the trust-management system SPKI/SDSI is given in Section 4. A distributed certificate-chain-discovery algorithm for SPKI/SDSI is described in Section 6. We have implemented a prototype of our algorithm. Our experimental results, presented in Section 7, demonstrate that the algorithm incurs a moderate overhead.

2 Related Work

A certificate-chain-discovery algorithm for SPKI/SDSI was first proposed by Clarke et al. [8]. An improved certificate-chain-discovery based on the theory of pushdown systems was presented by Jha and Reps [14]. As indicated earlier, both of these algorithms are centralized and assume that the proof of authorization consists of a single certificate chain. In the proof-carrying-authorization (PCA) framework of Appel and Felten [2], a client uses the theorem prover *Twelf* [21] to construct a proof of authorization, which the client presents to the server. However, they too assume that all logical facts used by the theorem prover reside at a single server. Li et al. [20] presented a distributed certificate-chain-discovery algorithm for the trust-management system RT_0. Their algorithm allows certificates to be distributed, but the proof of authorization is maintained at one site. SPKI/SDSI is a subset of RT_0 (SPKI/SDSI is equivalent to RT_0 without role intersection). In our distributed certificate-chain-discovery algorithm, various sites summarize their part of the proof of authorization before sending it to other sites; thus, the proof of authorization is distributed. Moreover, summarizing intermediate results also provides some privacy. We also implemented our algorithm in a trust-management server. To the best of our knowledge, Li et al. did not implement their algorithm. Bauer et al. [3] present an algorithm for assembling a proof that a request satisfies an access-control policy expressed in formal logic [18]. Bauer et al. advocate a lazy strategy, in which a party enlists help of others to prove particular subgoals. The precise relationship between the distributed algorithm of Bauer et al. and the algorithm presented in this paper will be explored in the future. The semantics of SPKI/SDSI has been widely studied [13, 1, 12]. In this context, the work that is most relevant is by Li and Mitchell [19], who pointed out that the "5-tuple reduction rule" of [9] is incomplete because, in general, a proof of authorization can require multiple certificate chains. Our algorithm does not suffer from this problem, due to the translation into a WPDS.

The work by Jim and Suciu on SD3 [16, 17], the successor of QCM, is also related to ours. SD3 is a trust-management system based on Datalog that, like our algorithms, allows for distributed evaluation of authorization queries. In [16], the author claims that SD3 can express "roughly the same policies as SDSI 2". While this claim is not further substantiated in [16], we believe it to be true. However, there are several differences that set our work apart from SD3:

- SD3 describes a generic evaluation algorithm where each instantiation corresponds to a particular strategy for distributing the computation. We propose several concrete evaluation strategies and argue that these strategies have certain advantages with respect to efficiency and privacy.
- Since [16] does not provide a concrete encoding of SPKI/SDSI in SD3, any comparison of the relative merits of our encoding vs SD3's is bound to be speculative. However, we believe that SD3's *site-safety* requirement would limit their evaluation to "forward" mode, whereas our algorithms can search both forward and backward (the latter is explained in Section 6).
- Unlike SD3, our framework allows certificates to have weights. As pointed out in [15], this provides a solution for situations in which proofs of authorization require multiple certificate chains, each of which prove *part* of the authorization. This solves the problem of semantic incompleteness pointed out by Li and Mitchell [19]. Moreover, in [24], we pointed out that weights allow to address such issues as privacy, recency, validity, and trust.

3 Weighted Pushdown Systems

Weighted pushdown systems were introduced in [7, 22–24]. In short, a pushdown system defines an infinite-state transition system whose states involve a stack of unbounded length. In a weighted pushdown system, the rules are given values from some domain of weights. Our weight domains of interest are the bounded idempotent semirings defined in Defn. 1.

Definition 1. *A* **bounded idempotent semiring** *is a quintuple* $(D, \oplus, \otimes, 0, 1)$*, where D is a set, 0 and 1 are elements of D, and $\oplus$ (the combine operation) and $\otimes$ (the extend operation) are binary operators on D such that*

1. *$(D, \oplus)$ is a commutative monoid whose neutral element is 0, and where $\oplus$ is idempotent.*
2. *$(D, \otimes)$ is a monoid with the neutral element 1.*
3. *$\otimes$ distributes over $\oplus$, i.e., for all $a, b, c \in D$ we have $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ and $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$.*
4. *0 is an annihilator with respect to $\otimes$, i.e., for all $a \in D$, $a \otimes 0 = 0 = 0 \otimes a$.*
5. *In the partial order $\sqsubseteq$ defined by: $\forall a, b \in D,\ a \sqsubseteq b$ iff $a \oplus b = a$, there are no infinite descending chains.*

Definition 2. *A* **pushdown system** *is a triple $\mathcal{P} = (P, \Gamma, \Delta)$, where P and Γ are finite sets called the* **control locations** *and the* **stack alphabet***, respectively. The elements of $Conf(\mathcal{P}) := P \times \Gamma^*$ are called the* **configurations** *of $\mathcal{P}$. Δ contains a finite number of* **rules** *of the form $\langle p, \gamma \rangle \hookrightarrow_{\mathcal{P}} \langle p', w \rangle$, where $p, p' \in P$, $\gamma \in \Gamma$, and $w \in \Gamma^*$, which define a transition relation $\Rightarrow_{\mathcal{P}}$ between configurations of $\mathcal{P}$ as follows:*

$$\text{If } r = \langle p, \gamma \rangle \hookrightarrow_{\mathcal{P}} \langle p', w \rangle, \text{ then } \langle p, \gamma w' \rangle \xRightarrow{\langle r \rangle}_{\mathcal{P}} \langle p', ww' \rangle \text{ for all } w' \in \Gamma^*.$$

We write $c \Rightarrow_{\mathcal{P}} c'$ to express that there exists some rule r such that $c \xRightarrow{\langle r \rangle}_{\mathcal{P}} c'$; we omit the subscript $\mathcal{P}$ if $\mathcal{P}$ is understood. The reflexive transitive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$.

Given a set of configurations C, we define $pre(C) \stackrel{\text{def}}{=} \{ c' \mid \exists c \in C \colon c' \Rightarrow c \}$ and $post(C) \stackrel{\text{def}}{=} \{ c' \mid \exists c \in C \colon c \Rightarrow c' \}$ as the sets of configurations that are reachable—backwards and forwards, respectively—from elements of C in a single step. Moreover, $pre^*(C) \stackrel{\text{def}}{=} \{ c' \mid \exists c \in C \colon c' \Rightarrow^* c \}$ and $post^*(C) \stackrel{\text{def}}{=} \{ c' \mid \exists c \in C \colon c \Rightarrow^* c' \}$ are the configuration reachable–backwards and forwards–in arbitrarily many steps. C is called **regular** *if for all $p \in P$ the language $\{ w \mid \langle p, w \rangle \in C \}$ is regular.*

Definition 3. *A* **weighted pushdown system** *is a triple $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$ such that $\mathcal{P} = (P, \Gamma, \Delta)$ is a pushdown system, $\mathcal{S} = (D, \oplus, \otimes, 0, 1)$ is a bounded idempotent semiring, and $f \colon \Delta \to D$ is a function that assigns a value from D to each rule of $\mathcal{P}$.*

Let $\sigma \in \Delta^*$ be a sequence of rules. Using f, we can associate a value to σ, i.e., if $\sigma = [r_1, \ldots, r_k]$, then we define $v(\sigma) \stackrel{\text{def}}{=} f(r_1) \otimes \ldots \otimes f(r_k)$. Moreover, for any two configurations c and c' of $\mathcal{P}$, we let $path(c, c')$ denote the set of all rule sequences $[r_1, \ldots, r_k]$ that transform c into c', i.e., $c \xRightarrow{\langle r_1 \rangle} \cdots \xRightarrow{\langle r_k \rangle} c'$.

Definition 4. *Let $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$, where $\mathcal{P} = (\mathcal{P}, \Gamma, \Delta)$ and $\mathcal{S} = (D, \oplus, \otimes, 0, 1)$, and let C be a set of configurations. A* **forwards** *(resp.* **backwards***) $(\mathcal{W}, C)$*-**dag** *is an edge-labeled directed acyclic graph (V, E) where $V \subseteq \mathit{Conf}(\mathcal{P}) \times D$ and $E \subseteq V \times \Delta \times V$ such that*

- *if a vertex (c, d) has no incoming edges, then $c \in C$ and $d = 1$;*
- *if $((c_1, d_1), r_1, (c, d)), \ldots, ((c_k, d_k), r_k, (c, d))$, $k \geq 1$ are the incoming edges of (c, d), then*
  - *$d = \bigoplus_{i=1}^{k}(d_i \otimes f(r_i))$ and $c_i \xrightarrow{\langle r_i \rangle}_{\mathcal{P}} c$ for all $1 \leq i \leq k$ (in a forwards $(\mathcal{W}, C)$-dag);*
  - *$d = \bigoplus_{i=1}^{k}(f(r_i) \otimes d_i)$ and $c \xrightarrow{\langle r_i \rangle}_{\mathcal{P}} c_i$ for all $1 \leq i \leq k$ (in a backwards $(\mathcal{W}, C)$-dag).*

We call a (forwards/backwards) $(\mathcal{W}, C)$-dag $\mathcal{D}$ a **witness dag** *for (c, d) if $\mathcal{D}$ is finite and (c, d) is the only vertex with no outgoing edges in $\mathcal{D}$.*

Notice that the extender operation $\otimes$ is used to calculate the value of a path. The value of a set of paths is computed using the combiner operation $\oplus$. The existence of a witness dag for (c, d) can be considered a proof that there exists a set of paths from C to c (or vice versa) whose combined value is d. Because of Defn. 1(5), it is always possible to identify a finite witness dag if such a set of paths exists.

3.1 Known Results

We briefly review some known results about (weighted) pushdown systems.

Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a pushdown system, and let C be a *regular* subset of $\mathit{Conf}(\mathcal{P})$. Then, according to [10], the sets $\mathit{pre}^*(C)$ and $\mathit{post}^*(C)$ are also regular and effectively computable (in the form of a finite automaton).

The results from [23, 24] show that the result can be extended to **generalized pushdown reachability (GPR) problems** on weighted pushdown systems:

Definition 5. *Let $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$ be a weighted pushdown system, where $\mathcal{P} = (P, \Gamma, \Delta)$, and let $C \subseteq P \times \Gamma^*$ be a regular set of configurations. The* **generalized pushdown predecessor (GPP) problem** *is to find for each $c \in \mathit{pre}^*(C)$:*

- *$\delta(c) \stackrel{\text{def}}{=} \bigoplus\{\, v(\sigma) \mid \sigma \in \mathit{path}(c, c'), c' \in C \,\}$;*
- *a backwards witness dag for $(c, \delta(c))$.*

The **generalized pushdown successor (GPS) problem** *is to find for each $c \in \mathit{post}^*(C)$:*

- *$\delta(c) \stackrel{\text{def}}{=} \bigoplus\{\, v(\sigma) \mid \sigma \in \mathit{path}(c', c), c' \in C \,\}$;*
- *a forwards witness dag for $(c, \delta(c))$.*

In [23, 24], the solutions for GPS and GPP are computed in the form of annotated finite automata. We describe the GPP case here; the GPS case is analogous, modulo certain details. Moreover, for the sake of keeping the presentation simple, we concentrate on the computation of the $\delta(c)$ values. A method for computing the witness dags is given in [23], and it is straightforward to transfer it to the distributed case.

Our input is a weighted pushdown system $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$, where $\mathcal{P} = (P, \Gamma, \Delta)$ and $\mathcal{S} = (D, \oplus, \otimes, 0, 1)$, together with a regular set of configurations C. The output is $\delta(c)$ for each $c \in pre^*(C)$. In general, there are infinitely many configurations in $pre^*(C)$ even if C itself is finite, so we can only hope to compute the solution symbolically. We use (annotated) finite automata for this purpose:

Definition 6. *A $\mathcal{P}$-automaton is a quintuple $\mathcal{A} = (Q, \Gamma, \eta, P, F)$ where $Q \supseteq P$ is a finite set of* **states**, *$\eta \subseteq Q \times \Gamma \times Q$ is the set of* **transitions**, *and $F \subseteq Q$ are the* **final states**. *The* **initial states** *of $\mathcal{A}$ are the control locations P. We say that a sequence of transitions $(p, \gamma_1, p_1), \ldots, (p_{n-1}, \gamma_n, q) \in \eta$* **reads** *configuration $\langle p, \gamma_1 \ldots \gamma_n \rangle$ if $p_1, \ldots, p_{n_1}, q$ are arbitrary states. The sequence is* **accepting** *iff q is a final state. If c is a configuration of $\mathcal{A}$, we denote by $acc_{\mathcal{A}}(c)$ the set of all accepting transition sequences in $\mathcal{A}$ for c; we say that c is accepted by $\mathcal{A}$ if $acc_{\mathcal{A}}(c)$ is non-empty.*

Note that a set of configurations of $\mathcal{P}$ is **regular** if and only if it is accepted by some $\mathcal{P}$-automaton. In what follows, $\mathcal{P}$ is fixed; hence, we usually omit the prefix $\mathcal{P}$ and speak simply of "automata".

A convenient property of regular sets of configurations is that they are closed under forwards and backwards reachability [6]. In other words, given an automaton $\mathcal{A}$ that accepts the set C, one can construct automata that accept the sets of all configurations that are forward or backwards reachable from C. Following [23, 24], two additional labelings for the transitions of $\mathcal{A}$ are computed to solve the GPP and GPS problems. The first, $l \colon \eta \to D$ assigns a weight from D to each automaton transition and allows to compute δ (see below). The second allows to compute the ω function. As mentioned earlier, we omit the second labeling for the sake of simplicity.

Without loss of generality, we assume henceforth that for every rule $\langle p, \gamma \rangle \hookrightarrow \langle p', w \rangle$ we have $|w| \leq 2$; this is not restrictive because every pushdown system can be simulated by another one that obeys this restriction and is larger by only a constant factor (e.g., [14]).

In the following, we first present an abstract version of the procedure given in [23, 24], which is designed for centralized computation. We then proceed to give an implementation for the distributed case.

Abstract algorithm Let $\mathcal{A} = (Q, \Gamma, \eta, P, F)$ be a $\mathcal{P}$-automaton that accepts a set of configurations C. Without loss of generality, we assume that $\mathcal{A}$ has no transition leading to an initial state.

Initially, we set $l(t) := 1$ for all $t \in \eta$. When we say that transition t should be updated with value d, we mean the following action: if t is not yet in η, add t to η and set $l(t) := d$; otherwise, update $l(t)$ to $l(t) \oplus d$.

For GPP, we add new transitions to $\mathcal{A}$ according to the following saturation rule:

> If $r := \langle p, \gamma \rangle \hookrightarrow \langle p', w \rangle$ is a rule, $t_1 \ldots t_{|w|}$ a sequence that reads $\langle p, w \rangle$ and ends in state q, then let d be $l(t_1) \otimes \ldots \otimes l(t_{|w|})$ and update (p, γ, q) with the value $f(r) \otimes d$.

The procedure terminates when the saturation rule can no longer be applied (i.e., a fixed point has been reached).

Concrete algorithm A concrete implementation is given in [23] and reproduced in Figure 1. Each iteration of the loop starting at line 14 executes one or more applications of the saturation rule. After the computation has finished, the resulting automaton accepts all configurations $c \in pre^*(C)$. Then, we have $\delta(c) = \bigoplus_{t_1 \cdots t_n \in acc_{\mathcal{A}'}(c)} l(t_1) \otimes \cdots \otimes l(t_n)$.

In [23] the time complexity of the GPP algorithm from Figure 1 was stated as $\mathcal{O}(|Q|^2 \cdot |\Delta| \cdot \ell)$, where ℓ is the length of the longest descending chain in $\mathcal{S}$, and the space complexity (determined by the number of transitions in the final automaton) as $\mathcal{O}(|Q| \cdot |\Delta| + |\eta|)$.

Algorithm 1

Input: a weighted pushdown system $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$, where $\mathcal{P} = (P, \Gamma, \Delta)$ and $\mathcal{S} = (D, \oplus, \otimes, 0, 1)$, and an automaton $\mathcal{A} = (Q, \Gamma, \eta_0, P, F)$ that accepts C, such that $\mathcal{A}$ has no transitions into states from P.

Output: an automaton $\mathcal{A}' = (Q, \Gamma, \eta, P, F)$ that accepts $pre^*(C)$, with annotation function $l \colon \eta \to D$

1 **procedure** $update(t, v)$
2 **begin**
3 $\quad \eta := \eta \cup \{t\}$
4 $\quad newValue := l(t) \oplus v$
5 $\quad$ **if** $newValue \neq l(t)$ **then**
6 $\quad\quad workset := workset \cup \{t\}$
7 $\quad\quad l(t) := newValue$
8 **end**
9
10 $\eta := \eta_0;\ workset := \eta_0;\ l := \lambda t.0$
11 **for all** $t \in \eta_0$ **do** $l(t) := 1$
12 **for all** $r = \langle p, \gamma \rangle \hookrightarrow \langle p', \varepsilon \rangle \in \Delta$ **do**
13 $\quad update((p, \gamma, p'), f(r))$
14 **while** $workset \neq \emptyset$ **do**
15 $\quad$ remove some transition $t = (q, \gamma, q')$ from $workset$;
16 $\quad$ **for all** $r = \langle p_1, \gamma_1 \rangle \hookrightarrow \langle q, \gamma \rangle \in \Delta$ **do**
17 $\quad\quad update((p_1, \gamma_1, q'), f(r) \otimes l(t))$
18 $\quad$ **for all** $r = \langle p_1, \gamma_1 \rangle \hookrightarrow \langle q, \gamma\gamma_2 \rangle \in \Delta$ **do**
19 $\quad\quad$ **for all** $t' = (q', \gamma_2, q'') \in \eta$ **do**
20 $\quad\quad\quad update((p_1, \gamma_1, q''), f(r) \otimes l(t) \otimes l(t'))$
21 $\quad$ **for all** $r = \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p', \gamma_2\gamma \rangle \in \Delta$ **do**
22 $\quad\quad$ **if** $t' = (p', \gamma_2, q) \in \eta$ **then**
23 $\quad\quad\quad update((p_1, \gamma_1, q'), f(r) \otimes l(t') \otimes l(t))$
24 **return** $((Q, \Gamma, \eta, P, F), l)$

Fig. 1. An algorithm for creating a weighted automaton for the GPP problem.

3.2 A Distributed Algorithm

We now discuss how the computation can be distributed when the rules in Δ are distributed over a set *Sites* of servers. As in Section 3.1, we discuss both the GPP and the GPS case, and give a concrete implementation for GPP, as the one for GPS is very similar.

We fix a weighted pushdown system $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$, where $\mathcal{P} = (P, \Gamma, \Delta)$ and $\mathcal{S} = (D, \oplus, \otimes, 0, 1)$, and a regular set C of configurations. The solution we discuss here distributes the workload among the servers according to control locations, i.e., for every control location there is a server that is 'responsible' for it. More precisely, we make the following assumptions:

1. There exists a mapping $f_S : P \to \mathit{Sites}$ that assigns control locations to sites.
2. Every rule $\langle p, \gamma \rangle \hookrightarrow \langle p', w \rangle$ is stored at the site $f_S(p)$ (for the GPS problem), or at $f_S(p')$ (for the GPP problem).

Stating assumption 2 differently, we are working with a collection $(\mathcal{W}_s)_{s \in \mathit{Sites}}$ of weighted pushdown systems that differ only in their rules, i.e., $\mathcal{W}_s = (\mathcal{P}_s, \mathcal{S}, f_{|\Delta_s})$ and $\mathcal{P}_s = (P, \Gamma, \Delta_s)$, where the set Δ_s satisfies assumption 2.

We say that a rule $\langle p, \gamma \rangle \hookrightarrow \langle p', w \rangle$ is a **boundary rule** if p and p' are assigned to different sites. If such a boundary rule exists, we call the sites responsible for p and p' **neighboring sites**.

Definition 7. *Let $\mathcal{D} = (V, E)$ be a $(\mathcal{W}, C)$-dag and $s \in \mathit{Sites}$. An edge (v, r, v') of $\mathcal{D}$, where $v = (\langle p, w \rangle, d)$, is called a* **boundary edge** *if r is a boundary rule. Moreover, v' is called a* **boundary node** *of the site $f_S(p)$. We denote by $\mathcal{T}(s) = \{\, \langle p, w \rangle \mid f_S(p) = s,\ w \in \Gamma^* \,\}$ the configurations that begin with the control locations for which site s is responsible. Moreover, the* **s-region of $\mathcal{D}$** *is the subgraph (V_s, E_s) of $\mathcal{D}$, where $V_s = \{\, (c, d) \in V \mid c \in \mathcal{T}(s) \,\}$ and $E_s = \{\, (v, r, v') \in E \mid v \in V_s \,\}$.*

Informally, the s-region contains the subgraph of $\mathcal{D}$ induced by V_s, i.e., the nodes for whose configurations s is responsible, plus the "fringe" of this subgraph, i.e., the boundary edges originating in V_s and their target nodes.

Abstract algorithm. We can now give an abstract description of the GPP and GPS algorithms. Given $\mathcal{W}$ and C, every site s computes the set $\mathcal{T}^C_{pre}(s) \stackrel{\text{def}}{=} (\mathit{pre} \cup \mathit{id})(\mathit{pre}^*(C) \cap \mathcal{T}(s))$ (in the GPP case) or $\mathcal{T}^C_{post}(s) \stackrel{\text{def}}{=} (\mathit{post} \cup \mathit{id})(\mathit{post}^*(C) \cap \mathcal{T}(s))$ (in the GPS case). In the following, we write $\bar{\mathcal{T}}(s)$ to mean $\mathcal{T}^C_{pre}(s)$ or $\mathcal{T}^C_{post}(s)$, depending on the context.

Intuitively speaking, every site s computes a partition of $\mathit{pre}^*(C)$ or $\mathit{post}^*(C)$, namely, the set of configurations that have control locations for which s is responsible, extended with the configurations reached by boundary rules. Note that the set $\bar{\mathcal{T}}(s)$ contains all the configurations that can be generated using rules stored at s.

The idea is that site s becomes involved in a GPP/GPS computation if it is discovered that $\bar{\mathcal{T}}(s) \neq \emptyset$. Initially, each site s starts with the set $C \cap \mathcal{T}(s)$. If a boundary rule causes a site s to discover configurations that belongs to $\mathcal{T}(s')$ (for some site $s' \neq s$), then s will send those configurations to s', and s' continues its GPP/GPS computation using those configuration.

Concrete algorithm. At a more concrete level of description, every site s computes an automaton $\mathcal{A}_s$ that accepts $\bar{\mathcal{T}}(s)$, and appropriate labeling functions for δ and for the witness dags. Basically, the distributed algorithm is a straightforward extension of the non-distributed case: every site s runs a GPP/GPS algorithm similar to the one in Figure 1 with $\mathcal{W}_s$. The main complication is that some parts of the automata need to be shared between sites.

To be more precise, let $\mathcal{A}$ be an automaton that accepts C. Initially, $\mathcal{A}_s$ is an automaton that accepts $C \cap \mathcal{T}(s)$, which can be constructed by merely taking the states and transitions of $\mathcal{A}$ that are reachable from initial states p such that $f_S(p) = s$.

Each site s then carries out the algorithm from Figure 1 using $\mathcal{W}_s$. If s and s' are neighboring sites, then, at some stage of the computation at s, the automaton $\mathcal{A}_s$ may accept configurations from $\mathcal{T}(s') \cap \bar{\mathcal{T}}(s)$, i.e., configurations that ought to be maintained by s'. Let $T_{s,s'}$ be the set of transitions in $\bigcup_{c \in \mathcal{T}(s')} acc_{\mathcal{A}_s}(c)$, i.e., the transitions in $\mathcal{A}_s$ that form part of an accepting path for such configurations. Whenever s detects a transition t that belongs to $T_{s,s'}$ (or an update in such a transition), then s keeps t in its automaton, but also sends it to s'. Thus, every site s ends up with an automaton that accepts $\bar{\mathcal{T}}(s)$.

Along with the configurations, every site also computes information to construct the δ function and witness dags. Notice that the vertices in an s-region of a $(\mathcal{W}, C)$-dag $\mathcal{D}$ are labeled with configurations from $\bar{\mathcal{T}}(s)$, and that the edges of the region are labeled with the rules stored at s. Thus, s has all the information needed to construct the s-region of $\mathcal{D}$. More precisely, the information needed to construct an s-region can be generated by an annotation of the automaton maintained by s, in the same way as in [23].

The δ function is computed in the form of another annotation that labels automaton transitions with semiring values. When sending a transition from one site to another, the semiring values are also sent. For a configuration $c = \langle p, w \rangle$, the value of $\delta(c)$ can be obtained by evaluating the automaton $A_{f_S(p)}$, as shown in Section 3.1.

Figure 2 shows the changes that must be made to Algorithm 1 to implement this approach. The figure shows the algorithm from the point of view of site s. The algorithm maintains a mapping $sites\colon Q \to 2^{Sites}$. If $s' \in sites(q)$, then the current automaton contains a path that leads from an initial state p, where $f_S(p) = s'$, to the state q. This means that all transitions of the form (q, y, q') are part of accepting paths for configurations from $\mathcal{T}(s')$. As a consequence, whenever such a transition is first generated or updated, it needs to be sent to s', and q' must be added to $sites(s')$.

The changes to Algorithm 1 consist of three parts:

- The procedure *update* is replaced by a new version;
- there is an additional procedure *add_recursive*;
- a couple of lines are added to the beginning of the main procedure.

The new lines in the main procedure initialize the *sites* function. The *update* function is extended by lines 8–11. These lines send the updated transition to other sites as required. Sending a transition t with value v to site s' is represented by $update_{s'}(t, v)$, which can be thought of as a remote procedure call (of the function *update*) on site s' that adds t to the worklist of s'. Finally, the target state of t must be added to $sites(s')$. This is done by procedure *add_recursive*, which also takes care of sending additional transitions to s', if required.

Algorithm 2 (running on site s)

Input: a weighted pushdown system $\mathcal{W}_s = (\mathcal{P}_s, \mathcal{S}, f_{|\Delta_s|})$, where $\mathcal{P}_s = (P, \Gamma, \Delta_s)$, and $\mathcal{S} = (D, \oplus, \otimes, 0, 1)$, and an automaton $\mathcal{A}_s = (Q, \Gamma, \eta_0, P, F)$ that accepts $C \cap \mathcal{T}(s)$, such that $\mathcal{A}$ has no transitions into states from P.

Output: an automaton $\mathcal{A}'_s = (Q, \Gamma, \eta, P, F)$ that accepts $\mathcal{T}^C_{pre}(s)$)' with annotation function $l \colon \eta \to D$

Replacement for *update* procedure:

procedure *update*(t, v)
begin
  $\eta := \eta \cup \{t\}$
  $newValue := l(t) \oplus v$
  if $newValue \neq l(t)$ **then**
    $workset := workset \cup \{t\}$
    $l(t) := newValue$
    // assume $t = (p, \gamma, q)$
    for all $s' \in sites(p)$ **do**
      $update_{s'}(t, l(t))$;
      add_recursive(q, s');
end

New procedure *add_recursive*:

procedure *add_recursive*(q, s')
begin
  if $s' \in sites(q)$ **then return**;
  $sites(q) := sites(q) \cup \{s'\}$;
  for all $t' = (q, \gamma', q') \in \eta$ **do**
    $update_{s'}(t', l(t'))$;
    add_recursive(q', s');
end

Additions to main procedure:

$sites := \lambda p.\emptyset$;
for all $r = \langle p, \gamma \rangle \hookrightarrow \langle p', w \rangle \in \Delta$ **do**
  if $f_S(p) \neq s$ **then**
    $sites(p) := sites(p) \cup \{f_S(p)\}$

Fig. 2. Modification of Algorithm 1 for distributed GPP.

Complexity Let us state the complexity of Algorithm 1 when run on site s. The main procedure is unchanged and runs in $\mathcal{O}(|Q|^2 \cdot |\Delta_s| \cdot \ell)$ time, where ℓ is the longest descending chain in $\mathcal{S}$. Additional work is required for sending and receiving transitions to/from neighboring sites. Suppose that s has n neighboring sites, and that these sites send t transitions to s. For every send or receive action, s needs to perform some constant amount of work.

Note that t is bounded by $\mathcal{O}(|Q| \cdot |\Delta|)$, and that every transition can be received at most ℓ times, so the effort for received transitions is at most $\mathcal{O}(|Q| \cdot |\Delta| \cdot \ell)$, although in practice we expect it to be much lower.

In the worst case, s must send all of its transitions to all n neighbors at most ℓ times, i.e., $\mathcal{O}(|Q| \cdot |\Delta| \cdot n \cdot \ell)$. Again, we expect his number to be much lower in practice.

4 Background on SPKI/SDSI

In SPKI/SDSI, all *principals* are represented by their public keys, i.e., the principal *is* its public key. A principal can be an individual, process, host, or any other entity. $\mathcal{K}$ denotes the set of public keys. Specific keys are denoted by K, K_A, K_B, K', etc. An *identifier* is a word over some alphabet Σ. The set of identifiers is denoted by $\mathcal{A}$. Identifiers will be written in typewriter font, e.g., `A` and `Bob`. A *term* is a key followed by zero or more identifiers. Terms are either keys, local names, or extended names. A *local name* is of the form K `A`, where $K \in \mathcal{K}$ and `A` $\in \mathcal{A}$. For example, K `Bob` is a local name. Local names are important in SPKI/SDSI because they create a decentralized name space. The local name space of K is the set of local names of the form K $\mathtt{A}$. An *extended name* is of the form K σ, where $K \in \mathcal{K}$ and σ is a sequence of identifiers of length greater than one. For example, K `UW` `CS` `faculty` is an extended name.

4.1 Certificates

SPKI/SDSI has two types of certificates, or "certs":

Name Certificates (or *name certs*): A name cert provides a definition of a local name in the issuer's local name space. Only key K may issue or sign a cert that defines a name in its local name space. A name cert C is a signed four-tuple $(K, \mathtt{A}, S, V)$. The issuer K is a public key and the certificate is signed by K. $\mathtt{A}$ is an identifier. The subject S is a term. Intuitively, S gives additional meaning for the local name K $\mathtt{A}$. V is the *validity specification* of the certificate. Usually, V takes the form of an interval $[t_1, t_2]$, i.e., the cert is valid from time t_1 to t_2 inclusive.

Authorization Certificates (or *auth certs*): An auth cert grants or delegates a specific authorization from an issuer to a subject. Specifically, an auth cert c is a five-tuple (K, S, D, T, V). The *issuer* K is a public key, which is also used to sign the cert. The *subject* S is a term. If the *delegation bit* D is turned on, then a subject receiving this authorization can delegate this authorization to other keys. The *authorization specification* T specifies the permission being granted; for example, it may specify a permission to read a specific file, or a permission to login to a particular host. The *validity specification* V for an auth cert is the same as in the case of a name cert.

A *labeled rewrite rule* is a pair $(L \longrightarrow R, T)$, where the first component is a rewrite rule and the second component T is an authorization specification. For notational convenience, we will write the labeled rewrite rule $(L \longrightarrow R, T)$ as $L \xrightarrow{T} R$. We will treat certs as labeled rewrite rules:[3]

- A name cert $(K, \mathtt{A}, S, V)$ will be written as a labeled rewrite rule K $\mathtt{A} \xrightarrow{\top} S$, where $\top$ is the authorization specification such that for all other authorization specifications t, $\top \cap t = t$, and $\top \cup t = \top$. [4] Sometimes we will write $\xrightarrow{\top}$ as simply $\longrightarrow$, i.e., a rewrite rule of the form $L \longrightarrow R$ has an implicit label of $\top$.
- An auth cert (K, S, D, T, V) will be written as $K \,\square \xrightarrow{T} S \,\square$ if the delegation bit D is turned on; otherwise, it will be written as $K \,\square \xrightarrow{T} S \,\blacksquare$.

4.2 Authorization

Because we only use labeled rewrite rules in this paper, we refer to them as rewrite rules or simply rules. A term S appearing in a rule can be viewed as a string over the alphabet $\mathcal{K} \cup \mathcal{A}$, in which elements of $\mathcal{K}$ appear only in the beginning. For uniformity, we also refer to strings of the form $S \,\square$ and $S \,\blacksquare$ as terms. Assume that we are given a labeled rewrite rule $L \xrightarrow{T} R$ that corresponds

[3] In authorization problems, we only consider valid certificates, so the validity specification V for a certificate is not included in its rule.

[4] The issue of intersection and union of authorization specifications is discussed in detail in [9, 13].

to a cert. Consider a term $S = LX$. In this case, the labeled rewrite rule $L \xrightarrow{T} R$ applied to the term S (denoted by $(L \xrightarrow{T} R)(S)$) yields the term RX. Therefore, a rule can be viewed as a function from terms to terms that rewrites the left prefix of its argument, for example,

$$(K_A\ \texttt{Bob} \longrightarrow K_B)(K_A\ \texttt{Bob}\ \texttt{myFriends}) = \ K_B\ \texttt{myFriends}$$

Consider two rules $c_1 = (L_1 \xrightarrow{T} R_1)$ and $c_2 = (L_2 \xrightarrow{T'} R_2)$, and, in addition, assume that L_2 is a prefix of R_1, i.e., there exists an X such that $R_1 = L_2X$. Then the *composition* $c_2 \circ c_1$ is the rule $L_1 \xrightarrow{T \cap T'} R_2X$. For example, consider the two rules:

$$\begin{aligned} c_1 :&\ \ K_A\ \texttt{friends} \xrightarrow{T} K_A\ \texttt{Bob}\ \texttt{myFriends} \\ c_2 :&\ \ K_A\ \texttt{Bob} \xrightarrow{T'} K_B \end{aligned}$$

The composition $c_2 \circ c_1$ is $K_A\ \texttt{friends} \xrightarrow{T \cap T'} K_B\ \texttt{myFriends}$. Two rules c_1 and c_2 are called *compatible* if their composition $c_2 \circ c_1$ is well defined.[5]

4.3 The Authorization Problem in SPKI/SDSI

Assume that we are given a set of certs $\mathcal{C}$ and that principal K wants access specified by authorization specification T. The authorization question is: "Can K be granted access to the resource specified by T?"

A *certificate chain* $ch = (c_k \circ c_{k-1} \circ \cdots \circ c_1)$ is a sequence such that for $\mathcal{C}$, where $c_1, c_2, \cdots, c_k$ are certificates in $\mathcal{C}$, certificate chain ch defines the transformation $c_k \circ c_{k-1} \circ \cdots \circ c_1$. The label of ch, denoted by $L(ch)$, is the label of $c_k \circ c_{k-1} \circ \cdots \circ c_1$. We assume that the authorization specification T is associated with a unique principal K_r (which could be viewed as the owner of the resource r to which T refers). Given a set of certificates $\mathcal{C}$, an authorization specification T, and a principal K, a *certificate-chain-discovery* algorithm looks for a finite set of certificate chains that "prove" that principal K is allowed to make the access specified by T.

Formally, certificate-chain discovery attempts to find a finite set $\{ch_1, \cdots, ch_m\}$ of certificate chains such that for all $1 \leq i \leq m$

$$ch_i(K_r\ \square) \in \{K\ \square, K\ \blacksquare\}\ .$$

and $T \subseteq \bigcup_{i=1}^{m} L(ch_i)$.

Clarke et al. [8] presented an algorithm for certificate-chain discovery in SPKI/SDSI with time complexity $O(n_K^2 |\mathcal{C}|)$, where n_K is the number of keys and $|\mathcal{C}|$ is the sum of the lengths of the right-hand sides of all rules in $\mathcal{C}$. However, this algorithm only solved a restricted version of certificate-chain discovery:

[5] In general, the composition operator $\circ$ is not associative. For example, c_3 can be compatible with $c_2 \circ c_1$, but c_3 might not be compatible with c_2. Therefore, $c_3 \circ (c_2 \circ c_1)$ can exist when $(c_3 \circ c_2) \circ c_1$ does not exist. However, when $(c_3 \circ c_2) \circ c_1$ exists, so does $c_3 \circ (c_2 \circ c_1)$; moreover, the expressions are equal when both are defined. Thus, we allow ourselves to omit parentheses and assume that $\circ$ is right associative.

a solution could only consist of a *single* certificate chain. For instance, consider the following certificate set:

$$c_1: \quad (K,\ K_A,\ 0,\ \texttt{((dir /etc) read)},\ [t_1, t_2])$$
$$c_2: \quad (K,\ K_A,\ 0,\ \texttt{((dir /etc) write)},\ [t_1, t_2])$$

Suppose that Alice makes the request

$$(K_A, \texttt{((dir /etc) (* set read write))}).$$

In this case, the chain "(c_1)" authorizes Alice to read from directory `/etc`, and a separate chain "(c_2)" authorizes her to write to `/etc`. Together, (c_1) and (c_2) prove that she has both read and write privileges for `/etc`. However, both of the certificates c_1 and c_2 would be removed from the certificate set prior to running the certificate-chain discovery algorithm of Clarke et al., because `read` $\not\supseteq$ `(* set read write)` and `write` $\not\supseteq$ `(* set read write)`. Consequently, no proof of authorization for Alice's request would be found. Schwoon et al. [24] presented algorithms for the full certificate-chain-discovery problem, based on solving reachability problems in weighted pushdown systems. Their formalization allows a proof of authorization to consist of a set of certificate chains. This paper uses the WPDS-based algorithm for certificate-chain-discovery introduced in [24].

5 Weighted Pushdown Systems and SPKI/SDSI

In the section, we show that WPDSs are a useful tool for solving problems related to certificate-chain discovery in SPKI/SDSI. The following definitions are largely taken from [23].

The following correspondence between SPKI/SDSI and pushdown systems was presented in [24]: let $\mathcal{C}$ be a (finite) set of certificates such that $\mathcal{K}_\mathcal{C}$ and $\mathcal{I}_\mathcal{C}$ are the keys and identifiers, respectively, that appear in $\mathcal{C}$. Moreover, let $\mathcal{T}$ be the set from which the authorization specifications in $\mathcal{C}$ are drawn. Then $\mathcal{S}_\mathcal{C} = (\mathcal{T}, \cup, \cap, \perp, \top)$, where $\cap, \cup$ are the intersection and union of auth specs as discussed in [9, 13], forms a semiring with domain $\mathcal{T}$. We now associate with $\mathcal{C}$ the weighted pushdown system $\mathcal{W}_\mathcal{C} = (\mathcal{P}_C, \mathcal{S}_\mathcal{C}, f)$, where $\mathcal{P}_\mathcal{C} = (\mathcal{K}_\mathcal{C}, \mathcal{I}_\mathcal{C} \cup \{\square, \blacksquare\}, \Delta_\mathcal{C})$, i.e., the keys of $\mathcal{C}$ are the control locations; the identifiers form the stack alphabet; the rule set $\Delta_\mathcal{C}$ is defined as the set of labeled rewrite rules derived from the name and auth certs as shown in Section 4.1; and f maps every rule to its corresponding authorization specification.

The usefulness of this correspondence stems from the following simple observation: A configuration $\langle K, \sigma \rangle$ of $\mathcal{P}_\mathcal{C}$ can reach another configuration $\langle K', \sigma' \rangle$ if and only if $\mathcal{C}$ contains a chain of certificates $(c_1, \ldots, c_k)$ such that $(c_k \circ \cdots \circ c_1)(K\ \sigma) = K'\ \sigma'$. Moreover, the label of the certificate chain is precisely $v(c_1 \cdots c_k)$. Thus, solving the GPP/GPS problem provides a way to find a set of certificate chains to prove that a certain principal K' is allowed to access a resource of principal K. Moreover, the solution of the problem identifies a set of certificate chains such that the union of their labels is maximal (with respect to the semiring ordering $\sqsubseteq$).

In the authorization problem, we are given a set of certs $\mathcal{C}$, a principal K, and resource K_r. In the PDS context, K can access the resource with authorization specification T iff the following statement is true: In the GPP problem for $\mathcal{W}_\mathcal{C}$ and $C = \{\langle K, \square\rangle, \langle K, \blacksquare\rangle\}$, it holds that $\delta(\langle K_r, \square\rangle) \sqsubseteq T$; equivalently, in the GPS problem for $\mathcal{W}_\mathcal{C}$ and $C = \{\langle K_r, \square\rangle\}$ we have $\delta(\langle K, \square\rangle) \oplus \delta(\langle K, \blacksquare\rangle) \sqsubseteq T$.

6 Distributed Certificate-Chain Discovery

The algorithms for GPR problems proposed in [23, 24] work under the assumption that all pushdown rules (or certificates, resp.) are stored centrally at the site that carries out the computation. In a real-world setting, certificates may be issued by many principals, and centralized storage at one site may not be desirable or possible. We therefore propose versions of these algorithms that solve the problems in a distributed environment.

Let $\mathcal{C}$ be a (finite) set of certificates and $\mathcal{W}_\mathcal{C} = (\mathcal{P}_C, \mathcal{S}_\mathcal{C}, f)$ be the WPDS associated with $\mathcal{C}$ (see Section 5 for details). As in Section 3.2, we assume that the rules/certificates in Δ are distributed over a set of servers, where the f_S function describes the distribution of principals over the sites, and also assume that every certificate/rule is stored at the site responsible for its issuer or subject. In the remainder of this section, we consider distributed solutions for the following distributed certificate-chain-discovery problem, under the aforementioned assumptions:

> Given a principal r (the **resource**) and a principal c (the **client**) with public keys K_r and K_c, is there a set of certificate chains in $\mathcal{W}$ that allows c to access r and, if there is, what is their combined value?

The problem is equivalent to either of the following problems in the WPDS setting:

- As a GPP problem: For $C = \{\langle K_c, \square\rangle, \langle K_c, \blacksquare\rangle\}$ and $c = \langle K_r, \square\rangle$, compute $\delta(c)$ and a backwards witness dag for $(c, \delta(c))$.
- As a GPS problem: For $C = \{\langle K_r, \square\rangle\}$, $c_1 = \langle K_c, \square\rangle$, and $c_2 = \langle K_c, \blacksquare\rangle$, compute $\delta(c_1) \oplus \delta(c_2)$ and forwards witness dags for $(c_1, \delta(c_1))$ and $(c_2, \delta(c_2))$.

Sections 6.1 and 6.2 propose protocols for the communication between the client, the resource, and the servers that co-operate to solve the distributed access problem. We propose two protocols, one based on the GPP formulation of the above problem, the other on the GPS formulation. The protocols assume algorithms for solving GPP and GPS in the distributed setting, and which are provided in Section 3. The relative merits of the protocols, as well as security and privacy-related issues, are discussed in Section 6.3.

6.1 The GPS Protocol for Distributed Certificate-Chain Discovery

In a distributed setting, multiple access requests may happen at the same time. We shall use unique *request ids* to distinguish them. In the GPS variant, the protocol consists of three phases.

Initialization: The initialization consists of the following steps:

1. The client c sends a message to the resource r requesting access. The message contains the public key of the client, K_c.

2. The resource r responds by sending a unique request identifier *reqid*, which will distinguish this request from other requests that may currently be in progress.
3. The client sends a message to the site $f_S(K_c)$ (called the *client site* and denoted s_c from here on). The message contains (i) its key K_c, (ii) the request id *reqid*, (iii) the so-called *client certificate*: the request id signed by the client.
4. The client site checks whether the contents and signature of the client certificate match expectations. If the check is successful, the client site tells the client that certificate discovery may begin.
5. The client asks the resource to initiate the search.
6. The resource sends a message to the site $f_S(K_r)$ (called the *resource site* and written s_r) containing its public key K_r, the request id *reqid*, and a request to initiate certificate discovery.

Search: The resource site initiates a GPS query for the singleton set $C = \{\langle K_r, \Box\rangle\}$, where *reqid* is used to distinguish this query from others (so that servers may work on multiple requests at the same time). The query is resolved by all the servers together, and the details of the search algorithm are given in Section 3. Here, the crucial points are that s_r starts a local GPS computation, and if it notices that $post^*(C)$ intersects $\mathcal{T}(s)$ for some other site s (because of some boundary certificate), then s is asked to participate in the search. Site s may, in the course of its computation, contact other sites. Each site s constructs the set $\mathcal{T}^C_{post}(s)$ and maintains information that allows to construct the s-region of the required witness dags.

Verification: Because of its earlier communication with the client, the client site s_c knows that $c_1 := \langle K_c, \Box\rangle$ and $c_2 := \langle K_c, \blacksquare\rangle$ are the targets of the search. Moreover, because $c_1, c_2 \in \mathcal{T}(s_c)$, the client site knows whether the finished search has reached c_1, c_2. To complete the algorithm, the result must be reported to the resource. Thus, in the verification phase, the direction of the flow of information is contrary to the search phase.

The client site starts by constructing the s_c-region of the witness dags. It then sends this sub-dag starting at its boundary nodes 'upstream' to the corresponding neighboring sites. The neighboring sites use this information to complete their own sub-dags and send them further upstream until s_r has the full witness dags for c_1 and c_2. The result is then reported by s_r to the resource. Moreover, all communications in this phase are accompanied by the *client certificate* mentioned earlier.

The resource verifies the result, i.e., checks the integrity of the dag, the signatures on all certificates used in the dags, whether the client certificate matches *reqid*, and whether its signature matches the client. Depending on the outcome, access is allowed or denied to the client.

The verification of the complete dag may place a great workload on the resource. An alternative is as follows: Instead of sending complete sub-dags, the sites only report the sum (w.r.t. $\oplus$) of the paths inside the dags. Then, the result given by s_r to the resource consists of certificates issued by the resource and the combined values of the paths below them. This also reduces the amount of network traffic.

6.2 The GPP Protocol for Distributed Certificate-Chain Discovery

In this setting, the search is started at the client site, and, in comparison with Section 6.1, the flow of information between the sites is reversed.

Initialization:

1. The client c sends a message to the resource r requesting access.
2. The resource generates $reqid$ and sends the pair $(R, reqid)$ to the resource site s_r (to notify it of an 'incoming' search). After s_r has acknowledged receipt of the message, the resource sends $reqid$ to the client.
3. The client contacts the client site s_c and asks it to initiate a GPP computation. Along with the request, it sends $reqid$ and the client certificate as in Section 6.1.
4. The client site again checks correctness of the client certificate. If correct, s_c begins the search.

Search: The search stage is analogous to the GPS protocol, except that it is started at the client site and from the set $C = \{\langle K_c, \blacksquare\rangle, \langle K_c, \square\rangle\}$. In brief, a site s becomes involved in the search if $pre^*(C)$ intersects $\mathcal{T}(s)$. Communications between sites are tagged with *both reqid* and the client certificate.

Verification: At the end of the search, the resource site (which knows that the search with id $reqid$ has the target $c = \langle K_r, \square\rangle$) can determine whether c was reachable from C and what the value of $\delta(c)$ is.

To generate a complete witness dag, s_r can request from the sites further 'downstream' their regions of the witness dag, and then pass the complete dag along with the client certificate to the resource, which will verify it and (if successful) grant access to the client.

As an alternative solution, s_r may report to the resource just the certificates issued by the resource and the combined values of the paths above them. In that case, no further communication between the sites is necessary.

Example 1. Consider the rules shown below:

$$
\begin{aligned}
r_1 &:= \langle K_r, \square\rangle \hookrightarrow \langle K_{uw}, \texttt{faculty}\,\blacksquare\rangle \\
r_2 &:= \langle K_{uw}, \texttt{faculty}\rangle \hookrightarrow \langle K_{ls}, \texttt{faculty}\rangle \\
r_3 &:= \langle K_{ls}, \texttt{faculty}\rangle \hookrightarrow \langle K_{cs}, \texttt{faculty}\rangle \\
r_4 &:= \langle K_{ls}, \texttt{faculty}\rangle \hookrightarrow \langle K_{bio}, \texttt{faculty}\rangle \\
r_5 &:= \langle K_{cs}, \texttt{faculty}\rangle \hookrightarrow \langle K_{Bob}, \varepsilon\rangle
\end{aligned}
$$

with $f(r_1) := t$ and $f(r_i) := \top$ for $2 \leq i \leq 5$. We assume that there are four sites, *UW*, *LS*, *CS*, and *BIO*. The sitemap f_S is as follows: $f_S(K_r)$ and $f_S(K_{uw})$ are equal to *UW*, $f_S(K_{ls})$ is equal to *LS*, $f_S(K_{bio})$ is equal to *BIO*, and $f_S(K_{cs})$ and $f_S(K_{Bob})$ are equal to *CS*. This example is used as Case 1 in Section 7.1. Suppose that Bob (at site CS) wants to access resource R (at site UW). Then, the site CS starts the search with $C = \{\langle K_{Bob}, \square\rangle, \langle K_{Bob}, \blacksquare\rangle\}$ and discovers, through r_5 and r_3, that $pre^*(C)$ intersects $\mathcal{T}(LS)$, so site LS gets involved and notices that (because of r_2), site UW must also take part in the search. The automata computed by CS, LS, and UW are shown in Figure 3; notice that site Bio does not become involved. At the end of the computation, site UW sees that $\langle K_r, \square \rangle$ is accepted by its automaton $\mathcal{A}_{UW}$ with weight t, and that is the result reported to resource R.

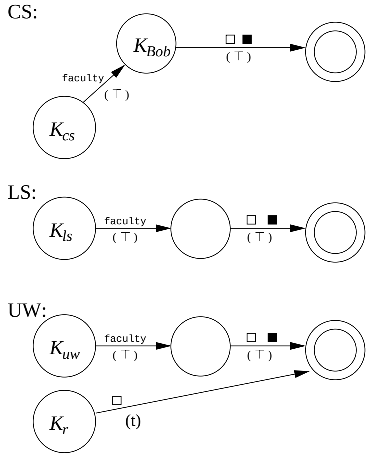

Fig. 3. pre^* automata for $\langle R, \square \rangle$ computed at sites CS, LS, and UW; weights on transitions shown in parentheses.

6.3 Discussion

Here, we discuss privacy and security-related topics, compare the two protocols, and discuss possible improvements.

Privacy: During the search, the parties involved learn the following:

– Only the resource and the client know that the client has asked to access the resource.
– The resource site knows that a request has been made to the resource, but not by whom.
– The client site knows only that the client has made a request, but not to whom.
– All other sites know only that a request has been made, but not by whom or to whom. They may surmise something about the nature of the request judging from the identifiers on the transitions, the direction from which the query comes, and the direction from where a confirmation comes, but they can only observe the communication with their neighbor sites.

Thus, the privacy of the access request is ensured during the search. However, when the witness dag is constructed during the construction phase, all sites learn the identity of the client. This can be avoided if the alternative method is used, in which only the values of certain paths in the dag are transmitted among sites. This alternative solution also prevents the unnecessary spread of certificates among sites (which might contain sensitive information).

Security against attacks

Spoofing and eavesdropping. We assume that all parties involved in the search can communicate securely and that no identification spoofing can take place.

Trusting the sites. Because the main part of the computation is carried out by the sites, the protocols are potentially susceptible to malicious behavior of the sites. A malicious site could either invent or ignore certificates. Ignoring certificates would only be to the detriment of the users for which the site is responsible, and seems unlikely to be a cause for concern.

Inventing certificates is also not a problem if the verification stage constructs the full witness dag because in this case all certificates (which are signed by their issuers) have to be supplied. The alternative solution, in which only values are reported, is more problematic: in essence, reporting the value of the paths in a sub-dag rooted at a node $(\langle K, w\rangle, d)$ amounts to issuing a confirmation (in the name of principal K) that there is a certificate chain from $\langle K, w\rangle$ to the client. Therefore, the alternative solution requires K to trust the site to use K's certificates truthfully. Note that if all boundary certificates have subjects that are under direct control of the respective site operator, this is not a problem.

The client certificate. The resource must verify that the reported result is indeed valid for the client who has initiated the request. If the verification stage constructs full witness dags, this becomes straightforward: the maximal nodes of the dags must refer to the client.

If the alternative solution is used in the verification, the client certificate serves this purpose, provided that both resource and client site verify its correctness.

A comparison of the two protocols In the GPP-based protocol, the search starts at the client site; in the GPS-based protocol it starts at the resource site. If a site is responsible for a 'popular' resource, the GPS-based protocol may put too much workload on it. Moreover, denial-of-service attacks are conceivable in which a malicious client causes a large number of GPS computations (under different identities) that are doomed to fail. In the GPP-based protocol, this is less likely to happen: the workload would fall mostly on the client site, which can be assumed to have a relationship to the client (e.g., the site is the client's company, ISP, etc.), and thus there is some 'social safeguard' against denial-of-service attacks.

Moreover, when the construction of complete witness dags is omitted, the GPP-based solution does not require a separate verification stage. For these reasons, it seems that the GPP-based solution has some advantages over the GPS-based solution. However, we have yet to carry out a more precise investigation of this issue.

Possible improvements

Caching results. Notice that the methods we describe do *not* have to be carried out every time that a client tries to access a resource. This would only have to be done for the first contact between a given client and a given resource. If the outcome is successful, the resource may remember this and grant access without a full search the next time.

Caching can also be used by the sites: unless a site is the client site or the resource site for some request, the result of its local search is independent of the request identifier. Therefore, sites may cache recent results and reuse them when an identical request (modulo *reqid*) comes along.

Guided search. In both protocols, the sets $pre^*(C)/post^*(C)$ may intersect the domains of many sites; therefore, any request could involve many different sites even if only a few of them are 'relevant' for the search. This increases the length of the computation as well as the amount of network traffic. Thus, the protocol could be improved by limiting the scope of the search. It is likely that the client has an idea of *why* he/she should be allowed to access the resource; therefore, one possibility would be to let the client and/or the client site suggest a set of sites that are likely to contain suitable certificates.

Termination. In the distributed GPP/GPS computation, a standard termination-detection algorithm can be applied to determine that the search has terminated, which entails additional time and communication overhead. However, even before the search has terminated, or before all relevant certificate chains have been found, the client site (in the GPS case) or the resource site (in the GPP case) may have discovered *some* paths with a tentative value (which may be 'larger' – with respect to the ordering – than the δ value). If the goal of the search is just to establish that the δ value is no larger than a certain threshold, then this information could be used to terminate the search early. Moreover, the computation could be limited by a timeout.

7 Implementation

We have implemented a prototype of our distributed certificate-chain-discovery algorithm. Figure 4 shows how a site is organized. Each SPKI/SDSI site consists of a SPKI/SDSI server and a WPDS server. The SPKI/SDSI server deals with SPKI/SDSI certificates and provides the interface for clients to perform requests for authorization. The WPDS server implements distributed certificate-chain discovery using an algorithm for solving reachability problems in Weighted Pushdown Systems (WPDS). The clients do not interact directly with the WPDS servers. In a typical authorization-request scenario, a client first initiates the request by contacting the SPKI/SDSI server (1). The SPKI/SDSI server then parses the request and sends it to the WPDS server at the same site (2). At this point, the WPDS server starts the distributed certificate-chain-discovery process and contacts other WPDS servers $(3, 4)$ as necessary. If a proof of authorization is found and verified, the client is granted access to the resource; otherwise the request is denied $(5, 6)$.

7.1 Examples

We illustrate how the system works using three examples. A graph is used to illustrate the configuration of sites for each example. In each graph, shaded nodes represents distinct sites of a distributed SPKI/SDSI system, while labels

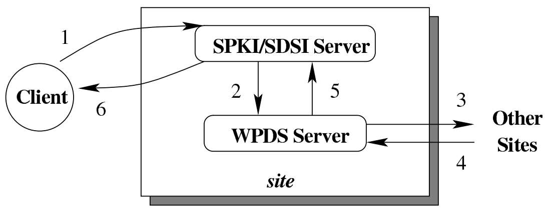

Fig. 4. Architecture Diagram Inside a Site

represent the cross-boundary SPKI/SDSI certificates. Nodes with a symbol *(R)* denote the resource from where SPKI/SDSI auth certs are issued. The dashed lines denote the certificate chain discovered by our algorithms when *Bob* requests access to resource R.

- **(Case 1)**: This case demonstrates the basic idea of distributed certificate-chain discovery. Let us assume that a university has the hierarchical structure shown in Figure 5, where each site represents one level of the university. Site *UW* denotes the top level of the University of Wisconsin; *LS* denotes one of the colleges of *UW*, i.e., the college of Letters and Sciences; while *CS* and *BIO* represent two departments under *LS*. Two sites are linked together if a SPKI/SDSI certificate refers to both sites. For instance, the site *UW* has issued two certificates with respect to site *LS*: the auth cert $K_r \ \square \xrightarrow{t} K_{uw}$ `faculty` $\blacksquare$ grants access right t to all K_{uw}'s faculty; the name cert K_{uw} `faculty` $\rightarrow K_{ls}$ `faculty` states that all K_{ls}'s faculty are K_{uw}'s faculty. Let us assume that *Bob*, from *CS*, requests access to a service R located at *UW*. The certificate-chain-discovery process starts from *UW* and continues down the hierarchy (*LS*, then *CS*) until it reaches *CS*, where *Bob* is granted access rights. Note that each individual site does not have sufficient knowledge to decide the authorization request. Instead, the certificates along the path must be used together to show that *Bob* has the required permissions.
- **(Case 2)** While Case 1 demonstrates the basic idea behind distributed certificate-chain discovery, Case 2 illustrates the situation where certificates from multiple paths must be combined to obtain the required authorization specifications (i.e., access permission). For instance, continuing with the example from Case 1, we now add a new joint department *BCS*, which is formed from both *CS* and *BIO* departments. The new structure is shown in Figure 6. Furthermore, *LS* issues two authorization certificates with distinct authorization specifications t_1 and t_2, to *CS* and *BIO*, respectively. Suppose that *Bob*, from *BCS*, wants to access R with both t_1 and t_2. This request cannot be granted if we followed either one of the two possible paths separately. The WPDS approach solves this issue by combining authorizations from both paths at *BCS*, and therefore will grant authorization to `Bob`.
- **(Case 3)**: The third case, shown in Figure 7, builds on top of the first two and demonstrates an even more complex environment. This case is constructed for two purposes. One, we want to demonstrate the scalability of the WPDS algorithm. Two, we want to study the performance with respect

to certificate-chain length. We will measure computation time against the length of chains in Section 7.2.

7.2 Performance Analysis

In this section, we report on the performance of our implementation, using the examples discussed before. We use response time from the perspective of clients as the performance metrics. Because we currently do not have the resources to perform a real-world test, all tests are conducted under a simulated environment: each site runs on a separate machine on a local area network. Therefore, the timing results do not reflect network latency in a real distributed environment. All test machines have 800 MHz Pentium III processors, 256 MB of RAM, running TAO Linux version 1.0.

For each experiment, we used three different configurations: *base*, *simple*, and *complex*. For comparison purposes, we also collected performance data for running certificate-chain discovery in centralized mode (i.e., all the certificates are stored at a single site), using the complex configuration.

- **Base configuration**: The base uses only the bare minimum number of certificates required for the tests (exactly as shown in Figures 5 - 7); the number of certificates ranges from 6 to 16 certs in these tests. We use the results from this configuration as the baseline for the other two test cases.
- **Simple configuration**: In a real-world scenario, each site would have more certificates. Each simple configuration adds between 60 and 160 certificates to the base configuration. For each site, we added a number of additional certificates (for students, staff, etc.), such as K_{uw} `student` $\rightarrow K_{ls}$ `student`, and K_{cs} `faculty` $\rightarrow K_{profA}$.
- **Complex configuration**: To measure how the system scales, we also tested each case using between 760 and 1600 certificates.

Table 1 shows the performance results for the three configurations. As one might expect, the more certificates there are in the system, the longer it takes to perform certificate-chain discovery. However, the time it takes to perform certificate-chain discovery increases at a lower rate compared to the increase in the number of certificates. The data shows insignificant changes from the base configuration to the simple configuration; and it shows a very small increase (about 4% on average) from simple to complex. Figure 8 illustrates this using data from case 3.[6] In addition, Table 1 shows that the performance difference between running certificate-chain discovery in distributed and in centralized mode is quite significant. For instance, in Case 3, distributed certificate-chain discovery took more than ten times as long as the centralized version. This is because in distributed certificate-chain discovery a significant percentage of time (about 80% to 93%) is spent on network-related operations, such as sending and receiving messages. We expect to be able to reduce some of the network overhead through optimizations. For example, we can reduce the number of messages exchanged during certificate-chain discovery by bundling several messages to-

[6] Two other cases tested showed similar results and therefore are omitted here.

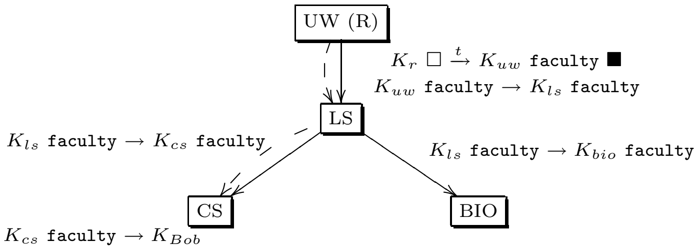

Fig. 5. (Case 1.): *R* grants `read` permission to directory `/etc` to *UW*'s faculty: t `= (tag (dir /etc (read)))`; *Bob* requests `read` access for directory `/etc`.

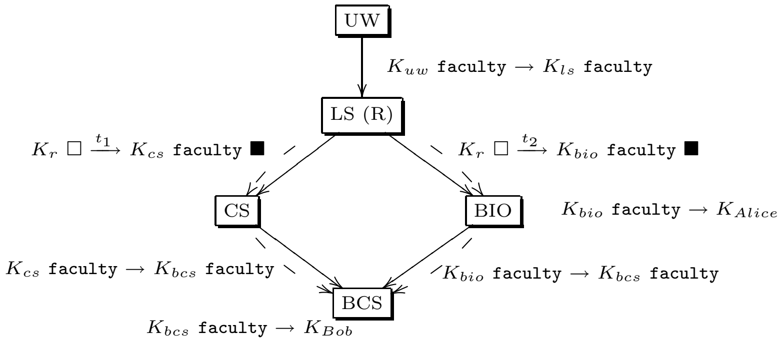

Fig. 6. (Case 2.): Authorization Over Multiple Paths. *R* grants `read` privilege to directory `/etc` to *CS*'s faculty: t_1 `= (tag (dir /etc (read)))`, and `write` privilege to *BIO*'s faculty: t_2 `= (tag (dir /etc (write)))`; *Bob* requests `(read write)` for the directory `/etc`.

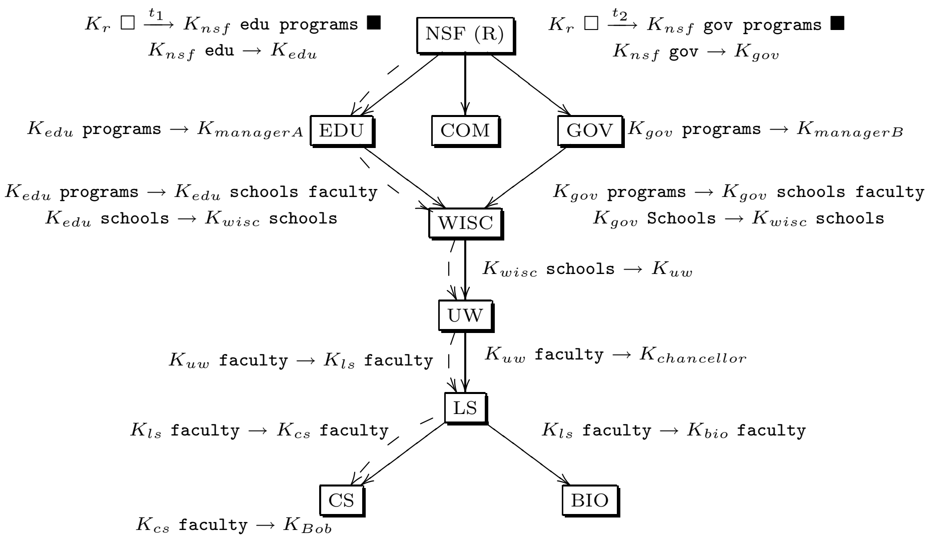

Fig. 7. (Case 3.): *R* authorizes all NSF's *EDU* programs to `apply` for `fundA`: t_1 `= (tag (fundA apply))`, and all NSF's *GOV* programs can `apply` for `fundB`: t_2 `= (tag (fundB apply))`; *Bob* attempts to `apply` for `fundA`.

gether and sending the bundle using one packet whenever possible. This is part of planned future work.

Table 1. Performance Results

	Time (ms) Distributed			
				Centralized
Client (Request)	**Base**	**Simple**	**Complex**	**Complex**
Case 1. See Figure 5				
Bob (`(dir /etc (read))`)	661	685	713	54
Case 2. See Figure 6				
Bob (`(dir /etc (read))`)	663	685	716	55
Bob (`(dir /etc (write))`)	717	730	741	55
Bob (`(dir /etc (read write))`)	723	736	741	55
Alice (`(dir /etc (write))`)	668	679	693	53
Case 3. See Figure 7				
ManagerA (`(fundA apply)`)	654	683	664	118
ManagerB (`(fundB apply)`)	793	769	796	116
Chancellor (`(fundA apply)`)	979	960	996	107
Bob (`(fundA apply)`)	1146	1133	1218	110
Bob (`(fundB apply)`)	1132	1150	1232	115

Performance data from Case 3 also illustrates an area for future work: *reducing response time for long certificate chains*. Here we define the length of a certificate chain as the number of distinct sites between the request site and the resource site. For example, Manager A is of chain length 1 since her site *EDU* is only one hop away from the resource site *NSF*. As illustrated by the ascending line at the top of Figure 9, the length of the certificate chain has a great impact on performance: the longer the chain, the longer it takes to service the request. For comparison purposes, the flat line shows the response time had we centralized all the certificates at one location. This time reflects the cost of running the GPS algorithm at one site, and therefore does not contain any network overhead. We are currently investigating techniques to improve the average performance for long certificate chains. For instance, in Section 6.3 we have discussed the possibility of using caching to reduce the discovery time.

References

1. M. Abadi. On SDSI's linked local name spaces. *Journal of Computer Security*, 6(1-2):3–21, 1998.
2. A. W. Appel and E. W. Felten. Proof-carrying authentication. In *Conf. on Comp. and Commun. Sec.*, Nov. 1999.
3. L. Bauer, S. Garriss, and M. K. Reiter. Distributed proving in access-control systems. In *In Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 81–95, May 2005.
4. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The role of trust management in distributed systems security. In Vitek and Jensen, editors, *Secure*

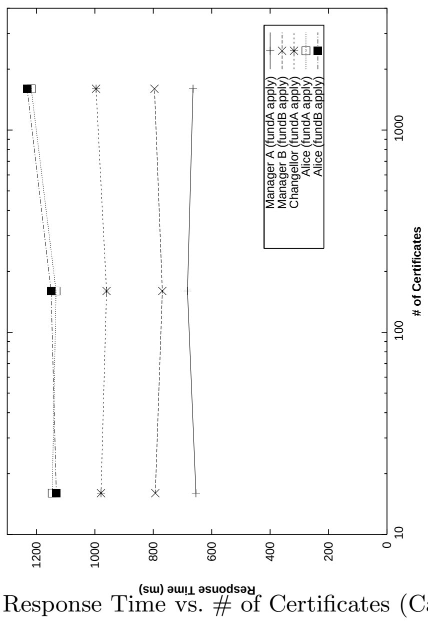

Fig. 8. Response Time vs. # of Certificates (Case 3.)

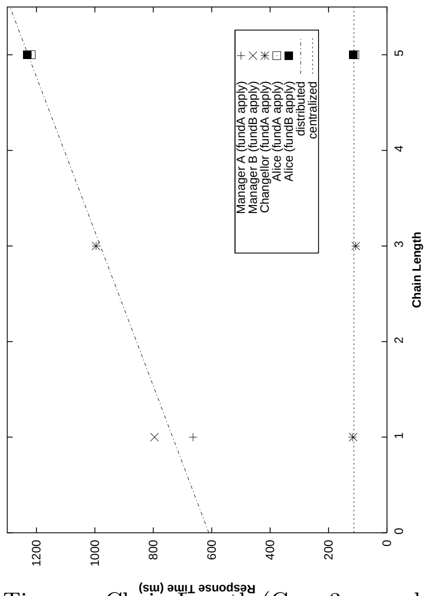

Fig. 9. Response Time vs. Chain Length (Case 3. complex configuration)

Internet Programming: Security Issues for Mobile and Distributed Objects, pages 185–210, 1999. LNCS 1603.

5. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The KeyNote Trust-Management System Version 2. RFC 2704, Sept. 1999.
6. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of CONCUR'97*, volume

1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.
7. A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. In *Proceedings of POPL'03*, 2003.
8. D. Clarke, J.-E. Elien, C. M. Ellison, M. Fredette, A. Morcos, and R. L. Rivest. Certficate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(1/2):285–322, 2001.
9. C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylönen. *RFC 2693: SPKI Certificate Theory.* The Internet Society, September 1999.
10. J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of CAV'2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 232–247. Springer, July 2000.
11. A. Finkel, B.Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. *Elec. Notes in Theor. Comp. Sci.*, 9, 1997.
12. J. Y. Halpern and R. van der Meyden. A logical reconstruction of SPKI. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pages 59–70. IEEE Computer Society Press, 2001.
13. J. Howell and D. Kotz. A formal semantics for SPKI. Technical Report 2000-363, Department of Computer Science, Dartmouth College, Hanover, NH, Mar. 2000.
14. S. Jha and T. Reps. Analysis of SPKI/SDSI certificates using model checking. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW)*, pages 129–146. IEEE Computer Society, June 2002.
15. S. Jha and T. Reps. Model checking SPKI/SDSI. *Journal of Computer Security*, 12(3–4):317–353, 2004.
16. T. Jim. SD3: A trust management system with certified evaluation. In *SP '01: Proceedings of the IEEE Symposium on Security and Privacy*, page 106. IEEE Computer Society, 2001.
17. T. Jim and D. Suciu. Dynamically distributed query evaluation. In *PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 28–39. ACM Press, 2001.
18. B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, November 1992.
19. N. Li and J. C. Mitchell. Understanding SPKI/SDSI using first-order logic. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW).* IEEE Computer Society, 2003.
20. N. Li, W. H. Winsborough, and J. C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, February 2003.
21. F. Pfenning and C. Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Int. Conf. on Auto. Deduc.*, pages 202–206. Springer-Verlag, LNAI 1632, July 1999.
22. T. Reps, S. Schwoon, and S. Jha. Weighted pushdown systems and their application to interprocedural dataflow analysis. In *Proceedings of the 10th Internation Static Analysis Symposium (SAS)*, San Diego, CA, June 11-13 2003.
23. T. Reps, S. Schwoon, S. Jha, and D. Melski. Weighted pushdown systems and their application to interprocedural dataflow analysis. *Science of Computer Programming*, 58(1-2):206–263, October 2005.
24. S. Schwoon, S. Jha, T. Reps, and S. Stubblebine. On generalized authorization problems. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 202–218. IEEE Computer Society, June 2003.

25. S. Weeks. Understanding trust management systems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Research in Security and Privacy, Oakland, CA, May 2001. IEEE Computer Society,Technical Committee on Security and Privacy, IEEE Computer Society Press.