Distributed Certificate-Chain Discovery in SPKI/SDSI

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

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

Technical Report
UW-CS-TR-1526

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. Although certificate-chain-discovery algorithms for SPKI/SDSI have been investigated by several researchers, previous work did not address how to perform certificate-chain discovery in distributed environments. We address the certificate-chain-discovery problem where the certificates are distributed over a number of sites, which then have to cooperate to identify the proof of authorization for a given request. We propose two protocols for this purpose. 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 [1, 2, 3], such as SPKI/SDSI [4], 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 certificate chain that is a proof of authorization for a request to access a resource by a principal.

Schwoon et al. [5] introduced a new algorithm for certificate-chain discovery that translates SPKI/SDSI certificates to rules in a weighted pushdown system (WPDS) [6]. The algorithm presented by [5] can discover proofs of authorization that consist of multiple certificate chains. In this approach, we also translate SPKI/SDSI certificates into rules in a WPDS, where the authorization specifications of the certificates are translated to weights of the rules. This translation to a WPDS yields a complete certificate-chain-discovery algorithm and is described in Section 4.

Although SPKI/SDSI was designed to provide trust management in distributed environments, its original proposal did not address how authorization (in the form of certificate chains) can be performed in *distributed* environments. This issue is also neglected by previous work on certificate-chain discovery. For instance, the algorithms of [7, 8, 5] assume that the set of all certificates relevant to a given request are known at a single site, at which is 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 certificate set. 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 the following contributions:

- We describe a distributed certificate-chain-discovery algorithm where the certificates are distributed across various sites. This distributed algorithm is described in Section 5.
- The algorithms that are presented in this paper are automaton-based. Compared with the rather limited amount of previous work that exists on authorization problem in a distributed setting, the automaton-based approach enjoys certain advantages over other approaches, such as [9]. In particular, (fragments of) automata can be computed at separate sites, and the information shipped between sites can take the form of (fragments of) automata.
- We have implemented a prototype system that incorporates our algorithm. Our experimental results, presented in Section 6, demonstrate that our distributed certificate-chain-discovery algorithm is both efficient and scalable.

Our approach also benefits from using WPDS as the underlying technology. Previous work on WPDS [5] pointed out that weighted domains enable one to address issues such as privacy, recency, validity, and trust. Moreover, weighted domains allow us to handle authorization specifications in a semantically correct manner.

Additionally, WPDS-based reachability-analysis algorithms can be used to answer certificate-analysis questions, such as "Is it the case that all authorizations to a resource R must involve a certificate signed by principal K?" Several other types of certificate-analysis questions are discussed in [10] and [8]. However, as this is orthogonal to the work presented here, we will not discuss it in this paper.

2. Related Work

A certificate-chain-discovery algorithm for SPKI/SDSI was first proposed by Clarke et al. [7]. An improved certificate-chain-discovery based on the theory of pushdown systems was presented by Jha and Reps [8]. Both of these algorithms are centralized and assume that the proof of authorization consists of a single certificate chain.

In general, 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 [7, 8] are incomplete. This observation is also the basis for the observation by Li and Mitchell [11] that the "5-tuple reduction rule" of [4] is incomplete. Our algorithm does not suffer from this problem, due to the translation into WPDS.

The semantics of SPKI/SDSI has also been studied in [12, 13, 14].

In the proof-carrying-authorization (PCA) framework of Appel and Felten [15], a client uses the theorem prover *Twelf* [16] to construct a proof of authorization, which the client presents to the server. However, they too assume that all logical facts used by theorem prover reside at a single server. Li et al. [9] presented a distributed credential-chain-discovery algorithm for the trust management system RT_0. Their algorithm allows credentials to be distributed, but the proof of authorization is constructed at one site. The credential-chain-discovery algorithms of Li et al. fetches credentials from other sites as needed. SPKI/SDSI is a subset of RT_0 (SPKI/SDSI is equivalent to RT_0 without role intersection). In our distributed credential-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 our knowledge, Li et al. did not implement their algorithm.

The work by Jim and Suciu on SD3 [17, 18], 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 [17], the author claims that SD3 can express "roughly the same policies as SDSI 2". While this claim is not further substantiated in [17], 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 [17] 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 5).

– Unlike SD3, our framework allows certificates to have weights. As pointed out in [19], 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 [11]. Moreover, in [5], we pointed out that weights allow to address such issues as privacy, recency, validity, and trust.

3. 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 `A`. An *extended name* is of the form $K\ \sigma$, 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.

3.1. 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. `A` is an identifier. The subject S is a term. Intuitively, S gives additional meaning for the local name K `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 $K \square \xrightarrow{T} S \blacksquare$.

3.2. Authorization

Since we will only use labeled rewrite rules in this paper, we will 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 $L \xrightarrow{T} R$ corresponding 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,

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

[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 [4, 12].

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:

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

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

3.3. The Authorization Problem in SPKI/SDSI

Assume that we are given a set of certs $\mathcal{C}$ and that principal K_r owns a set of resources that are identified by authorization specifications. Moreover, assume that principal K_c wants access specified by authorization specification T. (In the following, we call K_r the **resource owner** and K_c the **client**.) The authorization question is: "Can K_c be granted access to the resource specified by T?"

A *certificate chain* ch for $\mathcal{C}$ is of the form $c_k \circ c_{k-1} \circ \cdots \circ c_1$, where $c_1, c_2, \cdots, c_k$ are certificates in $\mathcal{C}$. The label of a certificate chain ch is denoted by $L(ch)$. Given $\mathcal{C}$, K_r, K_c, and T, a *certificate-chain-discovery* algorithm looks for a finite set of certificate chains proving that K_c is allowed 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 i, where $1 \le i \le m$,

$$ch_i(K_r \,\square) \in \{K_c \,\square, K_c \,\blacksquare\} \quad \text{and} \quad T \subseteq \bigcup_{i=1}^{m} L(ch_i).$$

Clarke et al. [7] presented an algorithm for certificate-chain discovery in SPKI/SDSI with $O(n_K^2|\mathcal{C}|)$ time complexity, 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: a solution could only consist of a *single* certificate chain. For instance, consider the following certificate set:

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

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. [5] presented algorithms for full certificate-chain discovery, 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 by [5].

4. Weighted Pushdown Systems and SPKI/SDSI

In this section, we introduce weighted pushdown systems, briefly review the algorithms proposed for them, and then show that they are a useful tool for solving problems related to certificate-chain discovery in SPKI/SDSI. The following definitions are largely taken from [20].

[5] In general, the composition operator $\circ$ is not associative. For example, c_3 can be compatible with $c_2 \circ c_1$ but not 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.

4.1. Weighted Pushdown Systems

Weighted pushdown systems were introduced in [21, 20, 5]. 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* $\Rightarrow$ *between configurations of* $\mathcal{P}$ *as follows:*

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

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 *via the transition relation.* 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* $V \subseteq Conf(\mathcal{P}) \times D$ *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.

4.2. 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 $Conf(\mathcal{P})$. Then, according to [22, 23], the sets $pre^*(C)$ and $post^*(C)$ are also regular and effectively computable (in the form of a finite automaton).

The results from [20, 5] 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 $c \in pre^*(C)$:*

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

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

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

In [20, 5], the solutions for GPS and GPP are computed in the form of annotated finite automata. We briefly review these solutions in Section 4.4.

4.3. The Connection Between SPKI/SDSI and Weighted Pushdown Systems

The following correspondence between SPKI/SDSI and pushdown systems was presented in [5]: 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 that appear in $\mathcal{C}$, respectively. Moreover, let $\mathcal{T}$ be the set from which the auth specs in $\mathcal{C}$ are drawn. Then $\mathcal{S}_\mathcal{C} = (\mathcal{T}, \cup, \cap, \bot, \top)$, where $\cup, \cap$ are the union and intersection of auth specs as discussed in [4, 12], forms a semiring with domain $\mathcal{T}$. Now we can 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 and 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 specs and auth specs as shown in Section 3.1, and f maps every rule to its corresponding auth spec.

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 amounts to the finding 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.

To conclude, in the generalized authorization problem, we pose the following question:

> Given a set of certificates $\mathcal{C}$, a resource owner K_r, an authorization specification T, and a client K_c, are there certificate chains in $\mathcal{C}$ proving that K_r grants authorization T to K_c?

This 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 $t := \delta(c)$ 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 $t := \delta(c_1) \oplus \delta(c_2)$ and forwards witness dags for $(c_1, \delta(c_1))$ and $(c_2, \delta(c_2))$.

Authorization for K_c is granted if and only if $t \supseteq T$.

4.4. Algorithms for GPR

We briefly review the solutions for GPR problems given in [20, 5], concentrating on the GPP case, because the GPS case is analogous except for some details.

Our input is a weighted pushdown system $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$, where $\mathcal{P} = (P, \Gamma, \Delta)$ and $\mathcal{S} = (D, \oplus, \otimes, 0, 1)$, and a regular set C of configurations. The output is $\delta(c)$ and a witness dag for $(c, \delta(c))$ for each $c \in pre^*(C)$.

In general, there are infinitely many configurations in $pre^*(C)$ (and in $post^*(C)$) 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 paths 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 the following $\mathcal{P}$ is fixed, so 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 [23]. 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 [20, 5], two additional labellings 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. In the following presentation, we omit the second labeling for the sake of simplicity. A detailed presentation is given in [20], and the method there is straightforward to transfer to the distributed case.

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, see e.g. [8].

In the following, we first present an abstract version of the procedure given in [20, 5], which is designed for centralized computation. Section 5 describes an implementation for the distributed case.

Abstract algorithm Let $\mathcal{A} = (Q, \Gamma, \eta, P, F)$ be a $\mathcal{P}$-automaton accepting 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 algorithm stops when further applications of the saturation rule cause no further changes in $\mathcal{A}$.

Pseudocode for the algorithm is given in [20] and reproduced in Figure 1. Each iteration of the loop starting at line 14 executes one or more applications of the saturation rule.

Example Assume that the pushdown system contains the following rules (the meaning of this example is explained in greater detail in Subsection 6, Case 1):

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

Let $f(r_1) = t$, (i.e., the auth cert grants permission t), and $f(r_i) = \top$ for $2 \leq i \leq 5$ (i.e., the name certs do not change permissions). Suppose that Bob wants permission t from the resource owner K_r. We determine whether Bob is authorized to do so by solving the GPP problem for $C = \{\langle K_{Bob}, \square\rangle, \langle K_{Bob}, \blacksquare\rangle\}$.

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

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

procedure *update*(t, v)
begin
  $\eta := \eta \cup \{t\}$
  $newValue := l(t) \oplus v$
  if $newValue \neq l(t)$ **then**
    $workset := workset \cup \{t\}$
    $l(t) := newValue$
end

$\eta := \eta_0;\ workset := \eta_0;\ l := \lambda t.0$
for all $t \in \eta_0$ **do** $l(t) := 1$
for all $r = \langle p, \gamma \rangle \hookrightarrow \langle p', \varepsilon \rangle \in \Delta$ **do**
  $update((p, \gamma, p'), f(r))$
while $workset \neq \emptyset$ **do**
  remove some transition $t = (q, \gamma, q')$ from $workset$;
  for all $r = \langle p_1, \gamma_1 \rangle \hookrightarrow \langle q, \gamma \rangle \in \Delta$ **do**
    $update((p_1, \gamma_1, q'), f(r) \otimes l(t))$
  for all $r = \langle p_1, \gamma_1 \rangle \hookrightarrow \langle q, \gamma\gamma_2 \rangle \in \Delta$ **do**
    for all $t' = (q', \gamma_2, q'') \in \eta$ **do**
      $update((p_1, \gamma_1, q''), f(r) \otimes l(t) \otimes l(t'))$
  for all $r = \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p', \gamma_2\gamma \rangle \in \Delta$ **do**
    if $t' = (p', \gamma_2, q) \in \eta$ **then**
      $update((p_1, \gamma_1, q'), f(r) \otimes l(t') \otimes l(t))$
return $((Q, \Gamma, \eta, P, F), l)$

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

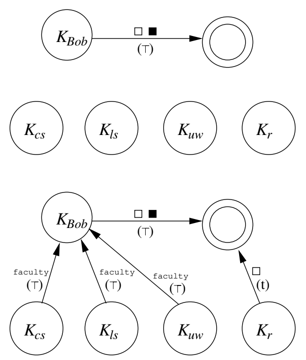

Fig. 2. Initial automaton (above) and final pre^* automaton created by the algorithm in Figure 1; weights on transitions are shown in parentheses.

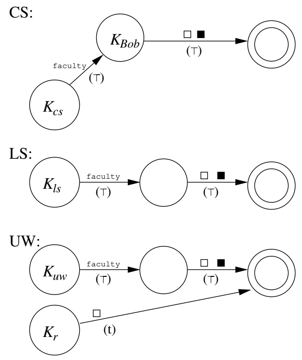

Fig. 3. Partial automaton computed at CS, LS, and UW for the query $pre^*(\langle R, \Box \rangle)$: weights on transitions are shown in parentheses.

The upper part of Figure 2 shows an automaton that accepts C. The automaton for $pre^*(C)$, produced by Algorithm 1, is shown in the lower part. There, we can see that $\langle K_r, \Box \rangle$ is accepted with weight t, and so Bob's authorization is granted. The extra annotations for witness dags (not shown) would let us deduce that the relevant certificate chain is $[r_1, r_2, r_3, r_5]$.

5. Distributed Certificate-Chain Discovery

The algorithms for GPR problems discussed in Section 4.4 work under the assumption that all pushdown rules (or certificates, resp.) are stored at a single site. In a real-world setting, certificates may be issued by many principals, and forcing them to be stored at (or shipped to) a single site may not be permitted. We therefore propose versions of these algorithms that solve GPR problems in a distributed environment.

Remark: Because of the connection between SPKI/SDSI and WPDS explained in Section 4.3, it is safe to use pushdown and SPKI/SDSI terminology interchangeably, and we shall do so in this section.

We proceed as follows: Section 5.1 introduces some definitions and notation. Section 5.2 gives high-level descriptions of protocols for the communication between the client, the resource, and the servers that co-operate to solve the distributed certificate-chain-discovery problem. We propose two protocols, one based on the GPP formulation, the other on the GPS formulation. Both protocols consist of several phases, the core of which is a search phase. The algorithms used in that phase are described in further detail in Section 5.3. The relative merits of the protocols, as well as security and privacy-related issues, are discussed in Section 5.4.

5.1. Preliminaries

For the rest of the section, let us fix a weighted pushdown system $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$, where $\mathcal{P} = (P, \Gamma, \Delta)$ and $\mathcal{S} = (D, \oplus, \otimes, 0, 1)$. We consider the authorization problem where client K_c requests permission T from the resource owner K_r.

We assume that the certificates are distributed over a set $Sites$ of **servers**, and that there exists a mapping $f_S \colon \mathcal{P} \to Sites$, which maps each principal to a site that is 'responsible' for the principals.

We say that certificate $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$ **crosses a site boundary** if $f_S(p) \neq f_S(p')$. If such a cross-boundary certificate exists, we call the sites responsible for p and p' **neighbouring sites**.

Moreover, we denote by $\mathcal{T}(s) = \{ \langle p, w\rangle \mid f_S(p) = s,\ w \in \Gamma^* \}$ the configurations that begin with the keys for which site s is responsible. The basic idea behind the distributed algorithms is that every site s computes (an automaton representation of) the set $pre^*(C) \cap \mathcal{T}_{\mathcal{C}}(s)$ or $post^*(C) \cap \mathcal{T}_{\mathcal{C}}(s)$, respectively. Moreover, s annotates its automaton with information that allows recovering part of the witness dags. This notion is made more precise in the following definition:

Definition 7. *Let $\mathcal{D} = (V, E)$ be a $(\mathcal{W}, C)$-dag and $s \in Sites$. The s-**slice of** $\mathcal{D}$ is the subgraph of $\mathcal{D}$ induced by the vertices $V_1^s \cup V_2^s$, where*

- $V_1^s \stackrel{\text{def}}{=} \{ (c, d) \in V \mid c \in \mathcal{T}(s) \}$
- $V_2^s \stackrel{\text{def}}{=} \{ v \in V \mid \exists v' \in V_1^s, r \in \Delta\colon (v', r, v) \in E \}$

Informally, the s-slice contains the part of $\mathcal{D}$ that consists of configurations for which s is responsible, and their immediate successor vertices (reached by cross-boundary certificates). An edge labeled by a cross-boundary certificate is henceforth called a **boundary edge***. v is called a* **boundary node of** *s if v is the target of a boundary edge, and s is responsible for the subject of the rule with which the edge is labeled.*

5.2. The Protocols

Our distributed solutions for the authorization problem make certain assumptions about the storage of certificates:

- In the GPP protocol, we assume that every certificate/rule $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$ is stored at the site responsible for its subject, i.e., at $f_S(p')$.
- In the GPS protocol, we assume that every certificate/rule $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$ is stored at the site responsible for its issuer, i.e., at $f_S(p)$.

These assumptions will make our algorithms more efficient because every site will know which other sites to contact for information concerning any given principal. The assumptions are realistic: they are basically saying that if a certificate mentions a principal (either the subject or the issuer), then its site should know about it. (In general, it would be realistic to assume that each certificate is known to the sites of *both* the issuer and the subject, but the stated conditions are the only ones actually required by our algorithms).

In a distributed setting, multiple access requests may happen at the same time. We shall use unique *request ids* to distinguish among them. Both protocols consist of three phases, *initialization*, *search*, and *verification*.

The GPP Protocol for Distributed Certificate-Chain Discovery In this setting, the search is started at the site that is responsible for the client, and the search works its way "up" towards the site that is responsible for the resource owner from whom the client is requesting permission.

Initialization Initialization consists of the following steps:

1. The client sends a request T to the resource owner.
2. The resource owner generates a unique request identifier $reqid$, which will distinguish this request from other requests that may be in progress now or in the future, until request $reqid$ is resolved.
3. The resource owner sends the pair $(K_r, reqid)$ to the site $f_S(K_r)$ (called the *resource site* and denote s_r from now) to notify it of an 'incoming' search. After s_r has acknowledged receipt of the message, $reqid$ is sent to the client.

4. The client sends a message to the site $f_S(K_c)$ (called the *client site* and denoted s_c). The message contains (i) its key K_c, (ii) the request id $reqid$, (iii) a so-called *client certificate*, i.e. the request id signed by the client.
5. The client site checks that the contents and signature of the client certificate match expectations. If the check is successful, s_c begins the search.

Search The client site initiates a GPP query for the set $C = \{\langle K_c, \blacksquare\rangle, \langle K_c, \square\rangle\}$, where $reqid$ is used to distinguish this query from others (so that servers may work on multiple requests simultaneously). The query is resolved by all the sites together; the details of the search algorithm are given in Section 5.3. At this point, it is sufficient to understand the following: s_c starts a local GPP computation, and may (transitively) request other sites to participate in the computation; each site s constructs the set $pre^*(C) \cap \mathcal{T}(s)$, and maintains information that allows constructing the s-slice of the required witness dags. Communications between sites are tagged with both $reqid$ and the client certificate.

Verification Because of its earlier communication with the resource owner, the resource site s_r knows that $c = \langle K_r, \square\rangle$ is the target of the search. Moreover, because $c \in \mathcal{T}(s_r)$, the resource site will be able to determine whether c is reachable from C, using the set $post^*(C) \cap \mathcal{T}(s_r)$ it has computed. To complete the algorithm, the result must be reported to the resource owner.

We propose two alternative methods:

– In the first alternative, the resource site starts by constructing the s_r-slice of the witness dag. When it reaches a boundary node of its slice, it requests the sub-dag ‘below’ that node from the neighbouring site at that node. The neighboring site computes this information, which possibly involves recursive queries to sites further ‘downstream’, and returns it to s_r. When s_r has constructed the full witness dag, it sends it to the resource owner along with the client certificate. The resource owner 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 second alternative is as follows: instead of constructing the witness graph, s_r just reports the certificates issued by the owner for the resource, the combined values of the paths that start with them, and the client certificate. In that case, no further communication between the sites is necessary.

The first alternative provides the resource owner with the complete witness set of certificate chains. This may give the owner a higher degree of confidence and control over the authorization process. On the other hand, the verification of the complete dag may place a great workload on the resource owner, which is reduced in the second alternative. The second alternative may also drastically reduce the amount of network traffic exchanged between sites.

The GPS Protocol for Distributed Certificate-Chain Discovery In this setting, the search is initialized at the resource site, and the search works its way “down” to the client.

Initialization

1. The client sends a request T to the resource owner.
2. The resource owner responds by sending a unique request identifier $reqid$.
3. The client sends a message to the client site s_c to register the search. Along with the message, it sends $reqid$ and the client certificate as in the GPP protocol.
4. The client site again checks correctness of the client certificate. If the check is successful, the client site tells the client that certificate-chain discovery may begin.
5. The client asks the resource owner to initiate the search.
6. The resource owner sends a message to its resource site containing its public key K_r, the request id $reqid$, and a request to initiate a certificate discovery.

Search The search stage is analogous to the GPP protocol, except that it is initiated by the resource site and from the singleton set $C = \{\langle K_r, \square\rangle\}$. The details of the search algorithm are given in Section 5.3. In brief, a site s becomes involved in the search if $post^*(C)$ intersects $\mathcal{T}(s)$, and s maintains information that allows constructing the s-slice of the required witness dags.

Verification Because of steps 3 and 4 in the initialization phase, the client site s_c knows that $c_1 = \langle K_c, \square \rangle$ and $c_2 = \langle K_c, \blacksquare \rangle$ are the targets of the search. Moreover, it can determine whether c_1 and c_2 are reachable from C, using the set $post^*(C) \cap \mathcal{T}(s_c)$ it has computed. To complete the algorithm, the result must be reported to the resource owner. In this phase, the direction of the flow of information is contrary to that of the search phase.

Like in the GPP protocol, we have two alternatives at this stage, which are analogues of the ones provided for GPP. For a discussion about their relative merits, see the remarks in the GPP protocol.

- In the first alternative, the client site starts by constructing the s_c-slice of the witness dags. It then sends the sub-dags starting at its boundary nodes 'upstream' to the corresponding neighboring sites. The neighboring sites supplement this information with 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 owner. Moreover, all communications in this phase are accompanied by the client certificate mentioned earlier.
  The resource owner can now verify the result, and grant or deny access to the client.
- In the second alternative, the sites only report the sum (w.r.t. $\oplus$) of the paths inside their slices of the witness dags. Then, the result given by s_r to the resource owner consists of certificates issued by K_r and the combined values of the paths below them.

5.3. Distributed Search Algorithms

In this section, we give some more details about the Search phase of the protocols.

At an abstract level of description, every site s computes the set $pre^*(C) \cap \mathcal{T}_\mathcal{C}(s)$ (or $post^*(C) \cap \mathcal{T}_\mathcal{C}(s)$, respectively). Site s becomes involved in the search if it is discovered that its intersection is non-empty. In the GPP protocol, the client site starts with the set $C = \{\langle K_c, \square \rangle, \langle K_c, \blacksquare \rangle\}$; in the GPS protocol, the resource site starts with the set $C = \{\langle K_r, \square \rangle\}$. If a cross-boundary cert causes some site s to discover terms belonging to $\mathcal{T}_\mathcal{C}(s')$ (for some other site s'), then s will send those terms to s', and s' continues the computation on those terms. All terms communicated between sites will be tagged with the request id, so that sites can distinguish among them when working on multiple queries.

At a more concrete level of description, the resource/client site starts by building an automaton accepting C, then carries out the algorithm from Figure 1 (or its $post^*$ counterpart [20], respectively), using its own certificates. If it derives an automaton transition $t = (K, \gamma, q)$ that begins at a state K (key, respectively) for which another site is responsible, then t and the *part of the automaton* reachable from q are shipped out to that other site. Thus, every site computes a "partial" automaton (i.e., a fragment of the full automaton).

Example: Consider once more the example from Section 4.4, and assume that the rules r_1 to r_5 are distributed over four sites called UW, LS, CS, and Bio as shown in Figure 4 of Section 6. Suppose that we use the GPP protocol to decide whether Bob at site CS is granted permission t by K_r. 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 partial automata computed by CS, LS, and UW are shown in Figure 3; notice that site Bio does not get involved. At the end of the computation, site UW sees that $\langle K_r, \square \rangle$ is accepted by its partial automaton with weight t, and that is the result reported to the resource owner.

Bidirectional search The approaches discussed so far allow for unidirectional search, either "forward" (from resource to client) or "backwards" (from client to resource). Taking a leaf from [9], one could envisage a hybrid algorithm that works in both directions at once. In this case, the resource site would initiate a GPS query, and the client site would initiate a GPP query, both with the same request id. All sites would maintain two automata, one for each direction. Because the intersection of two automata can be performed efficiently, a site "in the middle" would be able to notice when the two searches intersect. We have not investigated this approach in our prototype, but it does present an interesting direction for future work.

5.4. Discussion

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

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

– *Only the resource owner and the client know that the client has asked to access the resource.* This is because the resource owner and the client do not give out information about each other when they communicate with their sites. The sites can determine the outcome of a search just by using the request id, which is generated independently of either key.
– *The resource site knows that a request for the resource has been made, but not by whom.* Once again, this is because the resource site receives only a request id from the resource owner. Moreover, the resource site maintains only a partial automaton and a slice of the associated witness graph, so it cannot determine anything about principals at any other site.
– *The client site knows only that the client has made a request, but not for what or to whom.* This holds for reasons that are analogous to the previous argument.
– 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 phase. 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 between sites. This alternative solution also prevents the unnecessary spread of certificates between sites (which might contain sensitive information).

Security against attacks

Spoofing and eavesdropping. To protect the protocol-related communication from attacks such as spoofed messages or eavesdropping, all messages exchanged in the protocols are encrypted and digitally signed, e.g., using any of the well-established public-key cryptography systems.

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 cross-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 (minimal, resp.) 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 client's company, his/her ISP, etc), and thus there is some 'social safeguard' against denial-of-service attacks.

Moreover, the GPP-based solution does not require a separate verification stage when the construction of complete witness dags is omitted. 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 analysis on this topic.

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 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.

In SPKI/SDSI, certificates may be annotated with validity information that specifies how long a certificate is valid (see Section 3.1). A certificate chain is valid only as long as all certificates on it are valid. In both situations described above, the caching must take this validity information into account. This requires some straightforward additions to our algorithms that are omitted here.

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 resource site (in the GPP case) or the client site (in the GPS 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. For instance, if Alice is interested in a set of certificate chains that is valid for at least one hour, then the search algorithm could be stopped as soon as certificate chains are found that are valid for, say, one and a half hours (or any other amount of time longer than one hour). Moreover, the computation could be limited by a timeout.

6. Implementation

We have implemented a prototype of our distributed certificate-chain-discovery algorithm. We use the prototype to evaluate the scalability of the algorithm by varying two parameters: *configuration topology*, and *number of certificates*. 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 experiments are configured using synthetic data. However, we tried to make sure that the configurations mimicked the real world as closely as possible. The two main conclusions that can be drawn from these experiments are:

- **Network overhead is dominant**: The results show that the topology of the configuration affects the system performance. However, the most significant factor is network overhead: our experiments show that a significant percentage (about 80% to 90%) of the total time is spent on network operations. However, since our current implementation is only a prototype, we can reduce the overhead using optimization techniques.
- **Local filtering effect**: As one might have expected, the more certificates we have, the longer it takes to perform certificate-chain discovery. However, the time that it takes to perform certificate-chain discovery increases at a much smaller rate than the increase in the number of certificates. This is due to what we call the *local filtering effect*: only local rules are processed at each site (see Sections 5.2 and 5.3).

In summary, the experimental results show that the distributed certificate-chain algorithm implemented on top of WPDS is both efficient and scalable. In the rest of this section, we explain the experimental design and discussed the results obtained.

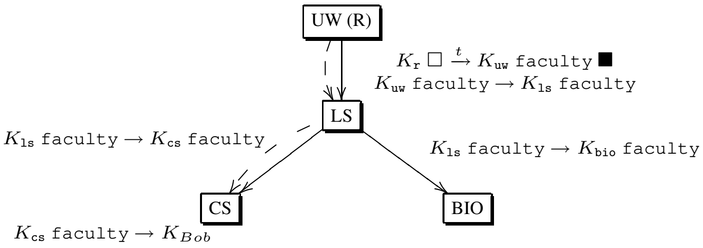

Fig. 4. (Case 1.): *R* grants `read` permission to directory `/etc` to *UW*'s faculty: t = `((dir /etc) read)`; *Bob* requests `read` access for the directory `/etc`.

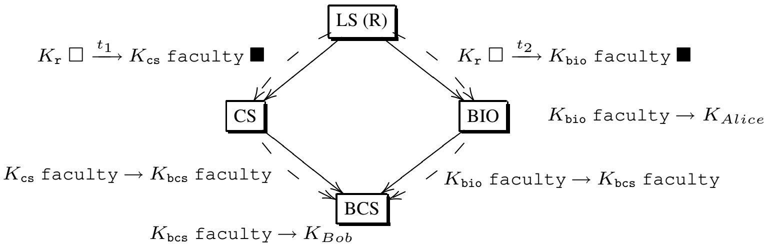

Fig. 5. (Case 2.): Authorization Over Multiple Paths. *R* grants `read` privilege to directory `/etc` to *CS*'s faculty: t_1 = `((dir /etc) read)`, and `write` privilege to *BIO*'s faculty: t_2 = `((dir /etc) write)`; *Bob* requests `(read write)` access for the directory `/etc`.

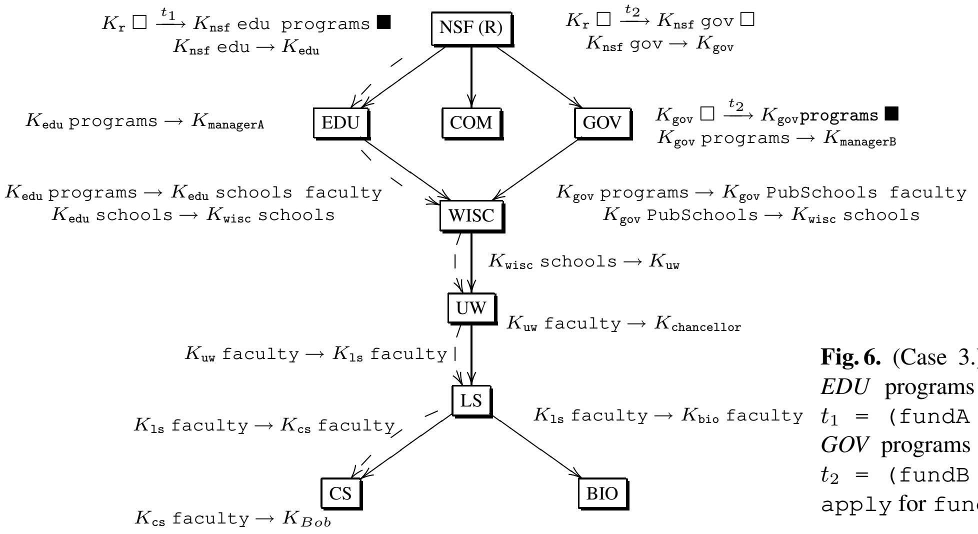

Fig. 6. (Case 3.): *R* authorizes all NSF's *EDU* programs to `apply` for `fundA`: t_1 = `(fundA apply)`, and all NSF's *GOV* programs can `apply` for `fundB`: t_2 = `(fundB apply)`; *Bob* attempts to `apply` for `fundA`.

Configuration Topology The first parameter that we considered is the configuration topology. A configuration of the SPKI/SDSI system consists of multiple connected sites and is represented as a graph. In our experiments, we considered three different configurations of varying degrees of complexity, as shown in Figures 4 – 6. In each graph, shaded nodes represent distinct sites of a distributed SPKI/SDSI system, while labels 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*.

For example, in Case 1, the root node *UW* denotes the University of Wisconsin; *LS* denotes the college of Letters and Science, one of the colleges of *UW*; while *CS* and *BIO* represent two departments, Computer Science and Biology, under *LS*. There is an edge between *UW* and *LS* because *UW* has issued two certificates with respect to site *LS*: the auth cert $K_r \square \xrightarrow{t} K_{uw}$ faculty ■ grants access right t to all K_{uw}'s faculty; the name cert K_{uw} faculty → K_{ls} faculty states that all K_{ls}'s faculty are K_{uw}'s faculty.

Case 1 represents the simplest topology of the three configurations. Case 2 adds additional complexity on top of Case 1 by forming a DAG. As a result, a certificate chain may consist of multiple paths, as demonstrated by the dashed lines in Figure 5. Case 3 builds on top of the first two and forms a more complex configuration.

Number of Certificates The second parameter that we considered is the number of certificates. In a real-world environment every principal has a key and can issue certificates. Consequently, we expect a SPKI/SDSI system to contain thousands or more certificates, distributed over the various sites in the system. Hence, for each of the three configurations, we varied the number of certificates used in the experiments ranged from 6 to 1600. We consider only the number of reachable certificates since unreachable ones do not contribute to certificate-chain discovery. Table 1 shows the breakdown of the certificates of the experiments.

	Number of Certificates		
Configuration	**Small**	**Medium**	**Large**
Case 1	1 : 5	3 : 57	30 : 570
Case 2	2 : 4	4 : 56	40 : 560
Case 3	3 :13	5 :155	50 :1550

Table 1. Test Configurations. Each cell contains two numbers: the first is the number of auth certs used in the run and the second number of name certs.

6.1. Analysis

In this section, we report on the results of the experiments. We use response time from the perspective of clients as the performance metric. All tests were conducted under a simulated environment: each site runs on a separate machine on a local area network. All test machines have identical configurations: 800 MHz Pentium III with 256MB RAM, running TAO Linux version 1.0.

Table 2 shows the performance results for the three configurations. Each configuration is run three times, with increasing numbers of certificates. For comparison purposes, we also collected performance data for running certificate-chain discovery in centralized mode (i.e., with all the certificates stored at a fixed site), using the largest number of certificates.

Topology effect: The data from Table 2 shows that the effect of configuration topology varies from case to case. For instance, comparing Cases 1 and 2 one can observe that the performance difference is small. This indicates that the path-combining operation (in Case 2) adds little overhead. However, in Case 3, we see a substantial variance in the time to process queries. One obvious observation is that the certificate-chain length affects the performance, as shown by the top line in Figure 7. In comparison, the flat line in the same figure 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, plus the network overhead of the two communicating between the client and the site (request and reply). This reveals that the most significant factor is *network overhead*. We collected additional data that confirmed this hypothesis: in distributed certificate-chain discovery, about 80% to 93% of the time is spent on network-related operations, such as establishing TCP connections, sending and receiving messages. Since this is currently a prototype, we are investigating optimization techniques to improve the average performance.

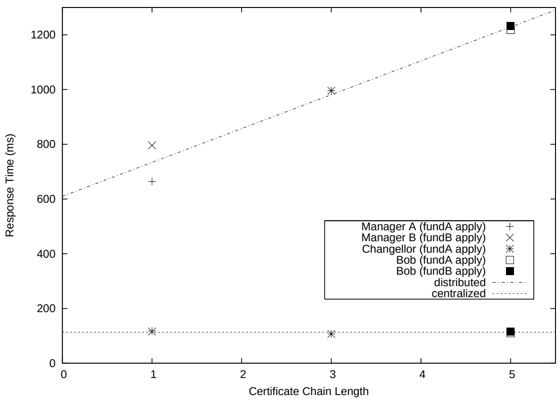

Fig. 7. Response Time vs. Chain Length (Case 3. complex configuration)

Number of certificates: Table 2 shows that there is an insignificant change in performance when the number of certificates increases from *small* to *medium* and a very small increase (about 4% on average) from *medium* to *large*. We attribute this to two reasons. First, the *local filtering effect* caused only relevant rules to be composed at each site. This corresponds to Lines 12-13 of the algorithm shown in Figure 1. Second, the WPDS methodology is efficient.

References

[1] Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.: The KeyNote trust-management system version 2. RFC 2704 (1999)

[2] Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.: The role of trust management in distributed systems security. In Vitek, Jensen, eds.: Secure Internet Programming: Security Issues for Mobile and Distributed Objects. (1999) 185–210 LNCS 1603.

Table 2. Performance Results

Configuration	Client	Request	Time (ms) Distributed Small	Medium	Large	Centralized Large
Case 1	Bob	`(dir /etc (read))`	661	685	713	54
Case 2	Bob	`(dir /etc (read))`	663	685	716	55
	Bob	`(dir /etc (read write))`	723	736	741	55
Case 3	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

[3] Weeks, S.: Understanding trust management systems. In: Proceedings of the IEEE Symposium on Research in Security and Privacy. Research in Security and Privacy, Oakland, CA, IEEE Computer Society,Technical Committee on Security and Privacy, IEEE Computer Society Press (2001)

[4] Ellison, C.M., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylönen, T.: RFC 2693: SPKI Certificate Theory. The Internet Society. (1999)

[5] Schwoon, S., Jha, S., Reps, T., Stubblebine, S.: On generalized authorization problems. In: Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW), IEEE Computer Society (2003) 202–218

[6] Reps, T., Schwoon, S., Jha, S.: Weighted pushdown systems and their application to interprocedural dataflow analysis. In: Proceedings of the 10th Internation Static Analysis Symposium (SAS), San Diego, CA (2003)

[7] Clarke, D., Elien, J.E., Ellison, C.M., Fredette, M., Morcos, A., Rivest, R.L.: Certficate chain discovery in SPKI/SDSI. Journal of Computer Security **9** (2001) 285–322

[8] Jha, S., Reps, T.: Analysis of SPKI/SDSI certificates using model checking. In: Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW), IEEE Computer Society (2002) 129–146

[9] Li, N., Winsborough, W.H., Mitchell, J.C.: Distributed credential chain discovery in trust management. Journal of Computer Security **11** (2003) 35–86

[10] Li, N., Winsborough, W.H., Mitchell, J.C.: Beyond proof-of-compliance: Safety and availability analysis in trust management. In: Proceedings of 2003 IEEE Symposium on Security and Privacy (Oakland), Berkeley, CA (2003)

[11] Li, N., Mitchell, J.: Understanding SPKI/SDSI using first-order logic. In: Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW), IEEE Computer Society (2003)

[12] Howell, J., Kotz, D.: A formal semantics for SPKI. Technical Report 2000-363, Department of Computer Science, Dartmouth College, Hanover, NH (2000)

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

[14] Halpern, J., van der Meyden, R.: A logical reconstruction of SPKI. In: Proceedings of the 14th IEEE Computer Security Foundations Workshop, IEEE Computer Society Press (2001) 59–70

[15] Appel, A., Felten, E.: Proof-carrying authentication. In: Conf. on Comp. and Commun. Sec. (1999)

[16] Pfenning, F., Schürmann, C.: System description: Twelf — a meta-logical framework for deductive systems. In Ganzinger, H., ed.: Int. Conf. on Auto. Deduc., Springer-Verlag, LNAI 1632 (1999) 202–206

[17] Jim, T.: SD3: A trust management system with certified evaluation. In: SP '01: Proceedings of the IEEE Symposium on Security and Privacy, IEEE Computer Society (2001) 106

[18] Jim, T., Suciu, D.: Dynamically distributed query evaluation. In: PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM Press (2001) 28–39

[19] Jha, S., Reps, T.: Model checking SPKI/SDSI. Journal of Computer Security **12** (2004) 317–353

[20] Reps, T., Schwoon, S., Jha, S., Melski, D.: Weighted pushdown systems and their application to interprocedural dataflow analysis. Science of Computer Programming (To appear)

[21] Bouajjani, A., Esparza, J., Touili, T.: A generic approach to the static analysis of concurrent programs with procedures. In: Proceedings of POPL'03. (2003)

[22] Esparza, J., Hansel, D., Rossmanith, P., Schwoon, S.: Efficient algorithms for model checking pushdown systems. In Emerson, E.A., Sistla, A.P., eds.: Proceedings of CAV'2000. Volume 1855 of Lecture Notes in Computer Science., Springer (2000) 232–247

[23] Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Application to model-checking. In: Proceedings of CONCUR'97. Volume 1243 of Lecture Notes in Computer Science., Springer (1997) 135–150