Computer Sciences Department

On Generalized Authorization Problems

S. Schwoon
Somesh Jha
Thomas Reps
S. Stubblebine

Technical Report #1469

January 2003

UNIVERSITY OF
WISCONSIN
MADISON

On Generalized Authorization Problems

S. Schwoon* S. Jha† T. Reps† S. Stubblebine‡

Abstract

This paper defines a framework in which one can formalize a variety of authorization and policy issues that arise in access control of shared computing resources. Instantiations of the framework address such issues as privacy, recency, validity, and trust. The paper presents an efficient algorithm for solving all authorization problems in the framework; this approach yields new algorithms for a number of specific authorization problems.

1 Introduction

The main issues in access control of shared computing resources are *authentication*, *authorization* and *enforcement*. Identification of principals is handled by authentication. Authorization addresses the following question: should a request r by a specific principal K be allowed? Enforcement addresses the problem of implementing the authorization during an execution. In a centralized system, authorization is based on the *closed-world assumption*, i.e., all authorized parties are known and trusted. In a distributed system where all the parties are not known a priori, the closed-world assumption is not applicable. Trust management systems [9] address the authorization problem in the context of distributed systems by requiring that authorization and access-control policies be defined explicitly, using an appropriate specification language, and relying on an algorithm to determine when a specific request is allowable. A survey of trust management systems, along with a formal framework for understanding them, is presented in [47]. Several trust management systems, such as Binder [17], Keynote [8], Referee [14], and SPKI/SDSI [18], have been proposed. Our work is presented in the context of SPKI/SDSI, but several aspects of the approach should carry over to other trust management systems and authorization frameworks.

In SPKI/SDSI, *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 fundamental problem in SPKI/SDSI (or any other trust management system) is the *authorization problem* (AP), which is defined as follows: given a security policy—which in SPKI/SDSI is represented by a set of name and authorization certificates—can a principal K access resource R?

Certificate-chain discovery refers to the problem of finding a "proof" that K can access resource R. (In the case of SPKI/SDSI, a proof is a chain of certificates.) If found, the proof can be presented by K to R. R checks the validity of the proof, and if the proof is valid, K is allowed access to R. Therefore, algorithms for certificate-chain discovery can also be used in frameworks such as *proof-carrying authorization* [3]. An efficient certificate-chain-discovery algorithm for SPKI/SDSI was presented by Clarke et al. [15]. An improved algorithm was presented by Jha and Reps [23]. The latter algorithm is based on

*Institutsverbund Informatik, Universität Stuttgart, Breitwiesenstr. 20–22, 70565 Stuttgart, Germany; E-mail: schwoosn@informatik.uni-stuttgart.de

†Comp. Sci. Dept., Univ. of Wisconsin, 1210 W. Dayton St., Madison, WI 53706. E-mail: {jha,reps}@cs.wisc.edu.

‡Stubblebine Research Labs, LLC 8 Wayne Blvd., Madison, NJ 07940. E-mail: stuart@stubblebine.com

translating SPKI/SDSI certificates to rules in a pushdown system. In [23] it was also demonstrated how this translation enables many other questions to be answered about a security policy expressed as a set of certificates.

In this paper, we generalize the pushdown-systems approach to enable it to address important security-policy issues such as privacy, recency, validity, and trust. For instance, consider the following authorization example: suppose that company X provides additional insurance to cover prescription-drug expenses that are not covered by a patient's health-maintenance organization (HMO). For example, the HMO might have a very high deductible for drugs, which will be covered by the additional insurance. However, company X only wants to provide this service to patients of a certain hospital H. For Alice to be able to buy insurance, she needs to prove to X that she is a patient of H. Suppose that there are two certificate chains that prove that Alice is a patient of H, where one reveals that Alice is a patient in the internal-medicine clinic and the other reveals that Alice is a patient in the AIDS clinic. For obvious reasons Alice will prefer to use the former chain. In other words, Alice prefers a certificate chain that *reveals the least amount of information about her*. Such privacy-related issues can be addressed in our generalized framework.

In the context of SPKI/SDSI, assume that we are given a metric μ on certificate chains, and hence on proofs of authorization. The details of the metric depend on the specific issue being addressed. In the *generalized authorization problem* (GAP) we are given a principal K, a set of name and authorization certificates $\mathcal{C}$, a resource R, and a metric μ on certificate chains. The question that GAP addresses is the same as AP—i.e., given $\mathcal{C}$, is K authorized to access resource R?—however, an authorization proof that solves a GAP minimizes or maximizes the given metric (depending on the application). We demonstrate that several security-policy issues in trust management systems can be cast as GAPs with appropriate metrics. In particular, we demonstrate how an extension of pushdown systems, called weighted pushdown systems, can be used to solve such generalized authorization problems.

The algorithm for solving GAPs can be thought of as a generalization of the certificate-chain-discovery algorithm. The general strategy is as follows: the set of labeled SPKI/SDSI certificates is first translated to a weighted pushdown system.[1] After the translation, the answer is obtained by solving a generalized shortest-path problem [26, 31, 35].

The main contributions of the work reported in the paper are as follows:

- **The GAP framework**. We define the generalized authorization problem and show how versions of several types of security issues related to authorization can be handled in the GAP framework.
- **An efficient algorithm for solving GAPs**. We present an efficient algorithm for solving GAPs. This yields several new algorithms for a number of specific authorization problems.
- **A prototype implementation**. MOPED [42] is a model checker for pushdown systems. We created an extended version of MOPED that handles weighted pushdown systems, and used it implement a prototype system that solves GAPs.

The remainder of the paper is organized as follows: Section 2 provides background on SPKI/SDSI. Section 3 defines the GAP framework and discusses several possible applications of it. Section 4 provides background on pushdown systems (PDSs). Section 5 reviews the connection between SPKI/SDSI and PDSs. Section 6 defines weighted PDSs, and shows how an analysis of the transition system defined by a weighted PDS can be used to solve GAPs. Section 7 returns to the discussion of applications of the GAP framework. Section 8 discusses related work. Appendix A describes an enhancement to the algorithm described in Section 6 to generate witnesses or proofs of authorization.

[1] In a GAP, each certificate is labeled with a value. However, a label might depend on the execution context. For example, for recency policies a certificate's value represents the time the certificate was issued, or last known to be current.

2 Background on SPKI/SDSI

2.1 Principals and Names

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 active 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.

2.2 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. A validity specification can also take the form of an on-line check to be performed.
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 same as in the case of a name cert.

We will treat certs as rewrite rules:[2]

- A name cert $(K, \mathtt{A}, S, V)$ will be written as K `A` $\longrightarrow S$.
- An auth cert (K, S, D, T, V) will be written as $K_T\ \square \longrightarrow S\ \square$ if the delegation bit D is turned on; otherwise, it will be written as $K_T\ \square \longrightarrow S\ \blacksquare$.

  The pair K, T of an auth cert refers to some resource R. Because we are primarily interested in questions about resources, rather than questions about either K or T individually, we generally write an auth cert as $R\ \square \longrightarrow S\ \square$ or $R\ \square \longrightarrow S\ \blacksquare$. Resources will be denoted by R, R_A, R_B, R', etc.

2.3 The Authorization Problem in SPKI/SDSI

In traditional discretionary access control, each protected resource has an associated access-control list, or ACL, describing which principals have various permissions to access the resource. An auth cert (K, S, D, T, V)

[2] In authorization problems, we only consider valid certificates, so the validity specification V for a certificate does not appear as part of its rewrite rule.

can be viewed as an ACL entry, where keys or principals represented by the subject S are given permission to access resource K_T.

A term S appearing in the rules 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 rewrite rule $L \longrightarrow R$ corresponding to a cert. Consider a term $S = LX$. In this case, the rewrite rule $L \longrightarrow R$ applied to the term S (denoted by $(L \longrightarrow R)(S)$) yields the term RX. Therefore, a rule can be viewed as a function from terms to terms, for example,

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

Consider two rules $c_1 = (L_1 \longrightarrow R_1)$ and $c_2 = (L_2 \longrightarrow R_2)$, 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 $L_1 \longrightarrow R_2X$. For example, consider the two rules:

$$c_1 : \quad K_A\ \texttt{friends} \longrightarrow K_A\ \texttt{Bob}\ \texttt{myFriends} \qquad\qquad c_2 : \quad K_A\ \texttt{Bob} \longrightarrow K_B$$

The composition $c_2 \circ c_1$ is $K_A\ \texttt{friends} \longrightarrow K_B\ \texttt{myFriends}$. Two rules c_1 and c_2 are called *compatible* if their composition $c_2 \circ c_1$ is well defined.[3]

A problem that often needs to be solved is the authorization question: "Given a set of certs $\mathcal{C}$ and a principal K, is K allowed to access resource R?" A *certificate-chain-discovery* algorithm provides more than just a simple yes/no answer to the authorization question; in the case of a yes answer, it identifies a chain of certificates $c_k \circ c_{k-1} \circ \cdots \circ c_1$ that proves that principal K is allowed to access R. Formally, certificate-chain discovery attempts to find a certificate chain $c_k \circ c_{k-1} \circ \cdots \circ c_1$ such that

$$(c_k \circ c_{k-1} \circ \cdots \circ c_1)(R\ \square) \in \{K\ \square, K\ \blacksquare\}\ .$$

Intuitively, $c_k \circ c_{k-1} \circ \cdots \circ c_1$ represents a path from $R\ \square$, which represents the resource, to either $K\ \square$ or $K\ \blacksquare$, which represent two forms of "permission for K to access": with and without delegation, respectively.

Clarke et.al [15] 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 $\mathcal{C}$. Jha and Reps [23] presented a different algorithm, based on the theory of pushdown systems.

3 The Generalized Authorization Problem

In this section, we formally define the *generalized authorization problem*, or *GAP*. Later in the section, we show that several issues, such as privacy, validity, recency, and trust, can be formulated in the GAP framework. In this framework, certificates are labeled with *weights* that are drawn from a bounded idempotent semiring.

Definition 3.1 *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 with* 0 *as its neutral element, and* $\oplus$ *is idempotent (i.e., for all* $a \in D$*,* $a \oplus a = a$*).*

[3]Note that 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.

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*

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) \qquad \textit{and} \qquad (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)\ .$$

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.*

A *weighted SPKI/SDSI system* $\mathcal{WSS}$ is a 3-tuple $(\mathcal{C}, \mathcal{S}, f)$, where $\mathcal{C}$ is a set of certs, $\mathcal{S} = (D, \oplus, \otimes, 0, 1)$ is a bounded idempotent semiring, and $f : \mathcal{C} \rightarrow D$ assigns weights to the certs in $\mathcal{C}$. We extend the function f to certificate chains in a natural way, i.e., given a certificate chain $c_k \circ c_{k-1} \cdots \circ c_1$, $f(c_k \circ c_{k-1} \circ \cdots \circ c_1)$ is defined as $f(c_1) \otimes \cdots \otimes f(c_{k-1}) \otimes f(c_k)$.

Definition 3.2 *Given a weighted SPKI/SDSI system* $\mathcal{WSS} = (\mathcal{C}, \mathcal{S}, f)$, *a principal* K, *and a resource* R_T, *proof*$(\mathcal{C}, K, R_T)$ *denotes the set of certificate chains that prove that* K *can access* R_T. *Formally, proof*$(\mathcal{C}, K, R_T)$ *is the set of certificate chains* $c_k \circ c_{k-1} \circ \cdots \circ c_1$ *such that:*

$$(c_k \circ c_{k-1} \circ \cdots c_1)(R_T\ \square) \in \{K\ \square, K\ \blacksquare\}$$

The generalized authorization problem *(GAP) asks the following two questions: (1) Is proof*$(\mathcal{C}, K, R_T)$ *non-empty? (2) If proof*$(\mathcal{C}, K, R_T)$ *is non-empty, then find the following two quantities:*

- $\delta := \bigoplus\{\, f(cc) \mid cc \in \textit{proof}(\mathcal{C}, K, R_T)\}$;
- *a* witness set *of certificate chains* $\omega \subseteq \textit{proof}(\mathcal{C}, K, R_T)$ *such that* $\bigoplus_{cc \in \omega} f(cc) = \delta$.

Notice that the extender operation $\otimes$ is used to calculate the value of a certificate chain. The value of a set of certificate chains is computed using the combiner operation $\oplus$. In general, it is enough for ω to contain only a finite set of minimal elements (i.e., minimal with respect to the partial order $\sqsubseteq$). Intuitively, GAP attempts to find a set of certificate chains proving that K can access R_T such that the combination (using the operator $\bigoplus$) of their weights is minimal.

We now demonstrate that several authorization-related problems can be cast in this framework.

Privacy-preserving certificate chains

We return to the example described in the Introduction, in which company X offers additional insurance to patients of a certain hospital H. The certificates relevant to the problem are shown in Figure 1. $K_X\ \square$ represents the service offered, i.e., the additional insurance offered by company X. The filled square represents the fact that this authorization cannot be delegated, e.g., an eligible patient cannot delegate the action of buying insurance to one of their friends. The principals corresponding to the clinics of AIDS treatment and internal medicine in hospital H are denoted by K_{H-AIDS} and K_{H-IM}. Alice is a patient in both clinics.

Suppose that Alice wants to buy the insurance. In this case, both (4) $\circ$ (2) $\circ$ (1) and (5) $\circ$ (3) $\circ$ (1) are equal to $K_X\ \square \longrightarrow K_{\texttt{Alice}}\ \blacksquare$. However, the certificate chain (4) $\circ$ (2) $\circ$ (1) reveals that Alice probably has AIDS, which is information that Alice may not wish to reveal to company X. Therefore, Alice would prefer to offer the certificate chain (5) $\circ$ (3) $\circ$ (1) to company X; it proves that she is authorized to buy additional insurance, but reveals the least amount of information about her.

Certificates		weights
K_X □ $\longrightarrow$ K_H patient ■	(1)	I
K_H patient $\longrightarrow$ K_{H-AIDS} patient	(2)	I
K_H patient $\longrightarrow$ K_{H-IM} patient	(3)	I
K_{H-AIDS} patient $\longrightarrow$ $K_{\texttt{Alice}}$	(4)	S
K_{H-IM} patient $\longrightarrow$ $K_{\texttt{Alice}}$	(5)	I

Figure 1: A set of weighted certificates.

Privacy can be modeled in the GAP framework using the semiring $(D, \oplus, \otimes, 0, 1)$, defined as follows: $D = \{I, S\}$, where I and S stand for "insensitive" and "sensitive", respectively. The 0 and 1 elements are S and I, respectively. The $\oplus$ and $\otimes$ operators are defined as follows (where x denotes either S or I):

$$I \oplus x = x \oplus I = I \quad \text{and} \quad S \oplus x = x \oplus S = x$$
$$S \otimes x = x \otimes S = S \quad \text{and} \quad I \otimes x = x \otimes I = x$$

It is easy to check that conditions 1−4 of Definition 3.1 are satisfied. Condition 5 is trivially satisfied because D is finite. The weights for the certificates are shown in Figure 1: certificate (4), K_{H-AIDS} patient $\longrightarrow$ $K_{\texttt{Alice}}$, is labeled S because it reveals that Alice is a patient in the AIDS clinic; all other certificates are labeled I. The weights of the certificate chain (4) ∘ (2) ∘ (1) and (5) ∘ (3) ∘ (1) are $I \otimes I \otimes S = S$ and $I \otimes I \otimes I = I$, respectively. Obviously, Alice prefers the certificate chain with weight I. In Section 6, we show how Alice can discover such a certificate chain.

Maximally-valid certificate chain. Let $V(c)$ be the expiration value of cert c, i.e., the cert c will expire at time $T_{current} + V(c)$, where $T_{current}$ is the current time. The expiration value of a certificate chain $c_k \circ c_{k-1} \circ \cdots \circ c_1$ is $\min_{i=1}^{k} V(c_i)$. Suppose that Alice wants to login to host H. If Alice provides a certificate chain that is only valid for two minutes, then she will be logged off by the host after two minutes. Thus, Alice wants to find a certificate chain that authorizes her to login to H, but has the maximum expiration value among all such certificate chains.

Most-recent certificate chain. Let $R(c)$ be the time (relative to the current time) when the cert c was issued or an on-line check was performed on cert c, i.e., $T_{current} - R(c)$ is the actual time of issue or the last on-line check. We call $R(c)$ the *recency* associated with cert c. The recency of a certificate chain $c_k \circ c_{k-1} \circ \cdots \circ c_1$ is equal to $\max_{i=1}^{k} R(c_i)$. Suppose that Alice wants to login to host H. For risk-reduction purposes, host H might mandate the use of a certificate chain whose recency is no more than ten minutes. In this case, Alice wishes to find a certificate chain that authorizes her to login to H and has the minimum recency among all such chains. Let $c_k \circ c_{k-1} \circ \cdots \circ c_1$ be the certificate chain with minimum recency. If $\max_{i=1}^{k} R(c_i)$ is less than or equal to ten minutes, then Alice can use the certificate chain to login to H.

Certificate chains with maximal trust

Assume that each certificate c is assigned a trust level $Tr(c)$ by the issuer of the certificate. Intuitively, $Tr(c)$ denotes the confidence that the issuer of c has in the relationship expressed by the certificate c. The trust level of a certificate chain $c_k \circ c_{k-1} \circ \cdots \circ c_1$ is $\bigotimes_{i=1}^{k} Tr(c_i)$, where $\bigotimes$ is defined in Table 1. Suppose that Alice wants to use server S, but S requires a certificate chain that has a trust level above a certain value v. In this case, Alice wants to find a certificate chain that authorizes her to use S, but has the maximal trust level among all such chains. If such a certificate chain has a trust level above v, Alice can use S.

Formalization using semirings. The semirings for the three cases discussed above are shown in Table 1. In the case of the maximal-trust example, the trust levels are drawn from a totally ordered set with four elements $\{N, L, M, H\}$, where $N \sqsupseteq L \sqsupseteq M \sqsupseteq H$. Elements L, M, and H denote low, medium, and high

	D	$\oplus$	$\otimes$	0	1
Validity	$\mathbb{N} \cup \{\pm\infty\}$	max	min	$-\infty$	$+\infty$
Recency	$\mathbb{N} \cup \{\infty\}$	min	max	∞	0
Trust	$\{N, L, M, H\}$	$\sqcap$	$\sqcup$	N	H

Table 1: Semirings for validity, recency, and trust.

levels of trust, respectively. The element N stands for "no link".[4] The join $\sqcup$ and the meet $\sqcap$ operator on this totally ordered set are defined as follows (where x and y are arbitrary elements of $\{N, L, M, H\}$):

$$x \sqcup y = \begin{cases} x & \text{if } x \sqsupseteq y \\ y & \text{otherwise} \end{cases} \qquad x \sqcap y = \begin{cases} y & \text{if } x \sqsupseteq y \\ x & \text{otherwise} \end{cases}$$

4 Pushdown Systems

A pushdown system is a transition system whose states involve a stack of unbounded length.

Definition 4.1 *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. A* configuration *of $\mathcal{P}$ is a pair $\langle p, w \rangle$, where $p \in P$ and $w \in \Gamma^*$. 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 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 also write $c \Rightarrow_{\mathcal{P}} c'$ to express that there is some rule r such that $c \xRightarrow{\langle r \rangle}_{\mathcal{P}} c'$, and we omit the index $\mathcal{P}$ if $\mathcal{P}$ is understood. The reflexive and transitive closure of $\Rightarrow$ is written $\Rightarrow^*$. Given a set of configurations C, we define $pre^*(C) := \{ c' \mid \exists c \in C : c' \Rightarrow^* c \}$ and $post^*(C) := \{ c' \mid \exists c \in C : c \Rightarrow^* c' \}$. to be the sets of configurations that are backwards and forwards reachable from elements of C, respectively.

Without loss of generality, we assume henceforth that for every $\langle p, \gamma \rangle \hookrightarrow \langle p', w \rangle$ 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., see [23].

Because pushdown systems have infinitely many configurations, we need some symbolic means to represent sets of configurations. We will use finite automata for this purpose.

Definition 4.2 *Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a pushdown system. A $\mathcal{P}$-automaton is a quintuple $\mathcal{A} = (Q, \Gamma, \rightarrow, P, F)$ where $Q \supseteq P$ is a finite set of* states*, $\rightarrow \subseteq Q \times \Gamma \times Q$ is the set of* transitions*, and $F \subseteq Q$ are the* final states*. The* initial states *of $\mathcal{A}$ are the control locations P. A configuration $\langle p, w \rangle$ is* accepted *by $\mathcal{A}$ if $p \xrightarrow{w}{}^* q$ for some final state q. A set of configurations of $\mathcal{P}$ is* regular *if it is recognized by some $\mathcal{P}$-automaton. (If $\mathcal{P}$ is understood, we omit the prefix $\mathcal{P}$ and merely refer to "automaton".)*

A convenient property of regular sets of configurations is that they are closed under forward and backward reachability. In other words, given an automaton $\mathcal{A}$ that accepts the set C, one can construct automata $\mathcal{A}_{pre^*}$ and $\mathcal{A}_{post^*}$ that accept $pre^*(C)$ and $post^*(C)$, respectively. The general idea behind the algorithm for pre^* [11, 19] is as follows:

[4] Note that "highest level of trust" is denoted by the element H, which is lowest in the total order.

$$
\begin{array}{|ll|l|}
\hline
\langle K_X, \square \rangle & \hookrightarrow \langle K_H, \texttt{patient}\ \blacksquare \rangle & (1) \\
\langle K_H, \texttt{patient} \rangle & \hookrightarrow \langle K_{H-AIDS}, \texttt{patient} \rangle & (2) \\
\langle K_H, \texttt{patient} \rangle & \hookrightarrow \langle K_{H-IM}, \texttt{patient} \rangle & (3) \\
\langle K_{H-AIDS}, \texttt{patient} \rangle & \hookrightarrow \langle K_{\texttt{Alice}}, \varepsilon \rangle & (4) \\
\langle K_{H-IM}, \texttt{patient} \rangle & \hookrightarrow \langle K_{\texttt{Alice}}, \varepsilon \rangle & (5) \\
\hline
\end{array}
$$

Figure 2: The PDS rules that correspond to Figure 1.

Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a pushdown system and $\mathcal{A} = (Q, \Gamma, \rightarrow_0, P, F)$ 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. $pre^*(C)$ is obtained as the language of an automaton $\mathcal{A}_{pre^*} = (Q, \Gamma, \rightarrow, P, F)$ derived from $\mathcal{A}$ by a saturation procedure. The procedure adds new transitions to $\mathcal{A}$ according to the following rule:

If $\langle p, \gamma \rangle \hookrightarrow \langle p', w \rangle$ and $p' \xrightarrow{w}{}^* q$ in the current automaton, add a transition (p, γ, q).

In [19] an efficient implementation of this procedure is given, which requires $\mathcal{O}(|Q|^2|\Delta|)$ time and $\mathcal{O}(|Q|\,|\Delta| + |\rightarrow_0|)$ space. Moreover, another procedure (and implementation) are presented for constructing a $\mathcal{P}$-automaton that accepts $post^*(C)$. In the following, we show that extensions of these procedures provide efficient algorithms for discovering the certificate chains needed in generalized authorization problems, such as those discussed in Section 3. We will present these extensions for pre^*; the same basic idea applies to $post^*$, but this is omitted for lack of space.

5 The Connection Between SPKI/SDSI and Pushdown Systems

The following correspondence between SPKI/SDSI and pushdown systems was presented in [23]: 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; with $\mathcal{C}$ we associate the pushdown system $\mathcal{P}_\mathcal{C} = (\mathcal{K}_\mathcal{C}, \mathcal{I}_\mathcal{C} \cup \{\square, \blacksquare\}, \Delta_\mathcal{C})$, 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 follows:

- if $\mathcal{C}$ contains a name cert $K\ \texttt{A} \longrightarrow K'\ \sigma$ (where σ is a sequence of identifiers), then $\Delta_\mathcal{C}$ contains a rule $\langle K, \texttt{A} \rangle \hookrightarrow \langle K', \sigma \rangle$;
- if $\mathcal{C}$ contains an auth cert $K\ \square \longrightarrow K'\ \sigma\ b$ (where $b \in \{\square, \blacksquare\}$), then $\Delta_\mathcal{C}$ contains a rule $\langle K, \square \rangle \hookrightarrow \langle K', \sigma b \rangle$.

For instance, consider the set of certificates $\mathcal{C}$ from Figure 1. The corresponding pushdown system $\mathcal{P}_\mathcal{C}$ has the control locations $\{K_X, K_H, K_{H-AIDS}, K_{H-IM}, K_{\texttt{Alice}}\}$, the stack alphabet $\{\texttt{patient}, \square, \blacksquare\}$, and the set of rules listed in Figure 2.

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 that, when applied to $K\ \sigma$, yield $K'\ \sigma'$. For instance, in the example above Alice can prove that she has the right to buy additional insurance because $\langle K_X, \square \rangle \Rightarrow^* \langle K_{\texttt{Alice}}, \blacksquare \rangle$. In the authorization problem, we are given a set of certs $\mathcal{C}$, a principal K, and resource R_T. In terms of the PDS $\mathcal{P}_\mathcal{C}$ that corresponds to certificate set $\mathcal{C}$, the authorization problem can be stated as follows: K should be granted access to resource R_T iff the condition $\langle R_T, \square \rangle \in pre^*(\{\langle K, \square \rangle, \langle K, \blacksquare \rangle\})$ holds. Thus, in the medical example, we wish to determine whether $\langle K_X, \square \rangle \in pre^*(S)$, where $S = \{\langle K_{\texttt{Alice}}, \square \rangle, \langle K_{\texttt{Alice}}, \blacksquare \rangle\}$. The automaton shown in Figure 3(a) accepts the set of configurations S. The set of predecessor configurations of the set S or $pre^*(S)$ is shown in Figure 3(b). Because there is a transition on the symbol $\square$ from state K_X to the

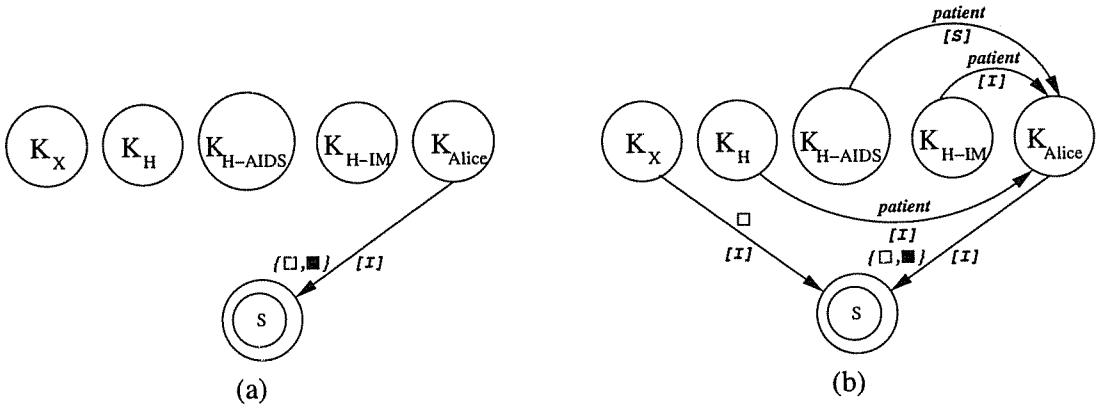

Figure 3: (a) Automaton representing the configurations $S = \{\langle K_{\texttt{Alice}}, \square\rangle, \langle K_{\texttt{Alice}}, \blacksquare\rangle\}$. (b) Automaton representing the configurations in $pre^*(S)$.

accepting state s, $\langle K_X, \square\rangle \in pre^*(S)$. In other words, Alice is authorized to buy additional insurance. (The extra annotations I (insensitive) and S (sensitive) on the transitions indicate whether the transitions involve sensitive information. The algorithm for deriving these labels is presented in Section 6.)

6 Solving the Generalized Authorization Problem

The types of problems treated in [23] could be characterized as having a qualitative nature; they answer questions such as "Is a given principal allowed to access a given resource?" In this section, we show how to answer questions that have an additional quantitative component, e.g. "How long is a given principal allowed to access a given resource?" To do so, we consider pushdown systems whose rules carry weights.

6.1 Weighted Pushdown Systems

We consider pushdown system whose rules are given values from some domain of weights. The weight domains of interest are the bounded idempotent semirings from Definition 3.1.

Definition 6.1 *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 $f : \Delta \to D$ 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 $v(\sigma) := f(r_1) \otimes \ldots \otimes f(r_k)$. 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., $c \overset{\langle r_1\rangle}{\Longrightarrow} \cdots \overset{\langle r_k\rangle}{\Longrightarrow} c'$.

Definition 6.2 *Given a weighted pushdown system $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$, where $\mathcal{P} = (P, \Gamma, \Delta)$, and a regular set $C \subseteq P \times \Gamma^*$, the* generalized pushdown reachability (GPR) problem *is to find for each $c \in P \times \Gamma^*$:*

- $\delta(c) := \bigoplus\{\, v(\sigma) \mid \sigma \in path(c, c'), c' \in C \,\}$;
- *a* witness set *of paths $\omega(c) \subseteq \bigcup_{c' \in C} path(c, c')$ such that $\bigoplus_{\sigma \in \omega(c)} v(\sigma) = \delta(c)$.*

In general, it is enough for $\omega(c)$ to contain only a finite set of paths whose values are minimal elements of $\{ v(\sigma) \mid \sigma \in path(c, c'),\ c' \in C \}$, i.e., minimal with respect to the partial order $\sqsubseteq$ defined in Definition 3.1(5).

For the remainder of this section, let $\mathcal{W}$ denote a fixed weighted pushdown system: $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$, where $\mathcal{P} = (P, \Gamma, \Delta)$ and $\mathcal{S} = (D, \oplus, \otimes, 0, 1)$; let C denote a fixed regular set of configurations, represented by a $\mathcal{P}$-automaton $\mathcal{A} = (Q, \Gamma, \rightarrow_0, P, F)$ such that $\mathcal{A}$ has no transition leading to an initial state.

The GPR problem is a multi-target meet-over-all-paths problem on a graph. The vertices of the graph are the configurations of $\mathcal{P}$, and the edges are defined by $\mathcal{P}$'s transition relation. The target vertices are the vertices in C. Both the graph and the set of target vertices can be infinite, but have some built-in structure to them; in particular, C is a regular set.

Because the GPR problem concerns infinite graphs, and not just an infinite set of paths, it differs from other work on meet-over-all-paths problems. As in the (ordinary) pushdown-reachability problem [11, 19], the infinite nature of the problem is addressed by reporting the answer in an indirect fashion, namely, in the form of an annotated automaton. An answer automaton without its annotations will be identical to an $\mathcal{A}_{pre^*}$ automaton created by the algorithm of [19]. For each $c \in pre^*(C)$, the values of $\delta(c)$ and $\omega(c)$ can be read off from the annotations by following all accepting paths for c in the automaton; for $c \notin pre^*(C)$, the values of $\delta(c)$ and $\omega(c)$ are 0 and $\emptyset$, respectively.

The solution to the GPR problem is presented in several stages:

- We first define a language that characterizes the sequences of transitions that can be made by a pushdown system $\mathcal{P}$ and automaton $\mathcal{A}$ for C.
- We then turn to weighted pushdown systems and the GPR problem. We use the language characterizations of transition sequences, together with previously known results on a certain kind of grammar problem [31, 35] to obtain a solution to the GPR problem.
- However, the solution based on grammars is somewhat inefficient; to improve the performance, we specialize the computation to our case, ending up with an algorithm for creating an annotated automaton that is quite similar to the pre^* algorithm from [19].

6.2 Languages that Characterize Transition Sequences

In this section, we make some definitions that will aid in reasoning about the set of paths that lead from a configuration c to configurations in a regular set C. We call this set the *reachability witnesses* for $c \in P \times \Gamma^*$ with respect to C: $ReachabilityWitnesses(c, C) = \bigcup_{c' \in C} path(c, c')$.

It is convenient to think of PDS $\mathcal{P}$ and automaton $\mathcal{A}$ (for C) as being combined in sequence, to create a combined PDS, which we will call $\mathcal{PA}$. $\mathcal{PA}$'s states are $P \cup Q = Q$, and its rules are those of $\mathcal{P}$, augmented with a rule $\langle q, \gamma \rangle \hookrightarrow \langle q', \epsilon \rangle$ for each transition $q \xrightarrow{\gamma} q'$ in $\mathcal{A}$'s transition set $\rightarrow_0$.

We say that a configuration $c = \langle p, \gamma_1 \gamma_2 \ldots \gamma_n \rangle$ is *accepted* by $\mathcal{PA}$ if there is a path to a configuration $\langle q_f, \epsilon \rangle$ such that $q_f \in F$. Note that because $\mathcal{A}$ has no transitions leading to initial states, $\mathcal{PA}$'s behavior during an accepting run can be divided into two phases—transitions during which $\mathcal{PA}$ mimics $\mathcal{P}$, followed by transitions during which $\mathcal{PA}$ mimics $\mathcal{A}$: once $\mathcal{PA}$ reaches a state in $(Q - P)$, it can only perform a sequence of pops, possibly reaching a state in F. If the run of $\mathcal{PA}$ does reach a state in F, in terms of the features of the original $\mathcal{P}$ and $\mathcal{A}$, the second phase corresponds to automaton $\mathcal{A}$ accepting some configuration c' that has been reached by $\mathcal{P}$, starting in configuration c. In other words, $\mathcal{PA}$ accepts a configuration c iff $c \in pre^*(C)$.

The first language that we define characterizes the *pop sequences* of $\mathcal{PA}$. A pop sequence for $q \in Q$, $\gamma \in \Gamma$, and $q' \in Q$ is a sequence of $\mathcal{PA}$'s transitions that, and (i) starts in a configuration $\langle q, \gamma \rangle$, and (ii) ends

	Production	for each
(1)	$PS_{(q,\gamma,q')} \rightarrow \epsilon$	$q \xrightarrow{\gamma} q' \in \rightarrow_0$
(2)	$PS_{(p,\gamma,p')} \rightarrow \epsilon$	$\langle p,\gamma\rangle \hookrightarrow \langle p',\varepsilon\rangle \in \Delta,\ p \in P$
(3)	$PS_{(p,\gamma,q)} \rightarrow PS_{(p',\gamma',q)}$	$\langle p,\gamma\rangle \hookrightarrow \langle p',\gamma'\rangle \in \Delta,\ p \in P,\ q \in Q$
(4)	$PS_{(p,\gamma,q)} \rightarrow PS_{(p',\gamma',q')}\ PS_{(q',\gamma'',q)}$	$\langle p,\gamma\rangle \hookrightarrow \langle p',\gamma'\gamma''\rangle \in \Delta,\ p \in P,\ q,q' \in Q$

Figure 4: A context-free language for the pop sequences of $\mathcal{PA}$, and the $\mathcal{PA}$ rules that correspond to each production.

in a configuration $\langle q', \varepsilon\rangle$. The family of pop sequences for a given q, γ, and q' can be characterized by the complete derivation trees [5] derived from nonterminal $PS_{(q,\gamma,q')}$, using the grammar shown in Figure 4.

Theorem 6.1 *PDS $\mathcal{PA}$ has a pop sequence for q, γ, and q' iff nonterminal $PS_{(q,\gamma,q')}$ of the grammar shown in Figure 4 has a complete derivation tree. Moreover, for each derivation tree with root $PS_{(q,\gamma,q')}$, a preorder listing of the derivation tree's production instances (where Figure 4 defines the correspondence between productions and PDS rules) gives a sequence of rules for a pop sequence for q, γ, and q'; and every such sequence of rules has a derivation tree with root $PS_{(q,\gamma,q')}$.*

Proof: [Sketch] To shrink the stack by removing the stack symbol on the left-hand side of each rule of $\mathcal{PA}$, there must be a transition sequence that removes each of the symbols that appear in the stack component of the rule's right-hand side. In other words, a pop sequence for the left-hand-side stack symbol must involve a pop sequence for each right-hand-side stack symbol.

The left-hand and right-hand sides of the productions in Figure 4 reflect the pop-sequence obligations incurred by the corresponding rule of $\mathcal{PA}$. □

To capture the set $\mathit{ReachabilityWitnesses}(\langle p, \gamma_1\gamma_2\ldots\gamma_n\rangle, C)$, where C is recognized by automaton $\mathcal{A}$, we define a context-free language given by the set of productions

$$\begin{aligned}
\mathit{Accepting}[\gamma_1\gamma_2\ldots\gamma_n]_{(p,q)} &\rightarrow PS_{(p,\gamma_1,q_1)}\ PS_{(q_1,\gamma_2,q_2)}\ \ldots\ PS_{(q_{n-1},\gamma_n,q)} \\
&\qquad \text{for each } q_i \in Q, \text{ for } 1 \le i \le n-1; \text{ and } q \in F \\
\mathit{Accepted}[\gamma_1\gamma_2\ldots\gamma_n]_{(p)} &\rightarrow \mathit{Accepting}[\gamma_1\gamma_2\ldots\gamma_n]_{(p,q)} \qquad \text{for each } q \in F
\end{aligned}$$

This language captures all ways in which PDS $\mathcal{PA}$ can accept $\langle p, \gamma_1\gamma_2\ldots\gamma_n\rangle$: the set of reachability witnesses for $\langle p, \gamma_1\gamma_2\ldots\gamma_n\rangle$ corresponds to the complete derivation trees derivable from nonterminal $\mathit{Accepted}[\gamma_1\gamma_2\ldots\gamma_n]_{(p)}$. The subtree rooted at $PS_{(q_{i-1},\gamma_i,q_i)}$ gives the pop sequence that $\mathcal{PA}$ performs to consume symbol γ_i. (If there are no reachability witnesses for $\langle p, \gamma_1\gamma_2\ldots\gamma_n\rangle$, there are no complete derivation trees with root $\mathit{Accepted}[\gamma_1\gamma_2\ldots\gamma_n]_{(p)}$.)

6.3 Weighted PDSs and Abstract Grammar Problems

Turning now to weighted PDSs, we will consider the weighted version of $\mathcal{PA}$, denoted by $\mathcal{WA}$, in which weighted PDS $\mathcal{W}$ is combined with $\mathcal{A}$, and each rule $\langle q,\gamma\rangle \hookrightarrow \langle q',\epsilon\rangle$ that was added due to transition $q \xrightarrow{\gamma} q'$ in $\mathcal{A}$'s transition set $\rightarrow_0$ is assigned the weight 1.

We are able to reason about semiring sums ($\oplus$) of weights on the paths that are characterized by the context-free grammars defined above using the following concept:

[5] A derivation tree is *complete* if it has is a terminal symbol at each leaf.

	Production	for each
(1)	$PS_{(q,\gamma,q')} \rightarrow g_1(\epsilon)$ $g_1 = 1$	$(q,\gamma,q') \in \rightarrow_0$
(2)	$PS_{(p,\gamma,p')} \rightarrow g_2(\epsilon)$ $g_2 = f(r)$	$r = \langle p,\gamma\rangle \hookrightarrow \langle p',\varepsilon\rangle \in \Delta,\ p \in P$
(3)	$PS_{(p,\gamma,q)} \rightarrow g_3(PS_{(p',\gamma',q)})$ $g_3 = \lambda a.f(r) \otimes a$	$r = \langle p,\gamma\rangle \hookrightarrow \langle p',\gamma'\rangle \in \Delta,\ p \in P,\ q \in Q$
(4)	$PS_{(p,\gamma,q)} \rightarrow g_4(PS_{(p',\gamma',q')}, PS_{(q',\gamma'',q)})$ $g_4 = \lambda a.\lambda b.f(r) \otimes a \otimes b$	$r = \langle p,\gamma\rangle \hookrightarrow \langle p',\gamma'\gamma''\rangle \in \Delta,\ p \in P,\ q,q' \in Q$
(5)	$Accepting[\gamma_1\gamma_2\ldots\gamma_n]_{(p,q)} \rightarrow g_5(PS_{(p,\gamma_1,q_1)}, PS_{(q_1,\gamma_2,q_2)}, \ldots, PS_{(q_{n-1},\gamma_n,q)})$ $g_5 = \lambda a_1.\lambda a_2\ldots\lambda a_n.a_1 \otimes a_2 \otimes \ldots \otimes a_n$	$q_i \in Q$, for $1 \le i \le n-1$, and $q \in F$
(6)	$Accepted[\gamma_1\gamma_2\ldots\gamma_n]_{(p)} \rightarrow g_6(Accepting[\gamma_1\gamma_2\ldots\gamma_n]_{(p,q)})$ $g_6 = \lambda a.a$	$q \in F$

Figure 5: An abstract grammar problem for the GPR problem.

Definition 6.3 *[35] Let $(S,\sqcap)$ be a semilattice. An* abstract grammar *over $(S,\sqcap)$ is a collection of context-free grammar productions, where each production θ has the form*

$$X_0 \rightarrow g_\theta(X_1,\ldots,X_k).$$

Parentheses, commas, and g_θ (where θ is a production) are terminal symbols. Every production θ is associated with a function $g_\theta\colon S^k \rightarrow S$. Thus, every string α of terminal symbols derived in this grammar (i.e., the yield of a complete derivation tree) denotes a composition of functions, and corresponds to a unique value in S, which we call $val_G(\alpha)$ (or simply $val(\alpha)$ when G is understood). Let $L_G(X)$ denote the strings of terminals derivable from a nonterminal X. The abstract grammar problem *is to compute, for each nonterminal X, the value*

$$m_G(X) := \bigsqcap_{\alpha \in L_G(X)} val_G(\alpha).$$

Because the complete derivation trees with root $Accepted[\gamma_1\gamma_2\ldots\gamma_n]_{(p)}$ encode the transition sequences by which $\mathcal{WA}$ accepts $\langle p,\gamma_1\gamma_2\ldots\gamma_n\rangle$, to cast the GPR as a grammar problem, we merely have to attach appropriate production functions to the productions so that for each rule sequence σ, and corresponding derivation tree (with yield) α, we have $v(\sigma) = val_G(\alpha)$. This is done in Figure 5: note how functions g_3, g_4, and g_5 place $f(r)$ at the beginning of the semiring-product expression; this corresponds to a preorder listing of a derivation tree's production instances (cf. Theorem 6.1).

To solve the GPR problem, we appeal to the following theorem:

Theorem 6.2 *[35] The abstract grammar problem for G and $(S,\sqcap)$ can be solved by an iterative computation that finds the maximum fixed point when the following conditions hold:*

1. *The semilattice $(S,\sqcap)$ has no infinite descending chains.*
2. *Every production function g_θ in G is distributive, i.e.,*

$$g(\bigsqcap_{i_1 \in I_1}, \ldots, \bigsqcap_{i_k \in I_k}) = \bigsqcap_{(i_1,\ldots,i_k) \in I_1 \times \cdots \times I_k} g(x_{i_1},\ldots,x_{i_k})$$

   for arbitrary, non-empty, finite index sets $I_1,\ldots,I_k$.

3. *Every production function g_θ in G is strict in 0 in each argument.*

The abstract grammar problem given in Figure 5 meets the conditions of Theorem 6.2 because

1. By Definition 3.1, the $\oplus$ operator is associative, commutative, and idempotent; hence $(D, \oplus)$ is a semilattice. By Definition 3.1(5), $(D, \oplus)$ has no infinite descending chains.

2. The distributivity of each of the production functions $g_1, \ldots, g_6$ over arbitrary, non-empty, finite index sets follows from repeated application of Definition 3.1(3).

3. Production functions $g_3, \ldots, g_6$ are strict in 0 in each argument because 0 is an annihilator with respect to $\otimes$ (Definition 3.1(4)). Production functions g_1 and g_2 are constants (i.e., functions with no arguments), and hence meet the required condition trivially.

Thus, one algorithm for solving the GPR problem for a given weighted PDS $\mathcal{W}$, initial configuration $\langle p, \gamma_1\gamma_2 \ldots \gamma_n \rangle$, and regular set C (represented by automaton $\mathcal{A}$) is as follows:

- Create the combined weighted PDS $\mathcal{WA}$.
- Define the corresponding abstract grammar problem according to the schema shown in Figure 5.
- Solve this abstract grammar problem by finding the maximum fixed point using chaotic iteration: for each nonterminal X, the fixed-point-finding algorithm maintains a value $l(X)$, which is the current estimate for X's value in the maximum fixed-point solution; initially, all $l(X)$ values are set to 0; $l(X)$ is updated whenever a value $l(Y)$ changes, for any Y used on the right-hand side of a production whose left-hand-side nonterminal is X.

6.4 A More Efficient Algorithm for the GPR Problem

The approach given in the previous section is not very efficient: for a configuration $\langle p, \gamma_1\gamma_2 \ldots \gamma_n \rangle$, it takes $\Theta(|Q|^{n-1}|F|)$ time and space just to create the grammar productions in Figure 5 with left-hand-side nonterminal $Accepting[\gamma_1\gamma_2 \ldots \gamma_n]_{(p,q)}$. However, we can improve on the algorithm of the previous section because not all instantiations of the productions listed in Figure 5 are relevant to the final solution; we want to prevent the algorithm from exploring useless nonterminals of the grammar shown in Figure 5.

Moreover, all GPR questions with respect to a given target-configuration set C involve the same subgrammar for the *PS* nonterminals. As in the (ordinary) pushdown-reachability problem [11, 19], the information about whether a complete derivation tree with root nonterminal $PS_{(q,\gamma,q')}$ exists (i.e., whether $PS_{(q,\gamma,q')}$ is a *productive* nonterminal) can be precomputed and returned in the form of an (annotated) automaton of size $\mathcal{O}(|Q|\,|\Delta| + |{\rightarrow_0}|)$. Exploring the *PS* subgrammar lazily saves us from having to construct the entire *PS* subgrammar. Productive nonterminals represent automaton transitions, and the productions that involve any given transition can be constructed on-the-fly, as is done in Algorithm 1, shown in Figure 6.

It is relatively straightforward to see that Algorithm 1 solves the grammar problem for the *PS* subgrammar from Figure 5: *workset* contains the set of transitions (*PS* nonterminals) whose value $l(t)$ has been updated since it was last considered; in line 8 all values are set to 0. A function call $update(t, r, T)$ computes the new value for transition t if t can be created using rule r and the transitions in the ordered list T. Lines 9 and 10 process the rules of types (1) and (2), respectively. Lines 11–17 represent the fixed-point-finding loop: lines 13, 15, and 17 simulate the processing of rules of types (3) and (4) that involve transition t on their right-hand side; in particular, line 4 corresponds to invocations of production functions g_3 and g_4. Note that line 4 can change $l(t)$ only to a smaller value (w.r.t. $\sqsubseteq$). The iterations continue until the values of all transitions stabilize, i.e., *workset* is empty.

From the fact that Algorithm 1 is simply a different way of expressing the grammar problem for the *PS* subgrammar, we know that the algorithm terminates and computes the desired result. Moreover, apart from

Algorithm 1
Input: a weighted pushdown system $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$
where $\mathcal{P} = (P, \Gamma, \Delta)$ and $\mathcal{S} = (D, \oplus, \otimes, 0, 1)$;
a $\mathcal{P}$-Automaton $\mathcal{A} = (Q, \Gamma, \rightarrow_0, P, F)$ that accepts C such that $\mathcal{A}$ has no transitions into P states

Output: a $\mathcal{P}$-automaton $\mathcal{A}_{pre^*} = (Q, \Gamma, \rightarrow, P, F)$ that accepts $pre^*(C)$
a function l that maps every $(q, \gamma, q') \in \rightarrow$ to the value of $m_G(PS_{(q,\gamma,q')})$
in the abstract grammar problem defined in Figure 5;

```
1  procedure update(t, r, T)
2  begin
3     → := → ∪ {t};
4     l(t) := l(t) ⊕ (f(r) ⊗ l(T(1)) ⊗ ... ⊗ l(T(|T|)));
5     if l(t) changed value then workset := workset ∪ {t}
6  end
7
8  → := →0; l ≡ 0; workset := →0;
9  for all t ∈ →0 do l(t) := 1;
10 for all r = ⟨p, γ⟩ ↪ ⟨p', ε⟩ ∈ Δ do update((p, γ, p'), r, ());
11 while workset ≠ ∅ do
12    select and remove a transition t = (q, γ, q') from workset;
13    for all r = ⟨p1, γ1⟩ ↪ ⟨q, γ⟩ ∈ Δ do update((p1, γ1, q'), r, (t));
14    for all r = ⟨p1, γ1⟩ ↪ ⟨q, γγ2⟩ ∈ Δ do
15       for all t' = (q', γ2, q'') ∈ → do update((p1, γ1, q''), r, (t, t'));
16    for all r = ⟨p1, γ1⟩ ↪ ⟨p', γ2γ⟩ ∈ Δ do
17       if t' = (p', γ2, q) ∈ → then update((p1, γ1, q'), r, (t', t));
18 return ((Q, Γ, →, P, F), l)
```

Figure 6: An on-the-fly algorithm for solving the grammar problem for the *PS* subgrammar from Figure 5.

operations having to do with l, the algorithm is remarkably similar to the pre^* algorithm from [19]—the only major difference being that transitions are stored in a workset and processed multiple times, whereas in [19] each transition is processed exactly once. Thus, the time complexity increases from the $\mathcal{O}(|Q|^2|\Delta|)$ complexity of the unweighted case [19] by a factor that is no more than the length of the maximal-length descending chain in the semiring. (More efficient techniques that apply to certain semirings that are total orders are discussed in Section 6.5.)

Given the annotated pre^* automaton, the value of $\delta(c)$ for any configuration c can be read off from the automaton by following all paths by which c is accepted—accumulating a value for each path—and taking the meet of the resulting value set. The value-accumulation step can be performed using a straightforward extension of a standard algorithm for simulating an NFA (cf. [1, Algorithm 3.4]).

Algorithm 1 is a dynamic-programming algorithm for determining $\delta(c)$; Appendix A describes how to extend Algorithm 1 to keep additional annotations on transitions so that the path set $\omega(c)$ can be obtained.

6.5 Total Orderings

In the examples given in Section 3, the semirings all have the following properties: (i) the ordering $\sqsubseteq$ is a total ordering; (ii) 1 is the least element with respect to $\sqsubseteq$; and (iii) for all $a, b \in D$, $a \otimes b \sqsupseteq \text{lub}(a, b)$ (where lub denotes "least upper bound", or maximum, in the total order). In such cases, there is a much more

efficient algorithm for the GPR problem based on ideas from Knuth's generalization of Dijkstra's algorithm for the shortest-path problem [26].[6]

- In Algorithm 1, *workset* is implemented using a priority queue, and the transition selected in line 12 is always one with minimum value. With this approach, the transitions processed form a non-decreasing sequence; hence, no transition is selected from *workset* more than once. (In the general case, the label of a transition may change even if the transition has been selected before, causing it to be added to *workset* again.) Compared to the PDS-reachability problem for the unweighted case, all it costs to compute the maximum fixed-point values is the cost of maintaining a priority queue. Thus, the time complexity increases by a factor of $\mathcal{O}(\log(|Q|\,|\Delta| + |{\rightarrow_0}|))$
- The set $\omega(c)$ contains exactly one path.

7 Discussion

We now discuss several issues that arise in applying the GAP framework.

Recency Policies. The recency metric presented in Section 3 is rather simplistic compared to some others that have been studied: recency policies can be based on a number of factors, such as the financial risk of the authentication/authorization decision [44], semantics and invalidity rate of the certificate contents, and the security of the system used to generate the certificate. In a realistic setting, recency values of certificates need to be normalized. One possibility is to base the normalization on the remaining lifetime of the certificate (assuming the "not after" times in the validity specification were appropriately chosen). Let the lifetime of a certificate be $L = T_{not\_after} - T_{current}$ (provided the certificate is still valid, i.e., $T_{current}$ is before T_{not_after}), and let the recency of a certificate c_i be defined by $\frac{T_{current} - T_{issue}}{L(c_i)}$. In this case, the semiring for recency is $(\mathbb{R}_{\geq 0} \cup \{\infty\}, \min, \max, \infty, 0)$.[7]

Multiple Security Policies. Authorization policies may be subject to multiple security policies. For example, we might wish to satisfy simultaneously a most-recent certificate-chain policy and a privacy-preserving policy. One approach is the *policy-priority* approach, in which the user declares the order of security-policy priorities; for instance, privacy may be the first priority and recency the second priority. Such problems can be addressed in the GAP framework, when the component policies involve total orders, by using pairs of values as semiring values—e.g., (privacy, recency) values—and defining $\oplus$ to be lexicographic minimum [40, Section 6.4.1]. The GAP framework can also handle partially ordered component policies, as well as the situation where there is no clear preference among component policies [40, Section 6.4.1].

Trust Policies. Several *trust policies or metrics* have been proposed in the literature, such as [7, 30, 36, 37, 48]. Not all trust metrics can be efficiently modeled in the GAP framework. For example, consider the proposed Bounded Disjoint Paths (BDP) and Bounded Connective Paths metric, which are are NP-hard and coNP-hard, respectively [36]. Thus, there is little hope of finding an efficient solution to these problems. We have not investigated whether the approximation algorithms [33, 36] developed for these problems are applicable in our setting. Similarly, the minimum-capacity-cut metric [37] cannot be easily formulated in our framework. Because BDP and weighted shortest paths are both interesting metrics in the certificate-chain

[6] The approach that we describe also applies to a slightly larger class of totally ordered sets studied by Ramalingam [35]; however, our examples all fall into the class defined above, which was studied by Knuth [26].

[7] $\mathbb{R}_{\geq 0} \cup \{\infty\}$ has infinite descending chains; however, the only operations performed are min and max, and hence only a finite number of values ever arise in any execution. Consequently, the GAP framework still applies.

context, one might consider trying to use a metric of weighted-disjoint-bounded paths for certificate-chain evaluation. However, the weighted-disjoint-bounded-paths problem has been shown to be NP-complete for length bounds greater than 5, and approximation algorithms are NP-hard [10].

8 Related Work

A certificate-chain-discovery algorithm for SPKI/SDSI was first proposed by Clarke et al [15]. A credential-chain-discovery algorithm for the role-based trust management language RT_0 was presented by N. Li et al. [29]. In the proof-carrying-authorization (PCA) framework of Appel and Felten [3], a client uses the theorem prover *Twelf* [34] to construct a proof of authorization, which the client presents to the server. To the best of our knowledge, no one has previously considered issues such as privacy and trust in the context of certificate-chain-discovery algorithms for trust management systems or authorization-proof-construction algorithms for PCA. Our algorithm is based on an algorithm for a generalized shortest-path problem in which weights on edges are drawn from a semiring. This approach is quite general, and it is likely that this approach applies to other formalisms besides SPKI/SDSI.

Pushdown systems are related to "unrestricted hierarchical state machines", which are collections of finite-state transition systems connected by call and return transitions [2, 6]. They are also related to the "interprocedural control-flow graphs" [43] and "exploded supergraphs" [38] used in interprocedural dataflow analysis. Thus, dataflow analysis is one possible application of weighted PDSs. The algorithm for solving GPR problems developed in Section 6.4 is related to certain existing dataflow-analysis algorithms [43, 25, 41]. In particular, Sagiv et al. showed how to compute meet-over-all(-valid)-paths values for multi-entry/multi-exit hierarchically structured graphs [41]. However, with respect to previous work on interprocedural dataflow analysis, Section 6 makes two contributions:

- Conventional dataflow-analysis algorithms merge together the values for all configurations with the same top-of-stack symbol. With weighted PDSs, dataflow queries can be posed with respect to a regular language of initial stack configurations. This provides a strict generalization of the kind of answers obtainable via ordinary interprocedural dataflow-analysis algorithms.

- Because the algorithm for solving GPR problems can provide a witness set of paths, one can provide a client of the analysis algorithm with an explanation of why the answer to a dataflow query has the value reported.

Model checking of pushdown systems has also been used for verifying security properties of programs [20, 22, 13]. Thus, another application of weighted pushdown systems is for verifying security properties of programs, where the verification process requires knowing interprocedural dataflow information.

A number of *trust policies* or *metrics* have been proposed to obtain assurance on a certificate binding. The most well-known notions stem from PGP [48] where each user acts as a certificate authority by creating certificates for entities they trust. In a transitive manner, other certificate authorities (or "recommendors") introduce new certificate authorities they trust by creating other certificates. Assurance through this certificate-chaining process is provided, in part, by independent certificate paths [48]. Subsequent work studies network connectivity as another trust metric [36]. Other work studies metrics based on confidence valuations [7, 27, 30], minimum-capacity cuts on certificated edges that represent financial liabilities [37], and an algebra for assessing trust in certificate chains [24].

Private or sensitive information may reside within certificates. This may include names, roles, and/or other identifying information. Furthermore, chains of authorization certificates tend to mirror organization structures, business processes, and personal relations, which may also be sensitive [4]. The principal making the authorization request may follow a privacy policy in order to control what information is disclosed or

leaked as part of the authorization process. Some flexibility may exist so that the requester can choose from an alternative set of credentials that may be supplied as part of the proof of authorization. The certificate privacy problem is related to the long history of work on information flow based on a lattice model [5, 16], which attempts to model controls on the flow of information. Traditional information-flow policies stemming from the military [32] are concerned with information-disclosure policies under which access to data requires a proper clearance (mandatory access control) and a need to know (discretionary access control). We can draw from this work in the sense that our willingness to provide credentials with certain categories of information are subject to the current "discretionary" access request. Furthermore, policies may be based on the Chinese-Wall security policy [12] under which access to data is not constrained by attributes of the data in question but by the data to which the subject already holds access rights. However, the objective of the current paper has been to demonstrate a simple privacy metric that quantifies information flow for a certificate chain.

Validity time periods have been included in certificate formats since the early certificate standards [46]. The validity of the certificate contents is suspect if the current time is not within the certificate-validity period. Certificate-revocation lists or directories can be queried to determine if the credentials are known to be invalid. Stubblebine [44] formalizes the notion of recent-secure authentication as a means for authenticating a channel subject to freshness constraints. That work provides a means for reasoning about recent-secure authentication by extending a calculus of authentication [28]. Rivest further develops the case for flexible mechanisms that support authentication subject to recency constraints [39]. Additional recency policies and methods of analysis for recent-secure authentication were further developed in a work that provides a monotonic logic for reasoning about synchronization, revocation, and recency [45]. Other monotonic logics for reasoning about validity intervals in the SPKI context have also been studied [21].

References

[1] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1985.

[2] R. Alur, K. Etessami, and M. Yannakakis. Analysis of recursive state machines. In *Proc. Computer-Aided Verif.*, July 2001.

[3] A.W. Appel and E.W. Felten. Proof-carrying authentication. In *Conf. on Comp. and Commun. Sec.*, November 1999.

[4] T. Aura and C. Ellison. Privacy and accountability in certificate systems. Res. Rep. A61, Helsinki Univ. of Tech., Espoo, Finland, April 2000.

[5] D.E. Bell and L.J. LaPadula. Secure computer system: Unified exposition and Multics interpretation. Technical Report ESD-TR-75-306, MITRE Corp. MTR-2997, Bedford, MA. Available as NTIS AD-A023 588.

[6] M. Benedikt, P. Godefroid, and T. Reps. Model checking of unrestricted hierarchical state machines. In *ICALP '01*, 2001.

[7] T. Beth, M. Borcherding, and B. Klein. Valuation of trust in open networks. In *Computer Security — ESORICS '94*, volume 875 of *Lec. Notes in Comp. Sci.*, pages 3–18. Springer-Verlag, 1994.

[8] M. Blaze, J. Feigenbaum, J. Ioannidis, and A.D. Keromytis. The KeyNote trust-management system version 2. RFC 2704, September 1999.

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

[10] A. Bley. On the complexity of vertex-disjoint length restricted path problems. Tech. Rep. SC-98-20, Konrad-Zuse-Zentrum für Informationstechnik, Berlin, Ger., 1998.

[11] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model checking. In *Proc. CONCUR*, volume 1243 of *Lec. Notes in Comp. Sci.*, pages 135–150. Springer-Verlag, 1997.

[12] D.F.C. Brewer and M.J. Nash. The Chinese wall security policy. In *Symp. on Sec. and Privacy*, pages 206–214, 1989.

[13] H. Chen and D. Wagner. MOPS: An infrastructure for examining security properties of software. In *Conf. on Comp. and Commun. Sec.*, November 2002.

[14] Y.-H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss. REFEREE: Trust management for Web applications. *Computer Networks and ISDN Systems*, 29(8–13):953–964, September 1997.

[15] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R.L. Rivest. Certificate chain discovery in SPKI/SDSI. *JCS*, 9, 2001.

[16] D.E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5), 1976.

[17] J. DeTreville. Binder, a logic-based security language. In *Symp. on Res. in Sec. and Privacy*, Oakland, CA, May 2002. IEEE Computer Society Press.

[18] C.M. Ellison, B. Frantz, B. Lampson, R. L. Rivest, B.M. Thomas, and T. Ylonen. SPKI certificate theory. RFC 2693, September 1999.

[19] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Proc. Computer-Aided Verif.*, volume 1855 of *Lec. Notes in Comp. Sci.*, pages 232–247, July 2000.

[20] J. Esparza, A. Kučera, and S. Schwoon. Model-checking LTL with regular valuations for pushdown systems. In *Proceedings of TACAS'01*, volume 2031 of *LNCS*, pages 306–339. Springer, 2001.

[21] J.Y. Halpern and R. van der Meyden. A logical reconstruction of SPKI. In *Comp. Sec. Found. Workshop*, pages 59–70, 2001.

[22] T. Jensen, D. Le Metayer, and T. Thorn. Verification of control flow based security properties. In *1999 IEEE Symposium on Security and Privacy*, May 1999.

[23] S. Jha and T. Reps. Analysis of SPKI/SDSI certificates using model checking. In *IEEE Comp. Sec. Found. Workshop (CSFW)*. IEEE Computer Society Press, 2002.

[24] A. Jsang. An algebra for assessing trust in certification chains. In J. Kochmar, editor, *Proceedings of the Network and Distributed Systems Security Symposium (NDSS'99)*, 1999.

[25] J. Knoop and B. Steffen. The interprocedural coincidence theorem. In *Int. Conf. on Comp. Construct.*, pages 125–140, 1992.

[26] D.E. Knuth. A generalization of Dijkstra's algorithm. *Inf. Proc. Let.*, 5(1):1–5, 1977.

[27] R. Kohlas and U.M. Maurer. Confidence valuation in a public-key infrastructure based on uncertain evidence. In *Public Key Cryptography*, pages 93–112, 2000.

[28] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *Trans. on Comp. Syst,*, 10(4):265–310, 1992.

[29] N. Li, W.H. Winsborough, and J.C. Mitchell. Distributed credential chain discovery in trust management. To appear in *J. Comp. Sec.*.

[30] U. Maurer. Modeling a public-key infrastructure. In *Computer Security — ESORICS '96*, volume 1146 of *Lec. Notes in Comp. Sci.* Springer-Verlag, 1996.

[31] U. Moencke and R. Wilhelm. Grammar flow analysis. In H. Alblas and B. Melichar, editors, *Attribute Grammars, Applications and Systems*, volume 545 of *Lec. Notes in Comp. Sci.*, pages 151–186, Prague, Czechoslovakia, June 1991. Springer-Verlag.

[32] US Department of Defense. *DoD Trusted Computer System Evaluation Criteria (DOD 5200.28-STD)*. 1985.

[33] Y. Perl and D. Ronen. Heuristics for finding a maximum number of disjoint bounded paths. *Networks*, 14:531–544, 1984.

[34] 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.

[35] G. Ramalingam. *Bounded Incremental Computation*, volume 1089 of *Lec. Notes in Comp. Sci.* Springer-Verlag, 1996.

[36] M. Reiter and S. Stubblebine. Resilient authentication using path independence. *Trans. on Computers*, 47(12):1351–1362, December 1998.

[37] M. Reiter and S. Stubblebine. Authentication metric analysis and design. *Trans. on Inf. and Syst. Sec.*, 2(2), May 1999.

[38] T. Reps, S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *Symp. on Princ. of Prog. Lang.*, pages 49–61, New York, NY, 1995. ACM Press.

[39] R.L. Rivest. Can we eliminate certificate revocation lists? In R. Hirschfeld, editor, *Financial Cryptography*, pages 178–183, February 1998.

[40] G. Rote. Path problems in graphs. In *G. Tinhofer, E. Mayr, H. Noltemeier, and M.M. Syslo (eds.) in cooperation with R. Albrecht, Computational Graphs Theory, Springer-Verlag Computing Supplementum 7*. 1990.

[41] M. Sagiv, T. Reps, and S. Horwitz. Precise interprocedural dataflow analysis with applications to constant propagation. *Theor. Comp. Sci.*, 167:131–170, 1996.

[42] S. Schwoon. Moped – A Model-Checker for Pushdown Systems, 2002. http://www7.in.tum.de/~schwoon/moped.

[43] M. Sharir and A. Pnueli. Two approaches to interprocedural data flow analysis. In S.S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 7, pages 189–234. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[44] S. Stubblebine. Recent-secure authentication: Enforcing revocation in distributed systems. In *Symp. on Research in Sec. and Privacy*, pages 224–234, May 1995.

[45] S. Stubblebine and R. Wright. An authentication logic with formal semantics supporting synchronization, revocation, and recency. *Trans. on Softw. Eng.*, 28(3), March 2002.

[46] International Telecommunications Union. ITU-T recommendation X.509 (08/97) – information technology – open systems interconnection – the directory: Authentication framework, August 1997.

[47] S. Weeks. Understanding trust management systems. In *Symp. on Res. in Sec. and Privacy*, Oakland, CA, May 2001. IEEE Computer Society Press.

[48] P. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, 1995.

Appendix

A Generation of Witness Sets

Section 6.4 gives an efficient algorithm for determining $\delta(c)$; this section addresses the question of how to obtain $\omega(c)$. It may help to think of this problem as that of examining an infinite graph $\mathcal{G}$ whose nodes are pairs (c, d), where c is a configuration and d a value from D, and in which there is an edge from (c_1, d_1) to (c_2, d_2) labeled with $r \in \Delta$ if and only if $c_1 \stackrel{\langle r \rangle}{\Longrightarrow} c_2$ and $f(r) \otimes d_2 = d_1$. For a given configuration c, finding $\omega(c)$ means identifying a set of paths $\sigma_1, \ldots, \sigma_k$ such that path σ_i, $1 \leq i \leq k$, leads from some (c, d_i) to some $(c_i, 1)$, where $c_i \in C$, and $\bigoplus_{i=1}^{k} d_i = \delta(c)$. In other words, $\omega(c) = \{\sigma_1, \ldots, \sigma_k\}$ proves that $\delta(c)$ really has the value computed by Algorithm 1. We note the following properties:

- In general, k may be larger than 1, e.g., we might have a situation where $\delta(c) = d_1 \oplus d_2$ because of two paths with values d_1 and d_2, but there may be no single path with value $d_1 \oplus d_2$.
- We want to keep $\omega(c)$ as small as possible. If a witness set contains two paths σ_1 and σ_2, where $v(\sigma_1) \sqsubseteq v(\sigma_2)$, then the same set without σ_2 is still a witness set.

Algorithm 2

```
1  procedure update(t, r, T)
2  begin
3     → := → ∪ {t};
4     d := f(r) ⊗ l(T(1)) ⊗ ... ⊗ l(T(|T|));
5     s := (d, [t], [r], ( n(t') | t' ∈ T ));
6     if l(t) ⊑ d then return;
7     if n(t) = nil or d ⊏ l(t) then
8        create n := {s};
9     else
10       create n := minimize(S ∪ {s}), where n(t) = [S];
11    n(t) := [n];
12    l(t) := l(t) ⊕ d;
13    workset := workset ∪ {t}
14 end
```

Figure 7: Modified *update* procedure.

Like $\delta(c)$, $\omega(c)$ will be given indirectly in the form of another annotation (called n) on the transitions of $\mathcal{A}_{pre^*}$. We use two data structures for this, called *wnode* and *wstruc*. If t is a transition, then $n(t)$ holds a reference to a *wnode*. (We shall denote a reference to some entity e by $[e]$.) A *wnode* is a set of *wstruc* items. A *wstruc* item is of the form $(d, [t], [r], N)$ where $d \in D$, $[t]$ is a reference back to t, $r \in \Delta$ is a rule, and N contains a sequence of references to *wnode*s. References may be *nil*, indicating a missing reference.

We can now extend Algorithm 1. The idea is that during execution, if $n(t) = [S]$, then $l(t) = \bigoplus_{(d,[t],[r],N)\in S} d$. An item $(d, [t], [r], N)$ in S denotes the following: Suppose that A_{pre^*} has an accepting path starting with t, and c is the configuration accepted by this path. Then, in the pushdown system, there is a path (or rather, a family of paths) with value d from c to some $c' \in C$, and this path starts with r. An accepting path (in A_{pre^*}) for a successor configuration can be constructed by replacing t with the transitions associated with the *wnode*s in N.

The concrete modifications to Algorithm 1 are as follows: In line 8, set $n \equiv nil$. In line 9, create a *wnode* $n := \{(1, [t], nil, ())\}$ for every $t \in \rightarrow_0$ and set $n(t) := [n]$.

Figure 7 shows a revised *update* procedure. Line 4 of Figure 7 computes the newly discovered value for transition t, and line 5 records how the new path was discovered. In line 6, if $l(t) \sqsubseteq d$, the update will not change $l(t)$ and nothing further needs to be done. If $d \sqsubset l(t)$ (see line 8), the new addition is strictly smaller than any path to t so far, and $n(t)$ only has to reference the new path. If d and $l(t)$ are incomparable, line 10 creates a new set consisting of the previous paths *and* the new path. Even though d is incomparable to $l(t)$, d might approximate ($\sqsubseteq$) one or more elements of S. The procedure *minimize* (not shown) removes these.

It is fairly straightforward to see that the information contained in S allows the reconstruction of a witness set involving t (see above). Moreover, every *wnode* created during execution contains references only to *wnode*s created earlier. Therefore, the process of reconstructing the witness set by decoding *wnode*/*wstruc* information must eventually terminate in a configuration from C.

During execution of the modified algorithm, several *wnode*s for the same transition t can be created; only one of them is referenced by t at any moment, although the other *wnode*s may still be referenced by other transitions. A garbage collector can be used to keep track of the references and remove those nodes to which there is no longer any chain of references from any transition.

In the totally ordered case described in Section 6.5, every *wnode* can contain exactly one *wstruc*.