Journal of Computer Security 12 (2004) 317–353
IOS Press

# Model checking SPKI/SDSI [*]

S. Jha and T. Reps
*Computer Sciences Department, University of Wisconsin, 1210 W. Dayton Street, Madison, WI 53706, USA*
*E-mail: {jha,reps}@cs.wisc.edu*

SPKI/SDSI is a framework for expressing naming and authorization issues that arise in a distributed-computing environment. In this paper, we establish a connection between SPKI/SDSI and a formalism known as pushdown systems (PDSs). We show that the SPKI/SDSI-to-PDS connection provides a framework for formalizing a variety of certificate-analysis problems. Moreover, the connection has computational significance: many analysis problems can be solved efficiently (i.e., in time polynomial in the size of the certificate set) using existing algorithms for model checking pushdown systems.

Keywords: SPKI/SDSI, model checking, pushdown system, naming, authorization, certificate-chain discovery, certificate-set analysis

## 1. Introduction

Systems with shared resources use access-control mechanisms for protection. There are two fundamental problems in access control: *authorization* and *enforcement*. Authorization addresses the following problem: should a request $r$ by a specific principal $A$ 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 of the parties are known and trusted. In a distributed system, the closed-world assumption is not valid. Trust management systems [6] solve the authorization problem in distributed systems by defining a formal language for expressing authorization and access-control policies, and rely 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 [27]. Two prominent trust management systems are Keynote [5] and SPKI/SDSI [16].

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. Clarke et al. [12] considered the problem of discovering a

[*]This work was supported in part by the National Science Foundation under grant CCR-9619219, by the Office of Naval Research under contracts N00014-01-1-0796 and N00014-01-1-0708, and by the Alexander von Humboldt Foundation. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes, notwithstanding any copyright notices affixed thereon. The views and conclusions contained herein are those of the authors, and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the above government agencies or the U.S. Government.

*certificate chain* for an authorization with respect to a set of SPKI/SDSI certificates; a certificate chain provides a proof that a client's public key is one of the keys that has been authorized to access a given resource – either directly or transitively, via one or more name-definition or authorization-delegation steps.

This paper studies the problem of certificate analysis in the context of SPKI/SDSI. In particular, we establish a connection between SPKI/SDSI (without threshold subjects) and a formalism known as *pushdown systems* (PDSs) [8,17]. (Except for Section 4.2, "SPKI/SDSI" refers to the language defined in [16] without threshold subjects; Section 4.2 discusses how our work can be extended to handle threshold subjects.)

Our work stems from a simple observation:

*A set of SPKI/SDSI name and authorization certificates defines a PDS.*

The significance of this connection, and the contributions made by the paper, can be summarized as follows:

- The SPKI/SDSI-to-PDS connection provides a framework for formalizing a variety of certificate-set analysis problems: Certificate-set analysis becomes a problem of model checking pushdown systems.[1] Analysis problems can be stated precisely in any of the standard formalisms for posing model-checking queries. Such problems include the authorization problem addressed by Clarke et al., i.e.,

  **Authorized access 1:** Given resource $R$ and principal $K$, is $K$ authorized to access $R$?

  However, there are many other questions that one may be interested in with respect to a certificate set $\mathcal{C}$, such as

  **Authorized access 2:** Given resource $R$ and name $N$ (not necessarily a principal), is $N$ authorized to access $R$?

  **Authorized access 3:** Given resource $R$, what names (not necessarily principals) are authorized to access $R$?[2]

  **Shared access 1:** For two given resources $R_1$ and $R_2$, what principals can access both $R_1$ and $R_2$?

  **Shared access 2:** Given two principals, $K_1$ and $K_2$, and a resource $R$, can both $K_1$ and $K_2$ access $R$?

  **Shared access 3:** For two given principals, $K_1$ and $K_2$, and a finite set of resources $\mathcal{R} = \{R_1, \ldots, R_k\}$, which resources from $\mathcal{R}$ can be accessed by both $K_1$ and $K_2$?

[1] There are many flavors of model checking. Henceforth, unless otherwise noted, the term "model checking" refers to model checking of pushdown systems [8,17]. Background on this problem is given in Section 3.

[2] In general, this can be an infinite set; as will be shown, the answer can be given in the form of a finite-state automaton that accepts the names that are authorized to access $R$.

**Compromisation assessment 1:** What resources from a finite set $\mathcal{R} = \{R_1, \ldots, R_k\}$ could principal $K$ have gained access to (solely) due to the presence of maliciously or accidentally issued certificate set $\mathcal{C}' \subseteq \mathcal{C}$?

**Compromisation assessment 2:** What principals could have gained access to resource $R$ (solely) due to the presence of maliciously or accidentally issued certificate set $\mathcal{C}' \subseteq \mathcal{C}$?

**Expiration vulnerability 1:** What resources from a finite set of resources $\{R_1, \ldots, R_k\}$ will principal $K$ be prevented from accessing if certificate set $\mathcal{C}' \subseteq \mathcal{C}$ expires?

**Expiration vulnerability 2:** What principals will be excluded from accessing resource $R$ if certificate set $\mathcal{C}' \subseteq \mathcal{C}$ expires?

**Universally guarded access 1:** Is it the case that all authorizations that can be issued for a given resource $R$ must involve a certificate signed by principal $K$?

**Universally guarded access 2:** Is it the case that all authorizations that grant a given principal $K'$ access to a finite set of resources $\{R_1, \ldots, R_k\}$ must involve a certificate signed by $K$?

- Analysis problems such as the ones listed above can be solved efficiently (i.e., in time polynomial in the size of certificate set $\mathcal{C}$) using existing model-checking algorithms for PDSs. In addition, annotating certificates with labels from a semiring enables us to address additional types of questions, such as "When does a specific authorization expire?" (see Section 4.6).
- In addition to the specific queries listed above, any certificate-set analysis question that can be posed as an LTL query can be solved in the time polynomial in the size of $\mathcal{C}$.
- In the case of certificate-chain discovery, we show that an operation that is used as a subroutine in algorithms for model checking PDSs provides a new algorithm for the problem. A special-purpose algorithm for certificate-chain discovery was developed by Clarke et al. [12]. Although the worst-case asymptotic running time for the new algorithm is the same as that of Clarke et al., the improved handling of tabulated data does lead to an asymptotic improvement for the family of examples given by Clarke et al. to illustrate that their worst-case upper bound is tight to within a constant factor. This family of examples does not cause our algorithm to exhibit worst-case behavior (see Section 4.5).
- The closure $\mathcal{C}^\star$ of a set of certificates $\mathcal{C}$ includes additional certificates that can be "derived" from a chain of certificates in $\mathcal{C}$. In general, the closure $\mathcal{C}^\star$ of $\mathcal{C}$ can be an infinite set. To circumvent this problem Clarke et al. defined a restricted type of closure called name-reduction closure, which is guaranteed to be finite. The PDS-based algorithm computes the actual closure, not just the name-reduction closure. (The PDS-based algorithm uses a finite-state automaton to represent a set of certificates, which hence provides a finite representation of a potentially infinite set.) In the SPKI/SDSI context, this allows us to answer questions where the capability to return an infinite set is necessary, such as

Table 1

Kinds of arrows used in the paper

| | |
|---|---|
| $K\ \mathtt{A} \longrightarrow S$ | An SPKI/SDSI name cert $(K, \mathtt{A}, S, V)$ |
| $K\ \Box \longrightarrow S\ \Box$ | An SPKI/SDSI auth cert $(K, S, D, T, V)$, with delegation bit $D$ on |
| $K\ \Box \longrightarrow S\ \blacksquare$ | An SPKI/SDSI auth cert $(K, S, D, T, V)$, with delegation bit $D$ off |
| $\langle p, \gamma \rangle \hookrightarrow \langle q, w \rangle$ | Transition rule of a PDS |
| $\langle p, w \rangle \Rightarrow \langle q, w' \rangle$ | Immediate-successor relation of a PDS |
| $\langle p, w \rangle \Rightarrow^{+} \langle q, w' \rangle$ | Transitive closure of immediate-successor relation of a PDS |
| $\langle p, w \rangle \Rightarrow^{\star} \langle q, w' \rangle$ | Reflexive transitive closure of immediate-successor relation of a PDS |
| $p \xrightarrow{w} q$ | Reachability relation on states of a configuration automaton |
| $p \overset{\gamma}{\rightsquigarrow} q$ | The relation $p\ (\xrightarrow{\epsilon})^{\star} \xrightarrow{\gamma} (\xrightarrow{\epsilon})^{\star}\ q$ in a configuration automaton |

"Authorized access 3" ("Given resource R, what names [not just principals] are authorized to access R?").

The remainder of the paper is organized as follows: Section 2 provides an introduction to SPKI/SDSI, and describes the algorithm for certificate-chain discovery from [12]. Section 3 provides background on model checking pushdown systems. Section 4 discusses applications of the formal machinery to certificate-set analysis problems. Section 5 discusses related work.

The paper is structured so as to be self-contained. It deals with several problem domains, and uses several kinds of arrows to denote relationships among different kinds of objects; these are summarized in Table 1.

Readers familiar with [12] and [8,17] may wish to skip Sections 2 and 3, respectively (although there are some minor notational differences with those papers; see footnotes 4 and 5).

## 2. Background on SPKI/SDSI

In SPKI/SDSI, all principals are represented by their public keys. A *principal* can be an individual, process, host, or any other active entity. SPKI/SDSI does not make any distinction between the principal and its public key, i.e., the principal *is* its public key.

$\mathcal{K}$ denotes the set of public keys. Specific keys are denoted by $K, K_A, K_B, K', \ldots$, etc. Data-structure issues related to representation of keys can be found in [16].

An *identifier* is a word over some alphabet $\Sigma$. The set of identifiers is denoted by $\mathcal{A}$. Identifiers are usually written in typewriter font, e.g., `A` and `Bob`.

A *term* is a key followed by 0 or more identifiers. Terms are either keys, local names, or extended names. A *local name* is of the form $K\ \mathtt{A}$, where $K \in \mathcal{K}$ and $\mathtt{A} \in \mathcal{A}$ is an identifier. For example, $K\ \mathtt{Bob}$ is a local name. Local names are important in SPKI/SDSI because they create a decentralized name space. The set of all local names is denoted by $\mathcal{N}_L$, and the local name space of $K$ (local names of the form $K\ \mathtt{A}$) is denoted by $\mathcal{N}_L(K)$.

An *extended name* is of the form $K\ \sigma$, where $K \in \mathcal{K}$ and $\sigma$ is a sequence of identifiers of length greater than one. For example, $K$ `UW CS faculty` is an extended name. Let $\mathcal{N}_E$ be the set of extended names and $\mathcal{N}_E(K)$ denote the set of extended names beginning with key $K$. The set of names $\mathcal{N}$ is $\mathcal{N}_L \cup \mathcal{N}_E$, and the name space $\mathcal{N}(K)$ of the key $K$ is $\mathcal{N}_L(K) \cup \mathcal{N}_E(K)$. The set of terms $\mathcal{T}$ is thus $\mathcal{K} \cup \mathcal{N}$.

## 2.1. Certificates

SPKI/SDSI has two types of certificates, or "certs". The first type of certificate, called a *name cert*, provides definitions of local names. Authorizations are specified using *authorization certs* (or *auth certs*, for short).

**Name Certificates.** 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 the local name space $\mathcal{N}_L(K)$. 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 in $\mathcal{T}$. Intuitively, $S$ gives additional meaning for the local name $K\ A$.
- The *validity specification* $V$ provides information regarding the validity of the certificate. Usually, the validity specification $V$ 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. For a complete explanation of validity specifications, see [16]. In the context of the authorization problem, we will generally ignore the validity specification, and assume that we are working exclusively with valid certificates. (Extensions to handle certain types of validity specifications are discussed in Section 4.6.)

**Authorization Certificates.** An auth cert grants or delegates a specific authorization from an issuer to a subject. Specifically, an auth cert $C$ is a five-tuple $(K, S, D, T, V)$, where

- The *issuer* $K$ is a public key, which is also used to sign the cert. The issuer is the one granting a specific authorization.
- The *subject* $S$ is a term.
- If the *delegation bit* $D$ is turned on, then the key 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 authorization specification $T$ of an auth cert refers to some resource, e.g., the authorization specification `(dir /afs/cs.wisc.edu/public/tmp)` refers to the resource `/afs/cs.wisc.edu/public/tmp`. However, in a slight abuse of terminology, we will refer to an authorization specification $T$ as a resource. Moreover,

we assume that there is a partial order $\subseteq$ between authorization specifications, e.g., $T' \subseteq T$ denotes the fact that $T$ is "more permissive" than $T'$. In the example shown below, $T' \subseteq T$.[3]

```
T' ((dir /afs/cs.wisc.edu/public/tmp) read)
T  ((dir /afs/cs.wisc.edu/public/tmp) (* set read write))
```

- The *validity specification* $V$ for an auth cert is same as in the case of name cert.

We will treat certs as rewrite rules:

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

### 2.2. *An authorization example*

In this section, we describe an authorization example that will be used for illustrative purposes later in the paper.

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 $(K, S, D, T, V)$ can be viewed as an ACL entry, where keys or principals represented by the subject $S$ are given permission to access resource $T$. We assume that each resource $T$ has a *unique owner* (denoted by $K_{\text{owner}[T]}$); all certificate chains granting access to resource $T$ should "start at" $K_{\text{owner}[T]}$. For instance, suppose that Alice (i.e., $K_A$) wants to login to host $H$. Initially, the owner of host $H$, $K_{\text{owner}[H]}$, denies access to her, but reports the following ACL entry to Alice (written as an auth cert):

$$K_{\text{owner}[H]} \,\square \longrightarrow K_0\ \mathtt{UW}\ \mathtt{CS}\ \mathtt{faculty} \,\square$$

Given the set of certs $\mathcal{C}$ shown in Fig. 1, Alice has to "prove" that she is authorized to access host $H$.

### 2.3. *Name-reduction closure*

We now describe the algorithm given in [12,14]. The reader is referred to [14] for additional details.

First, we define the concept of a closure of a set of certificates $\mathcal{C}$. 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 at the beginning. For uniformity, we also refer to strings of the form

[3]"`(* set read write)`" denotes read/write permission; "`read`" denotes read permission.

| | |
|---|---|
| $K_{\text{owner}[H]}\ \square \longrightarrow K_0\ \texttt{UW}\ \texttt{CS}\ \texttt{faculty}\ \square$ | (1) |
| $K_0\ \texttt{UW} \longrightarrow K_1$ | (2) |
| $K_1\ \texttt{CS} \longrightarrow K_2$ | (3) |
| $K_2\ \texttt{faculty} \longrightarrow K_3\ \texttt{Bob}$ | (4) |
| $K_3\ \texttt{Bob} \longrightarrow K_B$ | (5) |
| $K_B\ \square \longrightarrow K_4\ \texttt{Alice}\ \blacksquare$ | (6) |
| $K_4\ \texttt{Alice} \longrightarrow K_A$ | (7) |

Fig. 1. A set of certs.

$S\ \square$ 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,

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

A term $S$ and a rule $L \rightarrow R$ are called *compatible* if $S$ is of the form $LX$. Given a set of certificates $\mathcal{C}$ and a term $S$, we define $\mathcal{C}(S)$ as the following set:

$$\{C(S) \mid C \text{ is compatible with } S \text{ and } C \in \mathcal{C}\}$$

Consider two rules $C_1 = (L_1 \longrightarrow R_1)$ and $C_2 = (L_2 \longrightarrow R_2)$. Moreover, 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* $C_2 \circ C_1$ of the two rules $C_1$ and $C_2$ is the rule $L_1 \longrightarrow R_2X$. For example, consider the following two rules:

$$\begin{aligned} C_1 &: \ K_A\ \texttt{friends} \longrightarrow K_A\ \texttt{Bob}\ \texttt{myFriends} \\ C_2 &: \ K_A\ \texttt{Bob} \longrightarrow K_B \end{aligned}$$

The composition $C_2 \circ C_1$ is the following rule:

$$K_A\ \texttt{friends} \longrightarrow K_B\ \texttt{myFriends}$$

Two rules $C_1$ and $C_2$ are called *compatible* if their composition $C_2 \circ C_1$ is well-defined. Given a set of certificates $\mathcal{C}$, its *closure* (denoted by $\mathcal{C}^\star$) is the smallest set of certificates that includes $\mathcal{C}$ and is closed under composition.[4] In general, $\mathcal{C}^\star$ is infinite and hence cannot be computed directly. For example, consider the set of certificates $\mathcal{C} = \{(K\ \texttt{A} \longrightarrow K\ \texttt{A}\ \texttt{A})\}$. The closure $\mathcal{C}^\star$ of $\mathcal{C}$ is the following set:

$$\{(K\ \texttt{A} \longrightarrow K\ \texttt{A}^i) : i \geqslant 2\}$$

[4] For rule application, we write $(L \rightarrow R)(S)$ instead of $S \circ (L \rightarrow R)$ as used in [12]. For the composition of $C_1 = (L_1 \rightarrow L_2X)$ and $C_2 = L_2 \rightarrow R_2$, we write $C_2 \circ C_1$ instead of $C_1 \circ C_2$ (so that $(C_2 \circ C_1)(L_1) = (C_2(C_1(L_1))) = R_2X$). Finally, we use $\mathcal{C}^\star$ instead of $\mathcal{C}^+$.

Given a name $N$ and a set of certificates $\mathcal{C}$, $V_{\mathcal{C}}(N)$ is defined as

$$V_{\mathcal{C}}(N) = \mathcal{C}^{\star}(N) \cap \mathcal{K}.$$

In other words, $V_{\mathcal{C}}(N)$ is the set of keys that can be obtained from $N$ by using the rewrite rules corresponding to the set of certs $\mathcal{C}$. In applications, if $N$ is granted a certain authorization, every key in $V_{\mathcal{C}}(N)$ is also indirectly granted that authorization. For instance, in the authorization example from Section 2.2, it can be shown that $K_A \in V_C(K_0\ \texttt{UW CS faculty})$, and thus Alice has authorization to login to host $H$.

Because the closure $\mathcal{C}^{\star}$ of a set of certs $\mathcal{C}$ can be infinite, the concept of a *name-reduction closure* was introduced in [12,14]. A *reducing cert* is of the form $K\ \texttt{A} \longrightarrow K'$. A *name reduction* is a composition of two compatible rules $C_1$ and $C_2$, where $C_2$ is a reducing cert. The *name-reduction closure* $\mathcal{C}^{\sharp}$ of a set of certificates $\mathcal{C}$ is defined as the smallest set of certificates that contains $\mathcal{C}$ and is closed under name reduction. Given a name $N$ and a set of certs $\mathcal{C}$, the following equality is proved in [12]:

$$\mathcal{C}^{\star}(N) \cap \mathcal{K} = \mathcal{C}^{\sharp}(N) \cap \mathcal{K}.$$

In other words, it is safe to inspect just the name-reduction closure to find the set of keys that correspond to a name $N$ under certificate set $\mathcal{C}$.

We now return to our authorization example and describe the four-step procedure from [12] for determining whether a principal $K_P$ is authorized to access a given resource according to an authorization specification $T_P$ (to perform a particular operation), given a set of certificates $\mathcal{C}$. A tuple $(K_P, T_P)$ is referred to as a *request*.

1. **Remove useless certificates**
   All invalid name and auth certificates are removed from the set $\mathcal{C}$. All auth certs $(K, S, D, T, V)$ such that $T_P \not\subseteq T$ are also removed from $\mathcal{C}$.
2. **Name reduction**
   Compute the name-reduction closure $\mathcal{C}^{\sharp}$ for the set $\mathcal{C}$. The name-reduction closure of the set $\mathcal{C}$ shown in Fig. 1 yields the additional certs shown in Fig. 2.

| | |
|---|---|
| $K_2\ \texttt{faculty} \longrightarrow K_B$ | $(8) = (5) \circ (4)$ |
| $K_B\ \square \longrightarrow K_A\ \blacksquare$ | $(9) = (7) \circ (6)$ |
| $K_{\text{owner}[H]}\ \square \longrightarrow K_1\ \texttt{CS faculty}\ \square$ | $(10) = (2) \circ (1)$ |
| $K_{\text{owner}[H]}\ \square \longrightarrow K_2\ \texttt{faculty}\ \square$ | $(11) = (3) \circ (10)$ |
| $K_{\text{owner}[H]}\ \square \longrightarrow K_B\ \square$ | $(12) = (8) \circ (11)$ |

Fig. 2. Additional rules added by name-reduction closure.

3. **Depth-First Search**
   First, remove all the rules not of the form $K_1\ \square \longrightarrow K_2\ \square$ or $K_1\ \square \longrightarrow K_2\ \blacksquare$. In our example, the only rules that remain after this step are

   $$K_B\ \square \longrightarrow K_A\ \blacksquare,$$
   $$K_{\text{owner}[H]}\ \square \longrightarrow K_B\ \square.$$

   Second, remove all rules of the form $K_i\ \square \longrightarrow K_j\ \blacksquare$, where $K_j \neq K_P$. Third, construct a graph with a vertex for each key. There is an edge from $K_i$ to $K_j$ if there is a rule of the form $K_i\ \square \longrightarrow K_j\ \blacksquare$ or $K_i\ \square \longrightarrow K_j\ \square$. In our example, the edges are $K_{\text{owner}[H]} \longrightarrow K_B$ and $K_B \rightarrow K_A$. Fourth, perform depth-first search to determine whether there is a path from $K_{\text{owner}[H]}$ to $K_P$. In our example, there is a path from $K_{\text{owner}[H]}$ to Alice's key $K_A$, so Alice is authorized to login to host $H$.
4. **Reconstruct the certificate chain**
   Information from the previous steps can be used to create a certificate chain that "proves" that principal $K_P$ is authorized to access the desired resource. In our example, the certificate chain

   (1 2 3 4 5 6 7)

   proves that Alice is authorized to login to the host $H$, because

   $$((7) \circ (6) \circ (5) \circ (4) \circ (3) \circ (2) \circ (1))(K_{\text{owner}[H]}\ \square) = K_A\ \blacksquare.$$

   Certificate-chain reconstruction requires that additional information be stored during the algorithm used to perform name-reduction closure. Because the smallest size of a certificate chain can be exponential in the number of certs, it may be desirable to report certificate chains in a factored form [14, Chapter 3].

The time and space complexity of name-reduction closure can be analyzed as follows: Let $\mathcal{C}$ be the set of certificates and $n_K$ be the number of keys occurring in $\mathcal{C}$. Consider a typical certificate of the form $L \rightarrow KA_1A_2 \cdots A_m$. After one name reduction, we obtain a rule of the form $L \rightarrow K_1A_2 \cdots A_m$. After $i < m$ name reductions, we obtain rules of the form $L \rightarrow K_iA_{i+1} \cdots A_m$.

After $m$ reductions, we obtain a rule of the form $L \rightarrow K_m$. There are $n_K$ possibilities for the keys $K_1, \ldots, K_m$, so there are $n_Km$ possibilities for the rules that are generated. Let $|\mathcal{C}|$ be the sum of the lengths of the right-hand sides of all rules that occur in $\mathcal{C}$. The maximum number of new rules that can be produced is $n_K|\mathcal{C}|$. A rule can be compatible with at most $n_K$ reducing certs. Therefore, each rule can result in $O(n_K)$ work, and thus the time complexity of name-reduction closure is $O(n_K^2|\mathcal{C}|)$.

The number of nodes and edges in the graph constructed in the depth-first-search step is bounded by $n_K$ and $O(n_K^2)$, respectively. Therefore, the time complexity of the depth-first search in the authorization procedure is $O(n_K + n_K^2)$. Hence, the time complexity of the name-reduction closure step dominates the running time of the procedure. Because the number of new rules produced is bounded by $n_K|\mathcal{C}|$, the space complexity of the procedure is $O(n_K|\mathcal{C}|+n_K^2)$, where the second term appears because of the depth-first search. Data structures for representing certs are discussed in detail in Elien's thesis [14].

## 3. Background on model checking pushdown systems

This section provides the necessary background on model checking of *pushdown systems* (*PDS*s). A detailed treatment of model checking PDSs, including the computational complexity of various problems, can be found in [8]. An improved algorithm for model checking PDSs was presented in [17]. The material in this section is largely based on [17], and a few further improvements found in [25].

A pushdown system is a triple $\mathcal{P} = (P, \Gamma, \Delta)$, where $P$ is a finite set of *control locations*, $\Gamma$ is a finite *stack alphabet*, and $\Delta \subseteq (P \times \Gamma) \times (P \times \Gamma^\star)$ is a finite set of *transition rules*. If $((q,\gamma),(q',w)) \in \Delta$, then we write it as $\langle q,\gamma\rangle \hookrightarrow \langle q',w\rangle$.

A *configuration* of $\mathcal{P}$ is a pair $\langle q,w\rangle$, where $q \in P$ is a control location and $w \in \Gamma^\star$ represents the *stack contents*. The set of all configurations is denoted by $\mathcal{C}$. A *surface configuration* is a pair $\langle q,\gamma\rangle$, where $q \in P$ and $\gamma \in \Gamma$. If $\langle q,\gamma\rangle \hookrightarrow \langle q',w\rangle$, then for all $v \in \Gamma^\star$ the configuration $\langle q,\gamma v\rangle$ is an *immediate predecessor* of $\langle q',wv\rangle$, and $\langle q',wv\rangle$ is an *immediate successor* of $\langle q,\gamma v\rangle$ (denoted by $\langle q,\gamma v\rangle \Rightarrow \langle q',wv\rangle$). The reflexive transitive closure (known as the *reachability relation*) and the transitive closure of the immediate-successor relation are denoted by $\Rightarrow^\star$ and $\Rightarrow^+$, respectively.[5] A *run* of $\mathcal{P}$ is a sequence of configurations $c_0, c_1, \ldots, c_n$ such that $c_i$ is an immediate predecessor of $c_{i+1}$.

Pushdown systems are similar to pushdown automata; however, unlike pushdown automata they do not have an input alphabet. Therefore, a PDS should not be viewed as a language recognizer, but as a mechanism that specifies an infinite-state transition system, i.e., a transition system that may have an infinite number of distinct states. Consider a PDS $\mathcal{P} = (P, \Gamma, \Delta)$, where $P = \{K, K'\}$, $\Gamma = \{A\}$, and $\Delta$ consists of the following transition rules: $\langle K,A\rangle \hookrightarrow \langle K,AA\rangle$ and $\langle K,A\rangle \hookrightarrow \langle K',\epsilon\rangle$, where $\epsilon$ denotes the empty string of symbols from $\Gamma$. Starting from the configuration $\langle K,A\rangle$, the infinite-state transition system generated by the PDS $\mathcal{P}$ is shown in Fig. 3.

The remainder of the section summarizes algorithms and complexity results that provide the basic toolkit for addressing reachability questions on PDSs. The basic backward and forward reachability algorithms are discussed in Sections 3.1 and 3.2, respectively. Linear time logic (LTL) is a powerful query language for asking generalized reachability questions about a PDS. LTL model checking of PDSs is discussed in Section 3.3.

[5] In [17], the symbol $\Rightarrow$ denotes the reflexive transitive closure of the immediate predecessor relation. We use $\Rightarrow$ for the immediate predecessor relation, and $\Rightarrow^\star$ for its reflexive transitive closure.

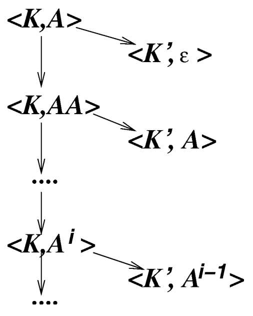

Fig. 3. Infinite-state transition systems generated by a PDS.

Given a set of configurations $C \subseteq \mathcal{C}$, the set of predecessors of $C$ (denoted by *pre*$[\mathcal{P}](C)$) is

$$\{c \mid \exists c' \in C.c \Rightarrow c'\}$$

The reflexive transitive closure of *pre*$[\mathcal{P}]$ is denoted by *pre*$^\star$; thus, *pre*$^\star[\mathcal{P}](C)$ is

$$\{c \mid \exists c' \in C.c \Rightarrow^\star c'\}$$

The set of immediate successors *post*$[\mathcal{P}](C)$ of a set of configurations $C$ is defined similarly. The reflexive transitive closure of *post*$[\mathcal{P}]$ is denoted by *post*$^\star[\mathcal{P}]$. When $\mathcal{P}$ is understood, we will merely write *pre*, *pre*$^\star$, *post*, and *post*$^\star$.

## 3.1. Computing pre$^\star$

Assume that we are given a pushdown system $\mathcal{P} = (P, \Gamma, \Delta)$. A *regular* set of configurations of $\mathcal{P}$ can be represented with a finite-state automaton, called a *configuration automaton* of $\mathcal{P}$, whose input alphabet is $\mathcal{P}$'s stack alphabet.

Formally, a configuration automaton of $\mathcal{P}$ is an automaton $\mathcal{A} = (\Gamma, Q, \delta, P, F)$, where $Q$ is a finite set of states and the set of locations $P$ of $\mathcal{P}$ is a subset of $Q$; $\delta \subseteq Q \times \Gamma \times Q$ is the set of *transitions*; $P$ is the set of *initial states*; and $F \subseteq Q$ is the set of *final states*. The configuration automaton's *reachability relation*, denoted by $\xrightarrow{w} \subseteq Q \times \Gamma^\star \times Q$, is defined as the smallest relation satisfying:

- $q \xrightarrow{\epsilon} q$ for every $q \in Q$,
- if $(q, \gamma, q') \in \delta$, then $q \xrightarrow{\gamma} q'$, and
- if $q \xrightarrow{w} q''$ and $q'' \xrightarrow{\gamma} q'$, then $q \xrightarrow{w\gamma} q'$.

Henceforth, we will refer to a configuration automaton simply as an automaton. An automaton *accepts* or *recognizes* a configuration $\langle p, w\rangle$ of $\mathcal{P}$ if $p \xrightarrow{w} q$ for some $p \in P$ and $q \in F$. The set of configurations recognized by an automaton $\mathcal{A}$ is denoted by *Conf*$(\mathcal{A})$.

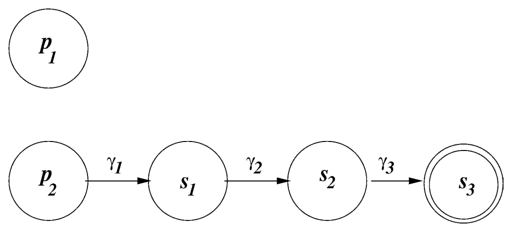

Fig. 4. Automaton that accepts $C = \{\langle p_2, \gamma_1\gamma_2\gamma_3\rangle\}$.

**Example 1.** Consider a PDS $\mathcal{P} = (P, \Gamma, \Delta)$, where $P = \{p_1, p_2\}$, $\Gamma = \{\gamma_1, \ldots, \gamma_6\}$, and $\Delta$ consists of the following transition rules:

$$\begin{aligned}
&(p_2, \gamma_4) \hookrightarrow (p_2, \gamma_1\gamma_2)\\
&(p_1, \gamma_5) \hookrightarrow (p_2, \gamma_4\gamma_3)\\
&(p_1, \gamma_6) \hookrightarrow (p_1, \epsilon)
\end{aligned}$$

The automaton shown in Fig. 4 recognizes the set of configurations $C = \{\langle p_2, \gamma_1\gamma_2\gamma_3\rangle\}$.

Assume that we are given a regular set of configurations $C$ accepted by an automaton $\mathcal{A}$. It has been shown that the set of configurations $pre^\star(C)$ is also regular [7–9,11]. An automaton recognizing $pre^\star(C)$ can be constructed from $\mathcal{A}$ by adding transitions to $\mathcal{A}$ using the following *saturation rule*; i.e., we add transitions to the automaton until no more can be added:

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

**Theorem 1** [25]. *Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a PDS and $\mathcal{A} = (\Gamma, Q, \delta, P, F)$ be a configuration automaton of $\mathcal{P}$. There exists an automaton $\mathcal{A}_{pre^\star}$ that recognizes $pre^\star(Conf(\mathcal{A}))$. Moreover, $\mathcal{A}_{pre^\star}$ can be constructed in $O(n_Q^2 n_\Delta)$ time and $O(n_Q n_\Delta + n_\delta)$ space, where $n_Q = |Q|$, $n_\delta = |\delta|$, and $n_\Delta$ is the sum of the lengths of the right-hand sides of transition rules in $\Delta$. The length of the right-hand side of transition rule $\langle p, \gamma\rangle \hookrightarrow \langle p, w\rangle$ is* $\max\{1, |w|\}$.

**Example 2.** Consider the PDS from Example 1. Recall that the automaton in Fig. 4 recognizes the set of configurations $C = \{\langle p_2, \gamma_1\gamma_2\gamma_3\rangle\}$. The automaton $A$ that recognizes $pre^\star(C)$ is shown in Fig. 5. The transition rule $(p_1, \gamma_6) \hookrightarrow (p_1, \epsilon)$ from $\mathcal{P}$ causes a self-loop $(p_1, \gamma_6, p_1)$ to be added to $A$. The transition rule $(p_2, \gamma_4) \hookrightarrow (p_2, \gamma_1\gamma_2)$ from $\mathcal{P}$ and the fact that $p_2 \xrightarrow{\gamma_1\gamma_2} s_2$ holds in $A$ causes the transition $(p_2, \gamma_4, s_2)$ to be added to $A$. The transition rule $(p_1, \gamma_5) \hookrightarrow (p_2, \gamma_4\gamma_3)$ in $\mathcal{P}$ and the fact that $p_2 \xrightarrow{\gamma_2\gamma_3} s_3$ holds in $A$ causes the transition $(p_1, \gamma_5, s_3)$ to be added to $A$. The automaton shown in Fig. 5 accepts the following set of configurations:

$$\{\langle p_1, \gamma_6^i\gamma_5\rangle\} \cup \{\langle p_2, \gamma_1\gamma_2\gamma_3\rangle, \langle p_2, \gamma_4\gamma_3\rangle\}.$$

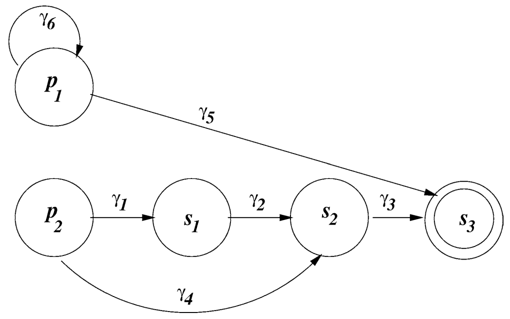

Fig. 5. Automaton that accepts $pre^\star(C) = \{\langle p_1, \gamma_6^i\gamma_5\rangle\} \cup \{\langle p_2, \gamma_1\gamma_2\gamma_3\rangle, \langle p_2, \gamma_4\gamma_3\rangle\}$.

## 3.2. *Computing post*$^\star$

Consider a PDS $\mathcal{P} = (P, \Gamma, \Delta)$ and a regular set of configurations $C$ that is represented as an automaton $\mathcal{A}$. We will assume that each transition rule $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$ of $\Delta$ satisfies $|w| \leqslant 2$. This assumption involves no loss of generality because a PDS that does not satisfy this constraint can be converted into one that does. Suppose that we are given a general PDS $\mathcal{P}' = (P', \Gamma, \Delta')$. Consider a transition rule $\langle p, \gamma\rangle \hookrightarrow \langle p', \gamma_1, \ldots, \gamma_k\rangle$, where $k \geqslant 3$. We add $k-2$ new control locations $p_1, \ldots, p_{k-2}$ and replace the original rule with the following $k-1$ transition rules:

$$\begin{aligned}
\langle p, \gamma\rangle &\hookrightarrow \langle p_1, \gamma_{k-1}\gamma_k\rangle \\
\langle p_1, \gamma_{k-1}\rangle &\hookrightarrow \langle p_2, \gamma_{k-2}\gamma_{k-1}\rangle \\
&\vdots \\
\langle p_i, \gamma_{k-i}\rangle &\hookrightarrow \langle p_{i+1}\gamma_{k-i-1}\gamma_{k-i}\rangle \\
&\vdots \\
\langle p_{k-2}, \gamma_2\rangle &\hookrightarrow \langle p', \gamma_1\gamma_2\rangle
\end{aligned}$$

Assume that we are given a regular set of configurations $C$ by means of an automaton $\mathcal{A}$. An automaton $\mathcal{A}_{post^\star}$ that accepts $post^\star(C)$ is obtained from $\mathcal{A}$ by the following two-phase procedure:

- **Phase I**
  For each pair $(p', \gamma')$ such that $\mathcal{P}$ contains at least one rule of the form $\langle p, \gamma\rangle \hookrightarrow \langle p', \gamma'\gamma''\rangle$, add a new state $q_{p',\gamma'}$.
- **Phase II (saturation phase)**
  In this phase, new transitions are added to the automaton until no more rules can be added. (The symbol $\overset{\gamma}{\rightsquigarrow}$ denotes the relation $(\xrightarrow{\epsilon})^\star \xrightarrow{\gamma} (\xrightarrow{\epsilon})^\star$.) The rules for adding new transitions are as follows:

  – If $\langle p, \gamma\rangle \hookrightarrow \langle p', \epsilon\rangle \in \Delta$ and $p \overset{\gamma}{\rightsquigarrow} q$ in the current automaton, add a transition $(p', \epsilon, q)$.

- If $\langle p, \gamma \rangle \hookrightarrow \langle p', \gamma' \rangle \in \Delta$ and $p \overset{\gamma}{\rightsquigarrow} q$ in the current automaton, add a transition $(p', \gamma', q)$.
- If $\langle p, \gamma \rangle \hookrightarrow \langle p', \gamma'\gamma'' \rangle \in \Delta$ and $p \overset{\gamma}{\rightsquigarrow} q$ in the current automaton, first add $(p', \gamma', q_{p',\gamma'})$ and then $(q_{p',\gamma'}, \gamma'', q)$.

**Theorem 2** [25]. *Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a pushdown system, and $\mathcal{A} = (\Gamma, Q, \delta, P, F)$ be a configuration automaton of $\mathcal{P}$. There exists an automaton $\mathcal{A}_{post^\star}$ recognizing $post^\star(Conf(\mathcal{A}))$. Moreover, $\mathcal{A}_{post^\star}$ can be constructed in time and space $O(n_P n_\Delta (n_1 + n_2) + n_P n_\delta)$, where $n_P = |P|$, $n_\Delta = |\Delta|$, $n_Q = |Q|$, $n_\delta = |\delta|$, $n_1 = |Q \backslash P|$, and $n_2$ is the number of different pairs $(p', \gamma')$ such that there is a rule of the form $\langle p, \gamma \rangle \hookrightarrow \langle p', \gamma'\gamma'' \rangle$ in $\Delta$.*

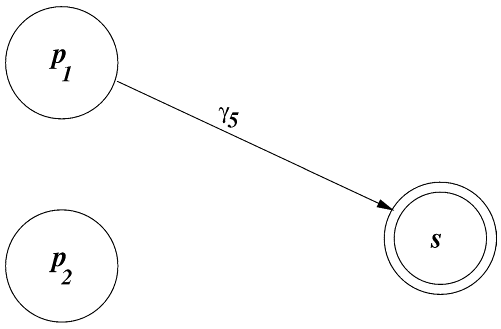

Fig. 6. Automaton that accepts $C = \{\langle p_1, \gamma_5 \rangle\}$.

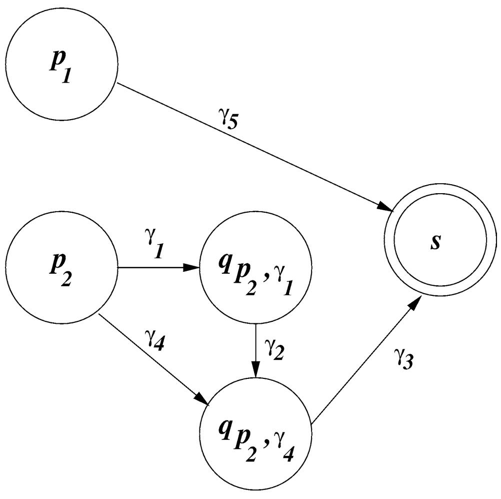

Fig. 7. Automaton that accepts $post^\star(C) = \{\langle p_1, \gamma_5 \rangle, \langle p_2, \gamma_4\gamma_3 \rangle, \langle p_2, \gamma_1\gamma_2\gamma_3 \rangle\}$.

**Example 3.** Consider again the PDS $\mathcal{P} = (P, \Gamma, \Delta)$, where $P = \{p_1, p_2\}$, $\Gamma = \{\gamma_1, \cdots, \gamma_6\}$, and $\Delta$ contains the following transition rules:

$$\langle p_2, \gamma_4 \rangle \hookrightarrow \langle p_2, \gamma_1\gamma_2 \rangle$$
$$\langle p_1, \gamma_5 \rangle \hookrightarrow \langle p_2, \gamma_4\gamma_3 \rangle$$
$$\langle p_1, \gamma_6 \rangle \hookrightarrow \langle p_1, \epsilon \rangle$$

Consider the automaton shown in Fig. 6, which accepts the set of configurations $C = \{\langle p_1, \gamma_5 \rangle\}$. The automaton corresponding to *post*$^\star(C)$ is shown in Fig. 7. The states $q_{p_2,\gamma_1}$ and $q_{p_2,\gamma_4}$ are the two states added during Phase I of the saturation procedure. The automaton shown in Fig. 7 accepts the following set of configurations:

$$\{\langle p_1, \gamma_5 \rangle, \langle p_2, \gamma_4\gamma_3 \rangle, \langle p_2, \gamma_1\gamma_2\gamma_3 \rangle\}.$$

## 3.3. Model checking for linear time logic

Let $AP$ be a finite set of atomic propositions, and let $\Sigma = 2^{AP}$. Let $\phi$ be an LTL formula over the atomic propositions $AP$. (The reader should consult [13, Chapter 3] and [8] for the syntax and semantics of LTL.) Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a PDS, and let $\Omega : (P \times \Gamma) \to \Sigma$ be a labeling function that associates a set of atomic propositions with each surface configuration $\langle p, \gamma \rangle$. By extension, the set of atomic propositions that hold at a configuration $\langle q, \gamma w \rangle$ is given by $\Omega(\langle q, \gamma \rangle)$.

We are interested in the following model-checking problem:

Given a configuration $c$ of $\mathcal{P}$ and an LTL formula $\phi$, determine whether $c$ satisfies $\phi$.

The *head* of a transition rule $\langle p, \gamma \rangle \hookrightarrow \langle p', w \rangle$ is the surface configuration $\langle p, \gamma \rangle$. Suppose that $\langle p, \gamma \rangle$ is in the set *pre*$^\star(\{\langle p, \gamma v \rangle\})$, or equivalently, $\langle p, \gamma \rangle \Rightarrow^\star \langle p, \gamma v \rangle$. In this case, we have the following path in the transition system defined by the PDS along which the head $\langle p, \gamma \rangle$ keeps repeating:

$$\langle p, \gamma \rangle \Rightarrow^\star \langle p, \gamma v \rangle \Rightarrow^\star \langle p, \gamma v^2 \rangle \Rightarrow^\star \cdots \Rightarrow^\star \langle p, \gamma v^i \rangle \Rightarrow^\star \cdots$$

Identifying such repeating heads is crucial in LTL model checking of PDSs. First, we generalize slightly the concept illustrated above.

**Definition 1.** Assume that we are given a PDS $\mathcal{P} = (P, \Gamma, \Delta)$ and a set of locations $G \subseteq P$. Given two configurations $c$ and $c'$, we say that $c \Rightarrow^{r(G)} c'$ if and only if $c \Rightarrow^\star \langle g, u \rangle \Rightarrow^+ c'$ such that $g \in G$, i.e., there is a path of length $\geqslant 1$ from $c$ to $c'$ that passes through a configuration whose location is in the set $G$.

A transition rule's head $\langle p, \gamma \rangle$ is called *$G$-repeating* if there exists a $v \in \Gamma^\star$ such that $\langle p, \gamma \rangle \Rightarrow^{r(G)} \langle p, \gamma v \rangle$. The set of heads and $G$-repeating heads corresponding to a PDS $\mathcal{P}$ and set of locations $G$ are denoted by $H(\mathcal{P})$ and $R(\mathcal{P}, G)$, respectively.

**Theorem 3** [25]. *Assume that we are given a PDS $\mathcal{P} = (P, \Gamma, \Delta)$ and a set of locations $G \subseteq P$. The set of repeating heads $R(\mathcal{P}, G)$ can be computed in $O(n_P^2 n_\Delta)$ time and $O(n_P n_\Delta)$ space, where $n_P = |P|$ and $n_\Delta = |\Delta|$.*

A *Büchi automaton* $\mathcal{B} = (\Sigma, Q, \delta, q_0, F)$ is a 5-tuple, where $\Sigma$ is the alphabet, $Q$ is the set of states, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. The set of infinite words over the alphabet $\Sigma$ is denoted by $\Sigma^\omega$. Let $\sigma = \alpha_1\alpha_2\ldots$ be an infinite word over the alphabet $\Sigma$. We say that $\sigma$ is *accepted by* $\mathcal{B}$ if there exists a sequence of states $s_0 s_1 s_2 \ldots$, such that $(s_{i-1}, \alpha_i, s_i) \in \delta$ and some state from $F$ appears infinitely often in the sequence. The sequence of states $s_0 s_1 s_2 \ldots$ is called an *accepting run* (under Büchi acceptance condition $F$). Let $\mathcal{L}(\mathcal{B})$ be the language accepted by the Büchi automaton $\mathcal{B}$.

Assume that we are given a configuration $c$ and an LTL formula $\phi$. It is well known that, given an LTL formula over the atomic propositions in $AP$, there exists a Büchi automaton over the alphabet $\Sigma = 2^{AP}$ that accepts the same $\omega$-regular language. Moreover, there are efficient algorithms to translate an LTL formula into a Büchi automaton [18]. Let $\mathcal{B} = (\Sigma, Q, \delta, q_0, F)$ be the Büchi automaton corresponding to the LTL formula $\neg\phi$. The product of a PDS $\mathcal{P} = (P, \Gamma, \Delta)$ and $\mathcal{B}$ produces a *Büchi pushdown system* – a PDS augmented with a Büchi acceptance condition – $\mathcal{BP} = ((P \times Q), \Gamma, \Delta', G)$, where

- $\langle (p, q), \gamma\rangle \hookrightarrow \langle (p', q'), w\rangle \in \Delta'$ if $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$, $q \xrightarrow{\sigma} q'$, and $\sigma \subseteq \Omega(\langle p, \gamma\rangle)$.
- $(p, q) \in G$ if $q \in F$.

Given a Büchi pushdown system, the *accepting-run problem* is the problem of answering the question "Is there an accepting run starting from the configuration $\langle (p, q), \gamma\rangle$ – i.e., a run that visits infinitely often the configurations with control locations in $G$?".

LTL model checking answers the following question about a PDS $\mathcal{P} = (P, \Gamma, \Delta)$ and labeling function $\Omega$: "Does a configuration $c = \langle p, \gamma\rangle$ satisfy $\phi$?" LTL model checking can be reduced to the accepting-run problem as follows [17]:

- Construct the Büchi automaton $\mathcal{B} = (\Sigma, Q, \delta, q_0, F)$ corresponding to $\neg\phi$. An algorithm for computing a Büchi automaton corresponding to an LTL formula is given in [18].
- Compute $\mathcal{BP}$ as the product of the Büchi automaton $\mathcal{B} = (\Sigma, Q, \delta, q_0, F)$ and the PDS $\mathcal{P} = (P, \Gamma, \Delta)$ with respect to labeling function $\Omega$.
- A configuration $\langle p, w\rangle$ violates $\neg\phi$ iff there is an accepting run in $\mathcal{BP}$ starting from the configuration $\langle (p, q_0), \gamma\rangle$.

Esparza et al. [17] have shown that there is an accepting run starting from $\langle (p, q), \gamma\rangle$ iff the following condition holds:

$$\langle (p, q), \gamma\rangle \in pre^\star(R\Gamma^\star),$$

where $R$ is the set of $G$-repeating heads in $\mathcal{BP}$, and $R\Gamma^\star$ denotes the set of configurations $\langle (p', q'), \gamma' w\rangle$, where $\langle (p', q'), \gamma'\rangle$ is a repeating head. (Thus, the set $\mathit{pre}^\star(R\Gamma^\star)$ is the set of configurations that have a run leading to a configuration $\langle (p', q'), \gamma' v\rangle$, where $\langle (p', q'), \gamma'\rangle$ is a repeating head.)

The complexity of computing repeating heads is given in Theorem 3. Let the Büchi pushdown system $\mathcal{BP}$ be the product of a Büchi automaton $\mathcal{B} = (\Sigma, Q, \delta, q_0, F)$ and a PDS $\mathcal{P} = (P, \Gamma, \Delta)$. Let $n_P = |P|$, $n_\Delta = |\Delta|$, $n_Q = |Q|$, and $n_\delta = |\delta|$. Then $\mathcal{BP}$ has at most $s = n_P n_Q$ control locations and $r = n_\Delta n_\delta$ rules, and can be computed in $O(r)$ time. $R\Gamma^\star$ can be represented by an automaton of size $O(sr)$; consequently, by Theorem 1, LTL model checking can be performed in time and space $O(s^2 r)$ and $O(sr)$, respectively (i.e., $O(n_P^2 n_Q^2 n_\Delta n_\delta)$ and $O(n_P n_Q n_\Delta n_\delta)$, respectively).

Other algorithms for LTL model checking of PDSs, along with their time and space complexity, are discussed in [25, Section 3.2].

## 4. From SPKI/SDSI to PDSs

This section explains the connection between SPKI/SDSI and PDSs, and demonstrates how the authorization problem, as well as a variety of other certificate-set analysis problems, can be viewed as model-checking problems on PDSs.

Assume that we are given a set of certs $\mathcal{C}$ and a request $r = (K_P, T_P)$. We assume that all invalid name and auth certificates, as well as auth certs $(K, S, D, T, V)$ such that $T_P \not\subseteq T$, are removed from $\mathcal{C}$. We assume that useless certificates have been removed from $\mathcal{C}$. Let the set of keys and identifiers that appear in $\mathcal{C}$ be denoted by $\mathcal{K}_\mathcal{C}$ and $\mathcal{A}_\mathcal{C}$, respectively.

We construct a PDS $\mathcal{P}_\mathcal{C} = (P, \Gamma, \Delta)$ as follows:

- The set of locations is $P = \mathcal{K}_\mathcal{C}$; i.e., each key represents a control location of $\mathcal{P}_\mathcal{C}$.
- The stack alphabet is $\Gamma = \mathcal{A}_\mathcal{C} \cup \{\square, \blacksquare\}$; i.e., the stack alphabet is the set of identifiers, along with filled and unfilled squares (which encode delegation bits).
- The set of transition rules $\Delta$ contains a rule $\langle K, \gamma\rangle \hookrightarrow \langle K', w\rangle$ iff $(K\ \gamma \rightarrow K'\ w) \in \mathcal{C}$; i.e., the certs in $\mathcal{C}$ correspond to the transition rules.

Consider a term $N \in \mathcal{K} \cup \mathcal{N}$. The term $N = KA_1 \cdots A_m$ corresponds to the configuration $c(N) = \langle K, A_1 \cdots A_m\rangle$ in the PDS $\mathcal{P}_\mathcal{C}$. If $N = K$, then $c(N) = \langle K, \epsilon\rangle$. The lemma given below establishes a correspondence between the closure $\mathcal{C}^\star$ of the set of certs $\mathcal{C}$ and the reachability relation $\Rightarrow^\star$ in the PDS $\mathcal{P}_\mathcal{C}$.

**Lemma 4.** *Assume that we are given a set of certs $\mathcal{C}$. Let $\mathcal{P}_\mathcal{C}$ be the PDS corresponding to $\mathcal{C}$, and let $N$ be a term. For all terms $N'$, we have $N' \in \mathcal{C}^\star(N)$ if and only if $c(N) \Rightarrow^\star c(N')$ in $\mathcal{P}_\mathcal{C}$. In other words, we have the following equality*:

$$\mathcal{C}^\star(N) = \mathit{post}^\star(c(N))$$

**Proof.** The mapping $c(\cdot)$ provides a one-to-one mapping between the set of terms and the set of configurations of the PDS $\mathcal{P}_{\mathcal{C}}$. It follows from the construction of $\mathcal{P}_{\mathcal{C}}$ that $c(N) \Rightarrow c(N')$ iff there exists a rule $C_1 \in \mathcal{C}$ such that $C_1(N) = N'$. The result follows directly from the definitions. □

There are two options to solve the authorization problem "Is $K_P$ authorized to access resource $T_P$?": one uses *pre*$^\star$; the other uses *post*$^\star$. A "proof" of the authorization is a run of the PDS $\mathcal{P}_{\mathcal{C}}$ that starts at the configuration $\langle K_{\text{owner}[T_P]}, \square\rangle$ and ends at one of the configurations from the following set:

$$\{\langle K_P, \square\rangle, \langle K_P, \blacksquare\rangle\}.$$

In terms of *pre*$^\star$ and *post*$^\star$, the condition described above can be formalized by either of the following:

$$\langle K_{\text{owner}[T_P]}, \square\rangle \in \mathit{pre}^\star(\{\langle K_P, \square\rangle, \langle K_P, \blacksquare\rangle\}) \tag{1}$$

$$\mathit{post}^\star(\{\langle K_{\text{owner}[T_P]}, \square\rangle\}) \cap \{\langle K_P, \square\rangle, \langle K_P, \blacksquare\rangle\} \neq \emptyset \tag{2}$$

Algorithms based on conditions (1) and (2) will be referred to as $A_{\mathit{pre}}$ and $A_{\mathit{post}}$, respectively. The SPKI/SDSI algorithm described in Section 2.3 will be referred to as $A_{\text{SPKI/SDSI}}$. Based on Lemma 4, the following theorem is easy to prove:

**Theorem 5.**

1. *A principal $K_P$ is granted authorization to access resource $T_P$ by algorithm $A_{\mathit{SPKI/SDSI}}$ iff algorithm $A_{\mathit{pre}}$ grants authorization to $K_P$ to access $T_P$.*
2. *A principal $K_P$ is granted authorization to access resource $T_P$ by algorithm $A_{\mathit{SPKI/SDSI}}$ iff algorithm $A_{\mathit{post}}$ grants authorization to $K_P$ to access $T_P$.*

**Proof.** Algorithm $A_{\mathit{pre}}$ checks whether the following condition is true:

$$\langle K_{\text{owner}[T_P]}, \square\rangle \in \mathit{pre}^\star(\{\langle K_P, \square\rangle, \langle K_P, \blacksquare\rangle\}).$$

The statement given above is equivalent to the following one, which is the condition checked by $A_{\mathit{post}}$:

$$\mathit{post}^\star(\{\langle K_{\text{owner}[T_P]}, \square\rangle\}) \cap \{\langle K_P, \square\rangle, \langle K_P, \blacksquare\rangle\} \neq \emptyset.$$

In the SPKI/SDSI context, the latter condition is equivalent to the following condition (via Lemma 4):

$$\mathcal{C}^\star(K_{\text{owner}[T_P]}\ \square) \cap \{K_P\ \square, K_P\ \blacksquare\} \neq \emptyset.$$

This is exactly the question that algorithm $A_{\text{SPKI/SDSI}}$ answers. □

The algorithm $A_{pre}$ works as follows:

1. Construct the PDS $\mathcal{P}_{\mathcal{C}}$ corresponding to the set of certs $\mathcal{C}$ as described before.
2. Let $Y$ be the following set of configurations:

   $$\{\langle K_P, \square\rangle, \langle K_P, \blacksquare\rangle\}.$$

   Construct the automaton $A_Y = (\Gamma, Q, \delta, P, F)$, where $Q = P \cup \{s\}$, $\delta = \{(K_P, \square, s), (K_P, \blacksquare, s)\}$, and $F = \{s\}$. (*Conf*$(A_Y)$ is $Y$.) Using the algorithm described in Section 3.1, create the automaton corresponding to *pre*$^\star(Y)$.
3. Grant authorization to $K_P$ iff $\langle K_{\text{owner}[T_P]}, \square\rangle$ is accepted by the automaton for *pre*$^\star(Y)$.

The complexity of algorithm $A_{pre}$ can be analyzed as follows: The number of states $n_Q$ in the automaton $A_Y$ is $n_K + 1$. There is a one-to-one correspondence between the transition rules and the certs in the set $\mathcal{C}$; therefore, $n_\Delta$ is equal to $|\mathcal{C}|$. Moreover, the number of transitions $n_\delta$ in the automaton $A_Y$ is 2. Invoking Theorem 1, we obtain that the time and space complexity of $A_{pre}$ are $O(n_K^2|\mathcal{C}|)$ and $O(n_K|\mathcal{C}|)$, respectively. Notice that this is exactly the same asymptotic complexity that Clarke et al. obtain for algorithm $A_{\text{SPKI/SDSI}}$.

**Example 4.** Consider the example described in Section 2.2. Let $\mathcal{C}$ be the set of configurations shown in Fig. 1. Let $\mathcal{P}_{\mathcal{C}} = (P, \Gamma, \Delta)$ be the PDS corresponding to $\mathcal{C}$. In this case, the control locations $P$ and the stack alphabet $\Gamma$ are given by the following sets:

$$\{K_{\text{owner}[H]}, K_0, K_1, K_2, K_3, K_B, K_4, K_A\}$$
$$\{\texttt{UW}, \texttt{CS}, \texttt{faculty}, \texttt{Bob}, \texttt{Alice}, \square, \blacksquare\}$$

The transition rules $\Delta$ are shown in Fig. 8.

We are interested in an authorization for Alice, whose key is $K_A$. Consider the following set of configurations $X$:

$$\{\langle K_A, \square\rangle, \langle K_A, \blacksquare\rangle\}.$$

| | |
|---|---|
| $\langle K_{\text{owner}[H]}, \square\rangle \hookrightarrow \langle K_0, \texttt{UW CS faculty}\ \square\rangle$ | (1) |
| $\langle K_0, \texttt{UW}\rangle \hookrightarrow \langle K_1, \epsilon\rangle$ | (2) |
| $\langle K_1, \texttt{CS}\rangle \hookrightarrow \langle K_2, \epsilon\rangle$ | (3) |
| $\langle K_2, \texttt{faculty}\rangle \hookrightarrow \langle K_3, \texttt{Bob}\rangle$ | (4) |
| $\langle K_3, \texttt{Bob}\rangle \hookrightarrow \langle K_B, \epsilon\rangle$ | (5) |
| $\langle K_B, \square\rangle \hookrightarrow \langle K_4, \texttt{Alice}\ \blacksquare\rangle$ | (6) |
| $\langle K_4, \texttt{Alice}\rangle \hookrightarrow \langle K_A, \epsilon\rangle$ | (7) |

Fig. 8. The set of transition rules $\Delta$ in $\mathcal{P}_{\mathcal{C}}$.

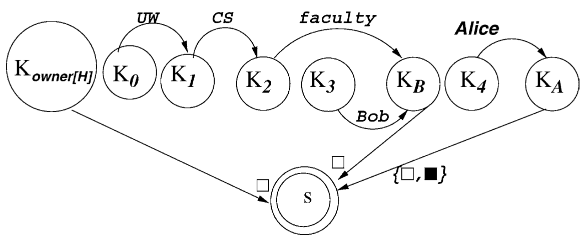

Fig. 9. Automaton that accepts the set of configurations $\mathit{pre}^\star(\{\langle K_A, \Box\rangle, \langle K_A, \blacksquare\rangle\})$.

A configuration automaton $A_X = (\Gamma, Q, \delta, P, F)$ that accepts $X$ can be defined as follows: $Q = P \cup \{s\}$, $\delta = \{(K_A, \Box, s), (K_A, \blacksquare, s)\}$, and $F = \{s\}$. The automaton that would be constructed for $\mathit{pre}^\star(X)$ is shown in Fig. 9. Note that the configuration $\langle K_{\text{owner}[H]}, \Box\rangle$ is accepted by the automaton, and thus principal $K_A$ (i.e., Alice) is authorized to login to host $H$.

Next, we describe in detail the algorithm $A_{\mathit{post}}$ that uses the construction $\mathit{post}^\star$. Again, suppose that we are interested in determining whether a principal $K_P$ is authorized to access resource $T_P$, given a set of certificates $\mathcal{C}$. The algorithm for solving the authorization problem is as follows:

1. Construct the PDS $\mathcal{P}_\mathcal{C}$ corresponding to the set of certs $\mathcal{C}$.
2. Let $S$ be the following set of configurations: $\{\langle K_{\text{owner}[T_P]}, \Box\rangle\}$. Construct the automaton $A_S = (\Gamma, Q, \delta, P, F)$, where $Q = P \cup \{s\}$, $\delta = \{(K_{\text{owner}[T_P]}, \Box, s)\}$, and $F = \{s\}$. ($\mathit{Conf}(A_S)$ is $S$.) Before computing $\mathit{post}^\star(S)$, we need to transform the PDS $\mathcal{P}_\mathcal{C}$ so that all transition rules $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$ satisfy $|w| \leqslant 2$. Then add a new state $q_{p',\gamma'}$ for each rule right-hand side of the form $\langle p', \gamma'\gamma''\rangle$. Finally, complete the construction of the automaton for $\mathit{post}^\star(S)$ by repeatedly applying the saturation rule.
3. Grant authorization to $K_P$ iff $\langle K_P, \Box\rangle$ or $\langle K_P, \blacksquare\rangle$ is accepted by the automaton corresponding to $\mathit{post}^\star(S)$.

We analyze the complexity of this algorithm as follows: the number of states $n_Q$ in the automaton $A_S$ is $n_K + 1$. Using Theorem 2, we obtain that the time and space complexity of $A_{\mathit{post}}$ are both $O(n_K|\mathcal{C}|^2)$.[6]

**Example 5.** Consider the set of certs $\mathcal{C}$ shown in Fig. 1. The PDS $\mathcal{P}_\mathcal{C} = (P, \Gamma, \Delta)$ corresponding to the set of certs $\mathcal{C}$ was explicitly constructed in Example 4 and Fig. 8.

[6]Whenever each principal signs at least one certificate, $n_K \leqslant |\mathcal{C}|$, and therefore the time and space complexity of $A_{\mathit{post}}$ is worse than that of $A_{\mathit{pre}}$. However, there are two reasons why $A_{\mathit{post}}$ is of interest: (i) it can serve as a subroutine in the algorithm for LTL model checking, which provides a way to answer a general class of certificate-set-analysis questions [25, Section 3.2.3], and (ii) in other kinds of PDS model-checking problems, it has been found that, in practice, $\mathit{post}^\star$ works faster than $\mathit{pre}^\star$ [25]. It remains to be seen whether this is also true for certificate-set-analysis problems.

Recall that the *post*$^\star$ algorithm assumes that every transition rule $\langle p,\gamma\rangle \hookrightarrow \langle p',w\rangle$ satisfies $|w| \leqslant 2$. The following rule in $\mathcal{P}_\mathcal{C}$ does not satisfy that constraint:

$$\langle K_{\text{owner}[H]}, \square\rangle \hookrightarrow \langle K_0, \texttt{UW}\ \texttt{CS}\ \texttt{faculty}\ \square\rangle$$

We transform the PDS by (i) adding two new locations $K_0^1$ and $K_0^2$, (ii) adding the following three rules, and (iii) deleting the rule given above.

$$\begin{aligned}\langle K_{\text{owner}[H]}, \square\rangle &\hookrightarrow \langle K_0^1, \texttt{faculty}\ \square\rangle\\ \langle K_0^1, \texttt{faculty}\rangle &\hookrightarrow \langle K_0^2, \texttt{CS}\ \texttt{faculty}\rangle\\ \langle K_0^2, \texttt{CS}\rangle &\hookrightarrow \langle K_0, \texttt{UW}\ \texttt{CS}\rangle\end{aligned}$$

One of the original transition rules of the PDS has two stack symbols on the right-hand side: $\langle K_B, \square\rangle \hookrightarrow \langle K_4, \texttt{Alice}\ \blacksquare\rangle$.

A configuration automaton $\mathcal{A}_S = (\Gamma, Q, \delta, P, F)$ that accepts the set $S = \{\langle K_{\text{owner}[H]}, \square\rangle\}$ can be defined as follows: $Q = P \cup \{s\}$, $F = \{s\}$, and $\delta = \{(K_{\text{owner}[H]}, \square, s)\}$. After Phase I of the construction from Section 3.2, the automaton has the following components

$$Q = P \cup \{K_0^1, K_0^2, q_{K_0^1,\texttt{faculty}}, q_{K_0^2,\texttt{CS}}, q_{K_0,\texttt{UW}}, q_{K_4,\texttt{Alice}}, s\}\ ,$$

$F = \{s\}$, and $\delta = \{(K_{\text{owner}[H]}, \square, s)\}$. The automaton that would be constructed for *post*$^\star(S)$ is shown in Fig. 10. Note that the configuration $\langle K_A, \blacksquare\rangle$ is accepted by this automaton, and thus principal $K_A$ (i.e., Alice) is authorized to login to host $H$.

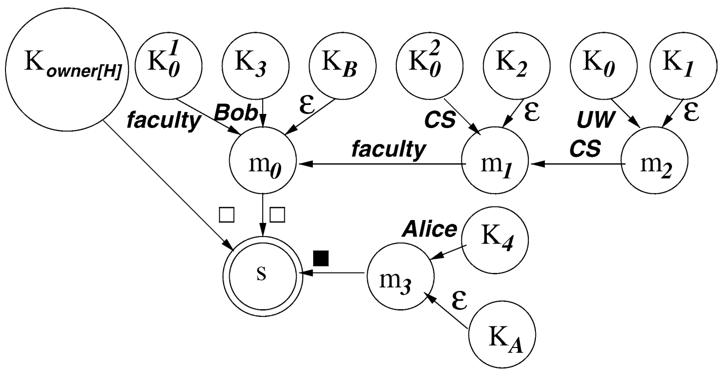

Fig. 10. Automaton that accepts the set of configurations *post*$^\star(S)$. In the figure, $m_0$, $m_1$, $m_2$, and $m_3$ stand for $q_{K_0^1,\texttt{faculty}}$, $q_{K_0^2,\texttt{CS}}$, $q_{K_0,\texttt{UW}}$, and $q_{K_4,\texttt{Alice}}$, respectively.

### 4.1. Certificate-chain reconstruction

We now describe how we can augment the automaton constructed by algorithm $A_{pre}$ with extra information for the purpose of certificate-chain reconstruction. ($A_{post}$ can be augmented similarly.)

The automaton for $pre^\star(X)$ is created by adding transitions to $A_X$, until no more transitions can be added, according to the saturation rule

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

With each transition $(p,\gamma,q)$ of the configuration automaton, we associate a structure with two components: an integer identifier and a list of transitions. Suppose that a transition $(p,\gamma,q)$ is added due to the PDS transition rule $\langle p,\gamma\rangle \hookrightarrow \langle p',w\rangle$ and the fact that a path $p' \xrightarrow{w} q$ holds in the automaton. Let path $p' \xrightarrow{w} q$ consist of the (possibly empty) sequence of transitions $t_1t_2\cdots t_n$. The structure associated with the transition $(p,\gamma,q)$ is

$$(id(\langle p,\gamma\rangle \hookrightarrow \langle p',w\rangle), [t_1t_2\cdots t_n]),$$

where the identifier of a PDS transition rule $r$ is denoted by $id(r)$. The collection of such structures forms a dag.[7]

We will explain how to construct a certificate chain from such structures using the example from Section 2.2, for which the final configuration automaton is given in Fig. 9. The structures associated with the transitions of the automation from Fig. 9 are shown in Fig. 11. An empty structure and an empty list are denoted by $\epsilon$ and NULL, respectively.

When a request $r = (K,T)$ succeeds, the final configuration automaton for $pre^\star(\{\langle K,\Box\rangle, \langle K,\blacksquare\rangle\})$ must contain the transition $(K_{\text{owner}[T]},\Box,s)$, where $s$ is the automaton's accepting state. When the $pre^\star$ algorithm has been extended as described above, a certificate chain for request $r$ can be obtained from the auxiliary structure associated with $(K_{\text{owner}[T]},\Box,s)$ in the final configuration automaton.

For instance, Fig. 9 contains the transition $t_8 = (K_{\text{owner}[H]},\Box,s)$, which has the associated structure $(1,[t_5t_4t_7t_6])$. Certificate-chain reconstruction can be performed by flattening the dag rooted at $t_8$, to produce a preorder listing – with repetitions – of the rule identifiers in the dag. Let the symbol $||$ denote list concatenation.

- If $t_i$'s structure is $\epsilon$, flatten($t_i$) = ().
- If $t_i$'s structure is ($j$, NULL), flatten($t_i$) = ($j$).

[7]The method described above keeps enough information to recover one certificate chain. In general, one can associate a *set* of pairs with each automaton transition, where a pair consists of an integer identifier (corresponding to the rule) and a list of transitions (corresponding to a path in the automaton). Each pair in the set associated with a transition $(p,\gamma,q)$ represents a way for the saturation rule to derive $(p,\gamma,q)$. This provides a way to recover a dag of certificate chains.

| | transition | structure |
|---|---|---|
| $t_0$ | $(K_A, \square, s)$ | $\epsilon$ |
| $t_1$ | $(K_A, \blacksquare, s)$ | $\epsilon$ |
| $t_2$ | $(K_4, \texttt{Alice}, K_A)$ | $(7, \text{NULL})$ |
| $t_3$ | $(K_3, \texttt{Bob}, K_B)$ | $(5, \text{NULL})$ |
| $t_4$ | $(K_1, \texttt{CS}, K_2)$ | $(3, \text{NULL})$ |
| $t_5$ | $(K_0, \texttt{UW}, K_1)$ | $(2, \text{NULL})$ |
| $t_6$ | $(K_B, \square, s)$ | $(6, [t_2 t_1])$ |
| $t_7$ | $(K_2, \texttt{faculty}, K_B)$ | $(4, [t_3])$ |
| $t_8$ | $(K_{\text{owner}[H]}, \square, s)$ | $(1, [t_5 t_4 t_7 t_6])$ |

Fig. 11. Structures associated with the transitions of the automaton shown in Fig. 9.

- If $t_i$'s structure is $(j, [t_i^1, \ldots, t_i^{k_i}])$, flatten($t_i$) $=$ $(j)$ $||$ flatten($t_i^1$) $||$ $\ldots$ $||$ flatten($t_i^{k_i}$).

(Because flattening a dag may create a list that is exponentially larger than the size of the dag, in practice it may be better for a system to manipulate certificate dags directly.)

For Fig. 9, flatten($t_8$) produces the certificate chain (1 2 3 4 5 6 7), which proves that Alice has the proper authorization for her request.

## 4.2. *Threshold subjects*

The document that defines SPKI/SDSI [16] provides for an additional kind of subject in name and auth certs: a *threshold subject* is a subject of the form $\theta_k(S_1, S_2, \ldots, S_m)$, where $1 \leqslant k \leqslant m$. The value $k$ is a threshold value, and a threshold subject specifies $k$ of the $m$ subjects $S_1, S_2, \ldots, S_m$. Although the SPKI/SDSI defining document permits threshold subjects in both name and auth certs, Clarke et al. restrict them to just auth certs [12, Section 10], observing

> The reason that a threshold subject may not appear in a name cert is that a name cert is used to define a name as a set of public keys; if a name cert could have a threshold subject as a subject then the notion of the value of a name would have to be generalized from a set of keys to a set of sets of keys, which would almost surely be too convoluted to be usable in practice.

In this section, we also adopt this restriction.[8]

As in [12, Section 10], an auth cert that uses a threshold subject, such as

$$K \ \square \longrightarrow \theta_k(S_1, S_2, \ldots, S_m) \ \square, \tag{3}$$

[8]Certificate-set analysis for certificates that use threshold subjects in name certs could be handled using model checking of alternating pushdown systems [8].

would be normalized by introducing $m$ new keys to serve as placeholders, and replacing rule (3) by the set of rules

$$\begin{aligned}
K\ \square &\longrightarrow \theta_k(\hat{K}_1, \hat{K}_2, \ldots, \hat{K}_m)\ \square\\
\hat{K}_1\ \square &\longrightarrow S_1\ \square\\
\hat{K}_2\ \square &\longrightarrow S_2\ \square\\
&\ \ \vdots\\
\hat{K}_m\ \square &\longrightarrow S_m\ \square
\end{aligned}$$

(If the right-hand side of rule (3) used $\blacksquare$, the $m$ rules for the $\hat{K}_i$ would have the form $\hat{K}_i\ \square \longrightarrow S_i\ \blacksquare$.)

As usual, to determine if principal $K_P$ has authorization to access $T$, we start with a configuration automaton that accepts the language $\{\langle K_P, \square\rangle, \langle K_P, \blacksquare\rangle\}$, and check whether $\langle K_{\text{owner}[T]}, \square\rangle$ is accepted by the final configuration automaton. However, the *pre*$^\star$ algorithm is changed to implement the following saturation rules:

- If $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$ and $p' \overset{w}{\to} q$ in the current automaton, add a transition $(p, \gamma, q)$.
- If $r = \langle p, \square\rangle \hookrightarrow \langle \theta_k(p_1, p_2, \ldots, p_m), \square\rangle$, and there are $k$ transitions of the form $(p_i, \square, q)$ in the current automaton, add a transition $(p, \square, q)$.

In the initial configuration automaton, the only transitions labeled with $\square$ or $\blacksquare$ are ones to the final state. No transitions can be added that are labeled with $\blacksquare$, and when transitions are added with the label $\square$, these must all be to the final state. Moreover, because the $\hat{K}_i$ symbols are new, transitions of the form $(\hat{K}_i, \blacksquare, s)$ and $(\hat{K}_i, \square, s)$ can only contribute to saturation steps of the second type. Thus, the only changes to the *pre*$^\star$ algorithm are that it needs to maintain a counter for each threshold rule $r$; these counters are initialized to 0; the algorithm must identify the first time a transition of the form $(\hat{K}_i, \blacksquare, s)$ or $(\hat{K}_i, \square, s)$ is generated, at which point the counter for rule $r$ is incremented; a transition $(p, \square, s)$ is generated when the value of the counter reaches $k$. The costs in time and space to perform these operations are dominated by the other costs of the *pre*$^\star$ algorithm, so this does not change the overall complexity of certificate-set analysis via algorithm $A_{pre}$.

## 4.3. Handling multiple requests

For certain certificate-set-analysis problems, we need to consider multiple requests (e.g., see "Shared access 3" and "Compromisation assessment 1" in Section 4.4). We describe how the authorization framework can be extended to handle multiple requests. Assume that we are given $m$ requests $r_1 = (K_1, T_1), \ldots, r_m = (K_m, T_m)$.

1. **Remove useless certificates**
   All invalid name and auth certificates are removed from the set $\mathcal{C}$.

2. **Encoding multiple requests**
   As before, each name cert $(K, A, S, V)$ in $\mathcal{C}$ corresponds to a rewrite rule of the form $KA \longrightarrow S$. However, each auth cert $(K, S, D, T, V)$ in $\mathcal{C}$ can generate more than one rewrite rule. For each auth cert $(K, S, D, T, V)$ in $\mathcal{C}$, perform the following steps:

```
**for** $1 \leqslant i \leqslant m$
    **if** $T_i \subseteq T$ **then**
        Generate a rewrite rule $\begin{cases} K\,\square_i \longrightarrow S\,\square_i & \text{if } D \text{ is on} \\ K\,\square_i \longrightarrow S\,\blacksquare_i & \text{otherwise} \end{cases}$
    **end if**
```

   Essentially, $\square_i$ and $\blacksquare_i$ represent access to resource $T_i$ – with and without delegation, respectively. The set of rewrite rules generated by the algorithm described above is denoted by $\mathcal{C}_R$.

Let $\mathcal{P}_{\mathcal{C}_R}$ be the PDS corresponding to $\mathcal{C}_R$. Note that the stack alphabet of $\mathcal{P}_{\mathcal{C}_R}$ is

$$\mathcal{A}_{\mathcal{C}_R} \cup \{\square_1, \cdots, \square_m, \blacksquare_1, \cdots, \blacksquare_m\}.$$

Algorithms $A_{pre}$ and $A_{post}$ are extended to handle multiple requests, as follows. For $A_{pre}$, we compute the following set of configurations:

$$pre^\star\left(\bigcup_{i=1}^{m}\{\langle K_i, \square_i\rangle, \langle K_i, \blacksquare_i\rangle\}\right).$$

Request $r_i$ is granted iff $\langle K_{\text{owner}[T_i]}, \square\rangle$ is in the set given above. Similarly, for $A_{post}$ we compute

$$post^\star\left(\bigcup_{i=1}^{m}\{\langle K_{\text{owner}[T_i]}, \square_i\rangle\}\right).$$

Request $r_i$ is granted iff $\langle K_i, \square_i\rangle$ or $\langle K_i, \blacksquare_i\rangle$ is in the set given above.

To discuss the time and space complexity of the extended algorithms, let $\mathcal{C}_N$ and $\mathcal{C}_A$ be the sets of name and auth certs in $\mathcal{C}_R$, respectively. Because each auth cert can generate at most $m$ rewrite rules (one for each request), the size of the set $\mathcal{C}_R$ is bounded by $|\mathcal{C}_N| + m|\mathcal{C}_A|$. Therefore, the time and space complexity of algorithm $A_{pre}$ is $O(n_K^2(|\mathcal{C}_N| + m|\mathcal{C}_A|))$ and $O(n_K(|\mathcal{C}_N| + m|\mathcal{C}_A|)$, respectively. Similarly, both the time and space complexity of algorithm $A_{post}$ are $O(n_K(|\mathcal{C}_N| + m|\mathcal{C}_A|)^2)$. In both cases, this is more efficient than running the algorithms $m$ times, once for each request. The basic intuition is that certain work for processing the set of name certs is shared when requests are processed simultaneously; the algorithm performs saturation steps due (solely) to name certs only once.

**Note:** Multiple requests can also be handled using the name-reduction-closure-based algorithm of Clarke et al. as follows: compute the name-reduction closure $\mathcal{C}_R^\sharp$ for the set $\mathcal{C}_R$. Remove all the rules from $\mathcal{C}_R^\sharp$ not of the form $K' \ \square_i \longrightarrow K'' \ \square_i$ and $K' \ \square_i \longrightarrow K'' \ \blacksquare_i$. Next we partition the rules into $m$ sets $\mathcal{C}_1, \cdots, \mathcal{C}_m$, where $\mathcal{C}_i$ contains all the rules of the following form:

$$\begin{array}{l} K' \ \square_i \longrightarrow K'' \ \square_i \ \text{ and} \\ K' \ \square_i \longrightarrow K'' \ \blacksquare_i \end{array}$$

Intuitively, $\mathcal{C}_i$ contains all the rules pertaining to request $r_i$. Now the validity of request $r_i$ can be determined by the depth-first-search algorithm described earlier, where rules $\mathcal{C}_i$ to construct the directed graph.

### *4.4. Certificate-set-analysis problems*

This section discusses applications of model checking to specific certificate-set-analysis problems; in particular, we show how model checking furnishes algorithms for the analysis problems listed in the introduction. (Here, we use the term “model checking” to mean both (i) the problem of checking whether a given PDS satisfies a given LTL formula, and (ii) the problem of answering simple forward and backward reachability queries; the latter can be stated in terms of set-former expressions that use the basic automaton-building operations *pre*$^*$ and *post*$^*$.) Given a set of certs $\mathcal{C}$ and a set of configurations $X$, we write *pre*$^\star[\mathcal{P}_\mathcal{C}](X)$ as *pre*$^\star[C](X)$. Similarly, *post*$^\star[\mathcal{P}_\mathcal{C}]$ is written as *post*$^\star[\mathcal{C}]$. In analysis problems involving multiple resources we use the encoding given in Section 4.3.

**Authorized access 1:** Given resource $R$ and principal $K$, is $K$ authorized to access $R$?

$$\begin{array}{l} \langle K_{\text{owner}[R]}, \square\rangle \in pre^*(\{\langle K, \square\rangle, \langle K, \blacksquare\rangle\}) \text{ or, alternatively,} \\ \{\langle K, \square\rangle, \langle K, \blacksquare\rangle\} \cap post^*(\{\langle K_{\text{owner}[R]}, \square\rangle\}) \neq \emptyset \end{array}$$

**Authorized access 2:** Given resource $R$ and name $N$ (not necessarily a principal), is $N$ authorized to access $R$?

$$\begin{array}{l} \langle K_{\text{owner}[R]}, \square\rangle \in pre^*(\{c(N.\square), c(N.\blacksquare)\}) \text{ or, alternatively,} \\ \{c(N.\square), c(N.\blacksquare)\} \cap post^*(\{\langle K_{\text{owner}[R]}, \square\rangle\}) \neq \emptyset \end{array}$$

In the expression given above $N.\square$ denotes the term obtained by concatenating $\square$ to $N$.

**Authorized access 3:** Given resource $R$, what names are authorized to access $R$?

$$post^*(\{\langle K_{\text{owner}[R]}, \square\rangle\})$$

**Shared access 1:** For two given resources $R_1$ and $R_2$, what principals can access both $R_1$ and $R_2$?

$$\begin{aligned} &\{K \mid \langle K, \square\rangle \text{ or } \langle K, \blacksquare\rangle \text{ is in } \mathit{post}^\star[\mathcal{C}](\{\langle K_{\text{owner}[R_1]}, \square\rangle\})\} \\ \cap\ &\{K \mid \langle K, \square\rangle \text{ or } \langle K, \blacksquare\rangle \text{ is in } \mathit{post}^\star[\mathcal{C}](\{\langle K_{\text{owner}[R_2]}, \square\rangle\})\} \end{aligned}$$

**Shared access 2:** Given two principals, $K_1$ and $K_2$, and a resource $R$, can both $K_1$ and $K_2$ access $R$? This question can be answered by checking if $K_1$ and $K_2$ are both in the following set:

$$\{K \mid \langle K, \square\rangle \text{ or } \langle K, \blacksquare\rangle \text{ is in } \mathit{post}^\star[\mathcal{C}](\{\langle K_{\text{owner}[R]}, \square\rangle\})\}$$

**Shared access 3:** For two given principals, $K_1$ and $K_2$, and a finite set of resources $\mathcal{R} = \{R_1, \ldots, R_k\}$, which resources from $\mathcal{R}$ can be accessed by both $K_1$ and $K_2$?

$$\left\{ R_i \in \mathcal{R} \,\middle|\, \langle K_{\text{owner}[R_i]}, \square_i\rangle \in \begin{pmatrix} \mathit{pre}^*(\bigcup_{i=1}^k \{\langle K_1, \square_i\rangle, \langle K_1, \blacksquare_i\rangle\}) \\ \cap\, \mathit{pre}^*(\bigcup_{i=1}^k \{\langle K_2, \square_i\rangle, \langle K_2, \blacksquare_i\rangle\}) \end{pmatrix} \right\}$$

**Compromisation assessment 1:** What resources from a finite set $\mathcal{R} = \{R_1, \ldots, R_k\}$ could principal $K$ have gained access to (solely) due to the presence of maliciously or accidentally issued certificate set $\mathcal{C}' \subseteq \mathcal{C}$?

$$\left\{ R_i \in \mathcal{R} \,\middle|\, \langle K_{\text{owner}[R_i]}, \square_i\rangle \in \begin{pmatrix} \mathit{pre}^\star[\mathcal{C}](\bigcup_{i=1}^k \{\langle K, \square_i\rangle, \langle K, \blacksquare_i\rangle\}) \\ -\mathit{pre}^\star[\mathcal{C} - \mathcal{C}'](\bigcup_{i=1}^k \{\langle K, \square_i\rangle, \langle K, \blacksquare_i\rangle\}) \end{pmatrix} \right\}$$

**Compromisation assessment 2:** What principals could have gained access to resource $R$ (solely) due to the presence of maliciously or accidentally issued certificate set $\mathcal{C}' \subseteq \mathcal{C}$?

$$\begin{aligned} &\{K \mid \langle K, \square\rangle \text{ or } \langle K, \blacksquare\rangle \text{ is in } \mathit{post}^\star[\mathcal{C}](\{\langle K_{\text{owner}[R]}, \square\rangle\})\} \\ -\ &\{K \mid \langle K, \square\rangle \text{ or } \langle K, \blacksquare\rangle \text{ is in } \mathit{post}^\star[\mathcal{C} - \mathcal{C}'](\{\langle K_{\text{owner}[R]}, \square\rangle\})\} \end{aligned}$$

**Expiration vulnerability 1:** What resources from a finite set of resources $\{R_1, \ldots, R_k\}$ will principal $K$ be prevented from accessing if certificate set $\mathcal{C}' \subseteq \mathcal{C}$ expires?
Same as compromisation assessment 1.

**Expiration vulnerability 2:** What principals will be excluded from accessing resource $R$ if certificate set $\mathcal{C}' \subseteq \mathcal{C}$ expires?
Same as compromisation assessment 2.

**Universally guarded access 1:** Is it the case that all authorizations that can be issued for a given resource $R$ must involve a certificate signed by principal $K$?
For this, we use LTL model checking, with the labeling Ω defined as follows:

- All surface configurations that involve location $K$ are labeled with atomic proposition $Q$.
- All surface configurations $\langle K', \square\rangle$ and $\langle K', \blacksquare\rangle$, such that $K' \in \mathcal{K}$, are labeled with atomic proposition $S$.

We then ask whether configuration $\langle K_{\text{owner}[R]}, \square\rangle$ satisfies the LTL formula

$$\square(\neg S\ \mathcal{U}\ (Q \vee \square\neg S)) \tag{4}$$

The LTL formula given above describes two types of runs of the PDS:

1. Runs in which $\neg S$ is true globally on the run. These represent chains where nobody is authorized to access the resource $R$.
2. Runs in which $S$ is true but $Q$ is true before $S$ becomes true. These represent chains where some principal does receive an authorization, but a certificate issued by principal $K$ is utilized.

**Universally guarded access 2:** Is it the case that all authorizations that grant a given principal $K'$ access to a finite set of resources $\{R_1, \ldots, R_k\}$ must involve a certificate signed by $K$?
We again use LTL formula (4). In this case, the labeling Ω is defined as follows:

- All surface configurations that involve location $K$ are labeled with atomic proposition $Q$.
- The surface configurations of the form $\langle K', \square_i\rangle$ and $\langle K', \blacksquare_i\rangle$ are labeled with atomic proposition $S$.

We then ask whether every surface configuration of the form $\langle K_{\text{owner}[R_i]}, \square_i\rangle$ satisfies LTL formula (4).

## *4.5. Efficiency of the automaton representation*

Clarke et al. [12] give a "worst case" example for their name-reduction-closure algorithm. We will use their example to illustrate the efficiency of the automaton representation of a set of configurations. Consider the following set of certificates $\mathcal{C}$:

| | |
|---|---|
| $K\ \mathtt{C} \rightarrow K_0\ \mathtt{A}^l\ \mathtt{B}_j$ | (for $0 \leqslant j < n$) |
| $K_0\ \mathtt{A} \rightarrow K_i$ | (for $0 \leqslant i < n$) |
| $K_i\ \mathtt{A} \rightarrow K_{(i+1) \bmod n}\ \mathtt{A}$ | (for $0 \leqslant i < n$) |

In the rules given above, $\mathtt{A}^l$ represents the string $\mathtt{A} \cdots \mathtt{A}$ of length $l$. Name-reduction closure yields the following $n^2(l+1) + n^2$ rules:

$$\begin{array}{ll} K\ \mathtt{C} \to K_i\ \mathtt{A}^k\ \mathtt{B}_j & (\text{for } 0 \leqslant i < n, 0 \leqslant j < n, 0 \leqslant k \leqslant l) \\ K_i\ \mathtt{A} \to K_j & (\text{for } 0 \leqslant i < n \text{ and } 0 \leqslant j < n) \end{array}$$

Let $\mathcal{P}_{\mathcal{C}}$ be the PDS corresponding to the set of certificates $\mathcal{C}$. It is true that *post*$^\star(\{\langle K, \mathtt{C}\rangle\})$ is equal to the following set of configurations:

$$\{\langle K_i, \mathtt{A}^k\mathtt{B}_j\rangle \mid \text{for } 0 \leqslant i < n, 0 \leqslant j < n, \text{ and } 0 \leqslant k \leqslant l\},$$

and, therefore, the size of the set *post*$^\star(\{\langle K, \mathtt{C}\rangle\})$ is $n^2(l+1)$. However, the automaton representation of *post*$^\star(\{\langle K, \mathtt{C}\rangle\})$ is only of size $O(nl)$. The basic idea is the following: given a pair of keys $K_i$ and $K_j$ and a stack configuration $w$, the configuration $\langle K_i, w\rangle$ is in *post*$^\star(\{\langle K, \mathtt{C}\rangle\})$ iff $\langle K_j, w\rangle$ is. In the automaton representation, such commonalities are captured by means of sharing. In particular, the automaton accepting the set of configurations *post*$^\star(\{\langle K, \mathtt{C}\rangle\})$ has states that represent the stack configurations $\mathtt{A}^k\mathtt{B}_j$, and various locations (representing the keys $K_i$) have $\epsilon$-edges and $\mathtt{A}$-edges pointing to those shared states.

We now provide a worst-case example for a PDS-based algorithm, in particular for the backward algorithm *pre*$^\star$.[9] Since there is a one-to-one correspondence between PDSs and a SPKI/SDSI system, the worst-case example can easily be recast in terms of certificates.

Consider the PDS $\mathcal{P} = (P, \Gamma, \Delta)$, where $P = \{p_0, \ldots, p_{n-1}\}$, $\Gamma = \{y\}$, and $\Delta$ contains the following transitions (for $0 \leqslant i < n$):

$$\begin{array}{l} \langle p_i, y\rangle \hookrightarrow \langle p_{(i+1) \bmod n}, \epsilon\rangle \\ \langle p_i, y\rangle \hookrightarrow \langle p_i, yy\rangle \end{array}$$

The size of the PDS $\mathcal{P}$ is $n$. Suppose we want to compute *pre*$^\star(\{\langle p_0, \epsilon\rangle\})$. The $\epsilon$-rules create a cycle between the locations, and the second rule turns the automaton into a clique. The resulting automaton has $\Theta(n^2)$ transitions, and the *pre*$^\star$ algorithm uses time $\Theta(n^3)$ (because every automaton transition can be created in $n$ different ways).

## *4.6. Semiring labelings*

This section discusses how annotating the PDS and the configuration automaton with labels from a bounded idempotent semiring can answer several useful questions, such as, "How long does a specific authorization last?" and "What is the trust level associated with an authorization?".[10] It also describes an alternative way to handle authorization specifications, which has certain advantages over previous proposals.

[9]We thank Stefan Schwoon for providing us with this example.

[10]A more complete treatment of this subject, including a justification of the necessary extensions to algorithm $A_{pre}$ and certificate-chain reconstruction, can be found in [26].

**Definition 2.** 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 operations on $D$ such that

1. $(D, \oplus)$ is a commutative monoid with 0 as its neutral element, and where $\oplus$ is idempotent (i.e., for all $a \in D$, $a \oplus a = a$).
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

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

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: for all $a, b \in D$, $a \sqsubseteq b$ iff $a \oplus b = a$, there are no infinite descending chains.

Let each cert $c$ in the set $\mathcal{C}$ be annotated with a label $l(c)$ from a semiring $\mathcal{R}$. Let $\mathcal{P}_\mathcal{C}$ be the PDS corresponding to $\mathcal{C}$. A transition rule $r$ of PDS $\mathcal{P}_\mathcal{C}$ has the label of the corresponding cert. Recall that the algorithm $A_{pre}$ constructs an automaton for $pre^\star(X)$, where $X$ is the following set of configurations:

$$\{\langle K_P, \square\rangle, \langle K_P, \blacksquare\rangle\}.$$

We start with the automaton $A_X$ that accepts the set of configurations $X$. Each transition $(p, \gamma, q)$ of the automaton will also be labeled with an element from the semiring $\mathcal{R}$ (denoted by $l(p, \gamma, q)$). Initially, all of the transitions in $A_X$ are labeled with 1. We add transitions $(p, \gamma, q)$ to $A_X$ according to the following saturation rule:

If $r = \langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$ and there is a path for string $w$ from $p'$ to $q$ with cost $c$ in the current automaton, either (i) introduce a transition $(p, \gamma, q)$ if the automaton does not already contain such a transition, or (ii) change the label on $(p, \gamma, q)$ if $(p, \gamma, q)$ already occurs in the automaton. The label of transition $(p, \gamma, q)$ is computed as follows:

$$\begin{array}{ll} l(r) \otimes c & \text{if } (p,\gamma,q) \text{ is a new transition} \\ (l(r) \otimes c) \ \oplus \ l(p,\gamma,q) & \text{otherwise} \end{array}$$

The cost of a path in the automaton is computed by taking the $\otimes$ of the labels on the transitions along the path.

Semirings for the two cases discussed below are shown in Table 2.

**Certificate chains with maximal trust levels.** For this problem, 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 certificate $c$.

The elements of the semiring represent trust levels, where low, medium, and high trust levels are denoted by $L$, $M$, and $H$, respectively. A fourth value, $Z$, is added as

Table 2

Semirings for trust and validity

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

the 0 element. $Z$, $L$, $M$, and $H$ form a totally ordered set, where $Z \sqsupseteq L \sqsupseteq M \sqsupseteq H$.[11]

The semiring operations $\otimes$ and $\oplus$ correspond to join ($\sqcup$) and meet ($\sqcap$) on the totally ordered set $\{Z, L, M, H\}$:

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

Referring back to the example from Section 2.2, for which the final automaton is given in Fig. 9, if a trust level drawn from $\{Z, L, M, H\}$ were assigned to each certificate, the value that labels the transition $(K_{\text{owner}[H]}, \square, s)$ in the automaton constructed by the algorithm $A_{pre}$ represents the "trust level" of Alice's authorization. The answer reported can then be used by the reference monitor for resource $H$ to make authorization decisions. The value that labels $(K_{\text{owner}[H]}, \square, s)$ represents the trust level of the certificate chain that has the *least* trust value in the ordering $Z \sqsupseteq L \sqsupseteq M \sqsupseteq H$ – where lower in the ordering corresponds to "higher level of trust" in the sense of everyday speech. For instance, if the label of $(K_{\text{owner}[H]}, \square, s)$ is $H$, then there exists a certificate chain such that (i) the chain justifies granting authorization to Alice, and (ii) all of the chain's labels have the value $H$.

**Maximally valid certificate chains.** Let $V(c)$ be the expiration value of cert $c$; i.e., cert $c$ will expire at time $T_{current} + V(c)$, where $T_{current}$ is the current time. The expiration value of a certificate chain $(c_1\ c_2\ \cdots\ c_k)$ 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 of the host after two minutes. Thus, Alice wants to find a certificate chain that not only authorizes her to login to $H$, but has the maximum expiration value among all such certificate chains. This is captured by the semiring $(\mathbb{N} \cup \{-\infty\}, \oplus, \otimes, -\infty, +\infty)$, where $\oplus$ is max and $\otimes$ is min.[12] We label a cert $C$ with the interval representing its validity period. Let $i$ be the label associated with the transition $(K_{\text{owner}[H]}, \square, s)$ in the automaton produced by algorithm $A_{pre}$. Then the authorization for Alice to login to host $H$ will be valid for $i$ time units. (The reference monitor for $H$ can use this information to log Alice off after $i$ time units.)

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

[12]$(\mathbb{N} \cup \{-\infty\}, \oplus, \otimes, -\infty, +\infty)$ has infinite descending chains; however, the only operations performed are min and max. Hence, only a finite number of values ever arise in any run of the saturation process, and the semiring-labeling framework still applies.

**Authorization specifications via semirings.** The semiring-labeling framework provides a different approach to handling the authorization specifications of auth certs. Earlier, following Clarke et al. [12], we imposed a step in which every auth cert $C = (K, S, D, T, V)$ is removed for which $T$ does not include the authorization specification $T'$ of the request (i.e., the certificates retained are those for which $T \supseteq T'$). (This step was in addition to one that removes every name and auth cert with an expired validity specification.)

However, this approach has a significant drawback, which is that it does not handle situations in which a proof of authorization requires multiple certificate chains (either a set of certificate chains or a dag of certificates, each path of which is a certificate chain that proves some *part* of the required authorization).[13] For instance, consider the following certificate set:

$c_1$ : $(K\ K_A\ 0$ `((dir /afs/cs.wisc.edu/public/tmp) read)` $[t_1 \ldots t_2])$

$c_2$ : $(K\ K_A\ 0$ `((dir /afs/cs.wisc.edu/public/tmp) write)` $[t_1 \ldots t_2])$

Suppose that Alice makes the request

$(K_A,$ `((dir /afs/cs.wisc.edu/public/tmp) (* set read write)))`.

In this case, the chain "$(c_1)$" authorizes Alice to read from directory `/afs/cs.wisc.edu/public/tmp`, and a separate chain "$(c_2)$" authorizes her to write to `/afs/cs.wisc.edu/public/tmp`. Together, $(c_1)$ and $(c_2)$ prove that she has both read and write privileges for `/afs/cs.wisc.edu/public/tmp`. 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, 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.

[13]This is the basis for the observation by Li and Mitchell that the "5-tuple reduction rule" of [16] is incomplete [22]. Although [12] speaks of "certificate chains", and [16] refers to "... finding the correct list of reductions ...", which appears to imply a single path, Carl Ellison confirmed to us that a scenario that requires justification via a set of certificate chains (or a dag of certificates) is reasonable, and noted that the justification of $k$-of-$m$ threshold subjects also requires a set or a dag [15].

Table 3

A semiring for authorization

| | $D$ | $\oplus$ | $\otimes$ | 0 | 1 |
|---|---|---|---|---|---|
| Authorization | $\mathcal{P}$ (Resources) | $\cup$ | $\cap$ | $\emptyset$ | Resources |

The semiring-labeling framework [26] overcomes these problems, provided that SPKI/SDSI authorization specifications form a bounded idempotent semiring.[14] A semiring of authorization-specification values is shown in Table 3.

Certificate-set analysis is performed using the techniques for generalized pushdown reachability on weighted PDSs developed in [26]. After all name and auth certs that have an expired validity specification are removed, the generalized pushdown reachability algorithm is run, using all of the remaining certificates, starting with a configuration automaton that accepts the language $\{\langle K_A, \Box\rangle, \langle K_A, \blacksquare\rangle\}$. In the weighted configuration automaton that would be constructed, there would be a transition $(K_{\text{owner[}\texttt{(dir /afs/cs.wisc.edu/public/tmp)}\text{]}}, \Box, s)$ with weight `(* set read write)`. This indicates that Alice's read/write request is authorized, and the algorithm given in [26] for identifying a set of certificate-chains that justify the authorization would return the set $\{(c_1), (c_2)\}$.

It should be noted that any number of such semirings can be used together. Formally, we use a cross-product semiring (i.e., elements are tuples, one component from each semiring; $\oplus$ and $\otimes$ are computed pointwise; etc.)

## *4.7. Distributed authorization*

This section describes how the automaton-based approach has certain advantages for distributed authorization problems. This is the subject of on-going work, but is sketched here to emphasize another potential advantage of the approach to certificate-set analysis that has been taken in this paper.

The Computer Sciences Department (CS) at the UW-Madison is in the College of Letters and Sciences (L&S). There are many other departments in L&S, such as Biology. Suppose that there is a resource $R$ that should only be accessible to the faculty in a department that belongs to L&S. The owner of $R$ might issue the cert

$$K_{\text{owner}[R]}\,\Box \rightarrow K_{LS}\ \texttt{faculty}\,\Box,$$

and a system administrator in L&S might issue the following set of certs $\mathcal{C}_{LS}$:

[14] As noted earlier, it is acceptable to have infinite descending chains as long only a finite number of values ever arise in any run of the saturation process [26]. Regardless of whether we think of the set of resources as being of finite or infinite cardinality, the authorization specifications of a given set of certificates can only name a finite number of sets, durations of valid certificates, and roots of tree-structured file hierarchies, and so only a finite number of values ever arise during saturation.

$K_{LS}$ `faculty` $\rightarrow K_{CS}$ `faculty`

$K_{LS}$ `faculty` $\rightarrow K_{Bio}$ `faculty`

$\cdots$ (certs for other departments in $L\&S$)

A system administrator for $CS$ might issue the following set of certs $\mathcal{C}_{CS}$:

$K_{CS}$ `faculty` $\rightarrow K_B$

$\cdots$ (certs for other faculty members in $CS$)

$K_{CS}$ `students` $\rightarrow K_A$

$\cdots$ (certs for other students in $CS$)

We want to determine whether principal $K_B$ is authorized to access resource $R$. In the Clarke et al. setting, we would first compute the name-reduction closure of the set of certs $\mathcal{C}_{LS} \cup \mathcal{C}_{CS}$ and then proceed as before. In a realistic setting, the sizes of the sets $\mathcal{C}_{CS}$ and $\mathcal{C}_{LS}$ could be quite large, and thus computing the closure of the union could require significant time and space.

The algorithms that have been presented in this paper are automaton-based. In the distributed setting, the automaton-based approach may enjoy certain advantages over an approach like that of Clarke et al. In particular, when work can be partitioned, the authorization question for $K_B$ can be determined in a distributed manner: automata can be computed at separate sites, and the information shipped between sites can take the form of automata. For instance, we could compute the following two sets at L&S and CS, respectively:

$$P_1 = \mathit{post}^\star[\mathcal{C}_{LS}](\{\langle K_{\text{owner}[R]}, \square\rangle\})$$

$$P_2 = \mathit{pre}^\star[\mathcal{C}_{CS}](\{\langle K_B, \square\rangle, \langle K_B, \blacksquare\rangle\})$$

If the intersection of $P_1$ and $P_2$ is non-empty (a standard operation on automata), $K_B$ is granted authorization. A similar operation can be used to answer authorization questions about other departments in L&S, such as Bio.

In this example, the independent name-reduction closures of $\mathcal{C}_{LS}$ and $\mathcal{C}_{CS}$ do not yield any new certificates; therefore, the procedure proposed by Clarke et al. would not provide a basis for savings during a distributed authorization-resolution procedure.

Certain technical conditions must hold for this approach to be correct. For example, a principal in CS must not refer to a local name in the L&S domain; i.e., the certs must be organized hierarchically. These issues are the subject of on-going work.

## 5. Related work

A certificate-chain-discovery algorithm for SPKI/SDSI was first proposed by Clarke et al. [12]. Their algorithm was based on the idea of computing the name-reduction closure of the certificate set. A credential-chain-discovery algorithm for the role-based trust management language $RT_0$ was presented by Li et al. [23]. One feature that distinguishes our work from both of those papers is the use of automaton-based model-checking techniques from the theory of model checking pushdown systems [8,17]. In the present paper, we have shown that the techniques from the PDS model-checking literature solve not only the problem of discovering certificate chains, but also provide answers to a broad array of questions that one might wish to pose about a set of SPKI/SDSI certificates. One of the most striking differences between this approach and previous work on certificate-chain-discovery is that the PDS-based certificate-chain-discovery algorithms compute the *actual closure* of the certificate set, not just the name-reduction closure (e.g., see "Authorized access 3" in Section 4.4). In general, the closure of a certificate set is an infinite set; however, it is a regular set – a fact that we are not aware of anyone observing before in the authorization literature – and hence can be represented via a finite-state automaton.

A fair amount of research exists on the formal semantics of SPKI/SDSI [1,19, 20,24]. Most of this research is geared towards giving a formal semantics to the local name spaces and tuple-reduction rules of SPKI/SDSI. The SPKI/SDSI-to-PDS connection presented in this paper provides an alternative semantics for SPKI/SDSI: The names of an SPKI/SDSI name space are identified with the configurations of the transition system defined by a PDS. Compared to existing work, the SPKI/SDSI-to-PDS connection has the following advantages:

- It provides a semantic account of a number of aspects of SPKI/SDSI.
- It leverages off the substantial body of research that exists on the subject of model-checking PDSs; in particular, one immediately obtains polynomial-time algorithms for a number of certificate-set analysis problems.
- A standard logic, LTL, provides a way to answer a general class of certificate-set-analysis questions. All such queries can be answered in time polynomial in the size of the certificate set.[15]

The model-checking problem for "context-free processes" has been addressed in [10,21]; context-free processes can be viewed as pushdown systems that have a single control location.

Benedikt et al. [4] showed that pushdown systems were equivalent to an "unrestricted" version of the Hierarchical State Machines (HSMs) introduced (in their restricted form) by Alur and Yannakakis [3]. ("Hierarchical" means that a system consists of several state machines that can call each other; "unrestricted HSMs" allow recursive calls between machines.) Benedikt et al. also gave algorithms for LTL

[15]The constant of proportionality is exponential in the size of the formula; however, as with other types of verification problems based on LTL model checking, the formulas of interest are usually small.

and CTL$^*$ model checking for unrestricted HSMs [4]. Similar algorithms for LTL model checking were developed independently and contemporaneously by Alur et al. [2].

## Acknowledgements

We are grateful to Stefan Schwoon for the discussions we had with him about the subject of the paper. The careful reviews and suggestions for improvement made by the referees are also greatly appreciated.

## References

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

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

[3] R. Alur and M. Yannakakis, Model checking of hierarchical state machines, Volume 23 of *Software Engineering Notes*, New York, ACM Press, 1998, pp. 175–188.

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

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

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

[7] R.F. Book and F. Otto, *String-Rewriting Systems*, Springer, 1993.

[8] 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.*, Springer-Verlag, 1997, pp. 135–150.

[9] J.R. Büchi, *Finite Automata, Their Algebras and Grammars*, Springer, 1988.

[10] O. Burkart and B. Steffen, Model checking for context-free processes, in: *Proc. CONCUR*, Volume 630 of *Lec. Notes in Comp. Sci.*, Springer-Verlag, 1992, pp. 123–137.

[11] D. Caucal, On the regular structure of prefix rewriting, *Theoretical Computer Science* **106** (1992), 61–86.

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

[13] E.M. Clarke, O. Grumberg and D. Peled, *Model Checking*, MIT Press, 2000.

[14] J.-E. Elien, Certificate discovery using SPKI/SDSI 2.0 certificates, Master's thesis, Massachusetts Institute of Technology, May 1998.

[15] C. Ellison, Personal communication to S. Jha, T. Reps and S. Schwoon, April 2003.

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

[17] 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.*, Springer-Verlag, 2000, pp. 232–247.

[18] R. Gerth, D. Peled, M. Vardi and P. Wolper, Simple on-the-fly automatic verification of linear temporal logic. in: *Protocol Specification Testing and Verification*, Chapman & Hall, Warsaw, Poland, 1995, pp. 3–18.

[19] J.Y. Halpern and R.V.D. Meyden, A logical reconstruction of SPKI, in: *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, 2001, pp. 59–70.

[20] J. Howell and D. Kotz, A formal semantics for SPKI, Technical Report 2000-363, Department of Computer Science, Dartmouth College, Hanover, NH, March 2000.

[21] J. Knoop, Demand-driven model checking for context-free processes, in: *Proc. Asian Comp. Sci. Conf.*, Volume 1742 of *Lec. Notes in Comp. Sci.*, P.S. Thiagarajan and R. Yap, eds, Springer-Verlag, 1999, pp. 201–213.

[22] N. Li and J.C. Mitchell, Understanding SPKI/SDSI using first-order logic, in: *Comp. Sec. Found. Workshop*, IEEE Comp. Soc., Wash., DC, 2003.

[23] N. Li, W.H. Winsborough and J.C. Mitchell, Distributed credential chain discovery in trust management, *Journal of Computer Security* 2002.

[24] N. Li, Local names in SPKI/SDSI 2.0, in: *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, 2000.

[25] S. Schwoon, Model-checking pushdown systems, PhD thesis, University of Munich, Munich, Germany, July 2002.

[26] S. Schwoon, S. Jha, T. Reps and S. Stubblebine, On generalized authorization problems, in: *Comp. Sec. Found. Workshop*, IEEE Comp. Soc., Wash., DC, 2003.

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