Analysis of SPKI/SDSI Certificates Using Model Checking*

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

Abstract

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 enforcing 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 [20]. Two prominent trust management systems are Keynote [5] and SPKI/SDSI [13].

SPKI/SDSI is a framework for expressing naming and authorization issues of the kind that arise in a distributed-computing environment. SPKI/SDSI *name certificates* define the names available in an issuer's local name space; SPKI/SDSI *authorization certificates* grant authorizations, or delegate the ability to grant authorizations. Clarke et al. [10] considered the problem of discovering a *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 and a formalism known as *pushdown systems* (PDSs) [7, 14]. 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

*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.

[1] There are many flavors of model checking. Henceforth, unless other-

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: For two given principals K_1 and K_2, what resources can be accessed by both K_1 and K_2?

Compromisation assessment 1: What resources 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 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 some resource must involve a certificate signed by K?

wise noted, the term "model checking" refers to model checking of pushdown systems [7, 14]. 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.

- 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 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. [10]. 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 4.3). In addition, annotating our data-structures with labels from a lattice enables us to answer additional questions (see Section 4.4), such as "When does a specific authorization expire?" Moreover, in Section 4.5 we show how our algorithm can be adapted to answer certain authorization questions in a distributed manner. Such a distributed algorithm is not possible in the framework provided by Clarke et al.

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 [10]. Section 3 and Appendix A provide background on model checking pushdown systems. Section 4 discusses applications of the formal machinery to certificate-set analysis problems. Section 5 discusses related work. Appendix B provides details about certificate-chain reconstruction. Appendix C presents some definitions pertaining to lattices that are used in Section 4.4.

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 [9] and [7, 14] may wish to skip Sections 2 and 3, respectively (although there are some minor notational differences with those papers; see footnotes 3 and 4).

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 re-

$K\ \mathtt{A} \longrightarrow S$	An SPKI/SDSI name cert $(K, \mathtt{A}, S, V)$
$K_T\ \square \longrightarrow S\ \square$	An SPKI/SDSI auth cert (K, S, D, T, V), with delegation bit D on
$K_T\ \square \longrightarrow S\ \blacksquare$	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
$p \overset{\gamma}{\leadsto} q$	The relation $p\ (\xrightarrow{\epsilon})^{\star} \xrightarrow{\gamma} (\xrightarrow{\epsilon})^{\star}\ q$ in a configuration automaton

Table 1. Kinds of arrows used in the paper.

lated to representation of keys can be found in [13].

An *identifier* is a word over some alphabet Σ. 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, locals 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 σ is a sequence of identifiers of length greater than one. For example, $K\ \mathtt{UW\ CS\ faculty}$ 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 *name certs*, 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 [13]. 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 4.4.)

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 *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 $K_T\ \square \longrightarrow S\ \square$ if the delegation bit D is turned on; otherwise, it will be written as $K_T\ \square \longrightarrow S\ \blacksquare$.

$R_H \; \square \longrightarrow K_0 \; \texttt{UW CS faculty} \; \square$	(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)

Figure 1. Complete set of certs $\mathcal{C}$.

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

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 K, T. For instance, suppose that Alice (i.e, K_A) wants to login to host H (i.e., use resource R_H). Initially, the reference monitor associated with H denies access to her, but reports the following ACL entry (written as an auth cert) to Alice:

$$R_H \; \square \longrightarrow K_0 \; \texttt{UW CS faculty} \; \square$$

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

2.3 Name-Reduction Closure

We now describe the algorithm given in [10, 12]. The reader is referred to [12] for additional details.

First, we define the concept of a closure of a set of certificates $\mathcal{C}$. A term S appearing in the rules can be viewed as a string, over the alphabet $\mathcal{K} \cup \mathcal{A}$, in which elements of $\mathcal{K}$ appear only in the beginning. For uniformity, we also refer to strings of the form $S \; \square$ and $S \; \blacksquare$ as terms. Assume that we are given a rewrite rule $L \longrightarrow R$ corresponding to 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,

$$\begin{aligned} &(K_A \; \texttt{Bob} \longrightarrow K_B)(K_A \; \texttt{Bob myFriends}) \\ &\quad = \; K_B \; \texttt{myFriends} \end{aligned}$$

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

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

Next, we define the composition of two rewrite rules. 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 $R_1 = L_2 X$. Then the *composition* $C_2 \circ C_1$ of the two rules C_1 and C_2 is the rule $L_1 \longrightarrow R_2 X$. For example, consider the following two rules:

$$\begin{aligned} &K_A \; \texttt{friends} \longrightarrow K_A \; \texttt{Bob myFriends} \\ &K_A \; \texttt{Bob} \longrightarrow K_B \end{aligned}$$

The composition of the previous two rules 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 $C_1 \circ C_2$ 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.[3] In general, $\mathcal{C}^\star$ is infinite and hence cannot be computed directly. For example, consider the set of certificates $\mathcal{C} = \{(K \; \texttt{A} \longrightarrow K \; \texttt{A A})\}$. The closure $\mathcal{C}^\star$ of $\mathcal{C}$ is the following set:

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

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 $K_A \in V_{\mathcal{C}}(K_0 \; \texttt{UW CS faculty})$, and thus Alice will be authorized 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

[3] For rule application, we write $(L \rightarrow R)(S)$ instead of $S \circ (L \rightarrow R)$ used in [10]. For the composition of $C_1 = (L_1 \rightarrow L_2 X)$ and $C_2 = L_2 \rightarrow R_2$, we write $C_2 \circ C_1$ instead of $C_1 \circ C_2$ (so that $(C_2 \circ C)(L_1) = (C_2(C_1(L_1))) = R_2 X$). Finally, we use $\mathcal{C}^\star$ instead of $\mathcal{C}^+$.

K_2 `faculty` $\longrightarrow K_B$	$(8) = (5) \circ (4)$
$K_B \ \Box \longrightarrow K_A \ \blacksquare$	$(9) = (7) \circ (6)$
$R_H \ \Box \longrightarrow K_1$ `CS faculty` $\Box$	$(10) = (2) \circ (1)$
$R_H \ \Box \longrightarrow K_2$ `faculty` $\Box$	$(11) = (3) \circ (10)$
$R_H \ \Box \longrightarrow K_B \ \Box$	$(12) = (8) \circ (11)$

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

in [10, 12]. A *reducing cert* is of the form $K \ \mathtt{A} \longrightarrow K'$. 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 [10]:

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

In other words, it is safe to inspect the name-reduction closure in order to find out the set of keys that correspond to a name N.

We now return to our authorization example and describe the four-step procedure from [10] for determining whether a principal K_P is authorized to access a given resource R, given a set of certificates $\mathcal{C}$.

1. **Remove useless certificates**
   All name and auth certificates that are are removed from the set $\mathcal{C}$. All auth certs whose authorization tag does not refer to resource R 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 Figure 1 yields the additional certs shown in Figure 2.

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

   $$\begin{array}{l} K_B \ \Box \longrightarrow K_A \ \blacksquare \ , \\ R_H \ \Box \longrightarrow K_B \ \Box \ . \end{array}$$

   Second, remove all rules of the form $K_i \ \Box \longrightarrow K_j \ \blacksquare$, 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 $K_i \ \Box \longrightarrow K_j \ \blacksquare$ or $K_i \ \Box \longrightarrow K_j \ \Box$. In our example, the edges are $R_H \rightarrow K_B$ and $K_B \rightarrow K_A$. Fourth, perform depth-first search to determine whether there is a path from R_H to K_P. In our example, there is a path from R_H to Alice's key K_A, so Alice is granted permission 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

   $$\begin{array}{l} ((7) \circ (6) \circ (5) \circ (4) \circ (3) \circ (2) \circ (1)) \, (R_H \ \Box) \\ \ = \ K_A \ \blacksquare \ . \end{array}$$

   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 [12, Chapter 3].

Next, we discuss the time and space complexity of name-reduction closure. 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

$$L \rightarrow K A_1 A_2 \cdots A_m \ .$$

After one name reduction, we obtain a rule of the following form:

$$L \rightarrow K_1 A_2 \cdots A_m \ .$$

After $i < m$ name reductions, we obtain rules of the following form:

$$L \rightarrow K_i A_{i+1} \cdots A_m$$

After m reductions, we obtain a rule of the form $L \rightarrow K_m$. There are n_K possibilities for the keys $K_1, \cdots, K_m$, so there are $n_K m$ 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 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 [12].

3 Background on Model Checking Pushdown Systems

This section provides the necessary background on model checking of *pushdown systems* (*PDSs*). The material in this paper is largely based on a paper by Esparza et al. [14]. A detailed treatment of model checking PDSs, including the computational complexity of various problems, can be found in [7].

A pushdown system is a triple $\mathcal{P} = (P, \Gamma, \Delta)$, where P is a finite set of *control locations*, Γ 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 $\langle q, \gamma \rangle \hookrightarrow \langle q', w \rangle$. Pushdown systems are similar to pushdown automata; however, unlike pushdown automata they do not have an input alphabet. Therefore, PDSs should not be viewed as language recognizers, but as mechanisms that specify possibly infinite-state transition systems.

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.[4] A *run* of $\mathcal{P}$ is a sequence of configurations $c_0, c_1, \cdots, c_n$ such that c_i is an immediate predecessor of c_{i+1}.

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.

[4] In [14], 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.

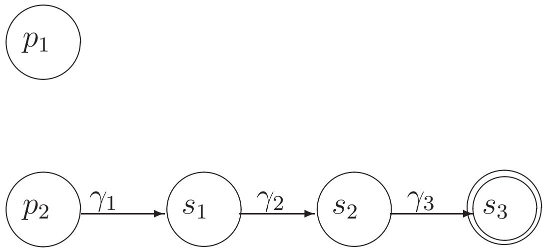

Figure 3. Automaton that accepts $C = \{\langle p_2, \gamma_1\gamma_2\gamma_3 \rangle\}$.

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 $Conf(\mathcal{A})$.

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

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

The automaton shown in Figure 3 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 regular [7]. The 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, γ, q)

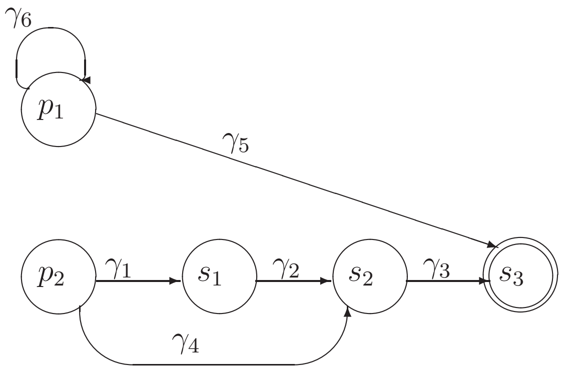

Figure 4. Automaton that accepts $pre^\star(C) = \{\langle p_1, \gamma_6^i\gamma_5\rangle\} \cup \{\langle p_2, \gamma_1\gamma_2\gamma_3\rangle, \langle p_2, \gamma_4\gamma_3\rangle\}$**.**

Theorem 1 *[14]* 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_Δ is the sum of the lengths of the right-hand sides of transition rules in Δ. 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 Figure 3 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 Figure 4. The transition rule $(p_1, \gamma_6) \hookrightarrow (p_1, \epsilon)$ from $\mathcal{P}$ causes a self-loop (p_1, γ_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 $p_2 \overset{\gamma_1\gamma_2}{\rightarrow} s_2$ holds in A causes the transition (p_2, γ_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 $p_2 \overset{\gamma_2\gamma_3}{\rightarrow} s_3$ holds in A causes the transition (p_1, γ_5, s_3) to be added to A. The automaton shown in Figure 4 accepts the following set of configurations:

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

3.2 Computing $post^\star$

Consider a PDS $\mathcal{P} = (P, \Gamma, \Delta)$ and a regular set of configurations C that is represented as a automaton $\mathcal{A}$. We will assume that each transition rule $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$ of Δ satisfies $|w| \leq 2$. 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 $\langle p, \gamma\rangle \hookrightarrow \langle p', \gamma_1, \cdots, \gamma_k\rangle$, where $k \geq 3$. We add $k-2$ new control locations $p_1, \cdots, p_{k-2}$ 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 as an automaton $\mathcal{A}$. We will construct an automaton $\mathcal{A}_{post^\star}$ that accepts $post^\star(C)$. The automaton $\mathcal{A}_{post^\star}$ is obtained from $\mathcal{A}$ in the following two phases:

- **Phase I**
  For each transition rule $r \in \Delta$ of the form $\langle p, \gamma\rangle \hookrightarrow \langle p', \gamma'\gamma''\rangle$, add to $\mathcal{A}$
  (i) a new state r, and
  (ii) a transition (p', γ', r).
- **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 $(\overset{\epsilon}{\rightarrow})^\star \overset{\gamma}{\rightarrow} (\overset{\epsilon}{\rightarrow})^\star$.) 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', ϵ, 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', γ', q).
  - If $r = \langle p, \gamma\rangle \hookrightarrow \langle p', \gamma'\gamma''\rangle \in \Delta$ and $p \overset{\gamma}{\rightsquigarrow} q$ in the current automaton, add a transition (r, γ'', q).

Theorem 2 *[14]* 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 $O(n_P n_Q(n_Q + n_\Delta) + n_P n_\delta)$ time and space, where $n_P = |P|$, $n_\Delta = |\Delta|$, $n_Q = |Q|$, and $n_\delta = |\delta|$.

Note: The complexities mentioned in Theorems 1 and 2 refer to the improved versions of the algorithms, which are presented in [14].

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

$$\begin{aligned} m_0 &: \langle p_2, \gamma_4\rangle \hookrightarrow \langle p_2, \gamma_1\gamma_2\rangle \\ m_1 &: \langle p_1, \gamma_5\rangle \hookrightarrow \langle p_2, \gamma_4\gamma_3\rangle \\ m_2 &: \langle p_1, \gamma_6\rangle \hookrightarrow \langle p_1, \epsilon\rangle \end{aligned}$$

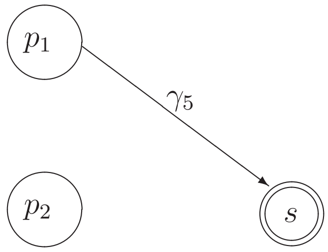

Figure 5. Automaton that accepts $C = \{\langle p_1, \gamma_5 \rangle\}$

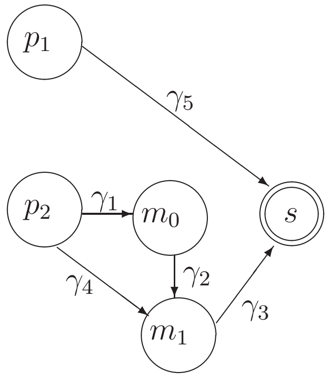

Figure 6. Automaton that accepts $post^\star(C) = \{\langle p_1, \gamma_5 \rangle, \langle p_2, \gamma_4\gamma_3 \rangle, \langle p_2, \gamma_1\gamma_2\gamma_3 \rangle\}$**.**

Consider the automaton shown in Figure 5, which accepts the set of configurations $C = \{\langle p_1, \gamma_5 \rangle\}$. The automaton corresponding to $post^\star(C)$ is shown in Figure 6. The states labeled m_0, m_1, and m_2 correspond to the transition rules with the same label. The automaton shown in Figure 6 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 Logics

Let AP be a finite set of atomic propositions, and let $\Sigma = 2^{AP}$. Let ϕ be an LTL formula over the atomic propositions AP. (The reader should consult [11, Chapter 3] and [7] 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 ϕ, determine whether c satisfies ϕ (i.e., $c \models \phi$).

A summary of the LTL model-checking procedure from [14] can be found in Appendix A.

4 From SPKI/SDSI to PDSs

This section explores 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 $\mathcal{C}$. Let the set of keys, identifiers, and resources that appear in $\mathcal{C}$ be denoted by $\mathcal{K}_\mathcal{C}$, $\mathcal{A}_\mathcal{C}$, and $\mathcal{R}_\mathcal{C}$, respectively.

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

- The set of locations is $P = \mathcal{K}_\mathcal{C} \cup \mathcal{R}_\mathcal{C}$, i.e., each key and resource 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 Δ contains a rule $\langle K, \gamma \rangle \hookrightarrow \langle K', w \rangle$ iff $(K\ \gamma \to K'\ w) \in \mathcal{C}$, i.e. the certs in $\mathcal{C}$ correspond to the transition rules.

Assume that we are given a set of certs $\mathcal{C}$. Consider a term $N \in \mathcal{K} \cup \mathcal{N}$. The term $N = K A_1 \cdots A_m$ 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 3 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', $N' \in \mathcal{C}^\star(N)$ if and only if $c(N) \Rightarrow^\star c(N')$ in $\mathcal{P}_\mathcal{C}$. In other words, we have the following equality:

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

Given a set of certificates $\mathcal{C}$, suppose that we are interested in determining whether a principal K_P is authorized to access resource R. There are two options to solve this authorization problem: 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 $\langle R, \square \rangle$ 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:

$$\langle R, \square\rangle \in pre^\star(\{\langle K_P, \square\rangle, \langle K_P, \blacksquare\rangle\}) \quad (1)$$
$$post^\star(\{\langle R, \square\rangle\}) \cap \{\langle K_P, \square\rangle, \langle K_P, \blacksquare\rangle\} \neq \emptyset \quad (2)$$

Algorithms based on conditions (1) and (2) will be referred to as A_{pre} and A_{post}, respectively. The SPKI/SDSI algorithm described in Section 2.3 will be referred to as $A_{\text{SPKI/SDSI}}$. Based on Lemma 3, the following theorem is easy to prove:

Theorem 4

1. A principal K_P is granted authorization to access resource R by algorithm $A_{\text{SPKI/SDSI}}$ iff algorithm A_{pre} grants authorization to K_P to access R.

2. A principal K_P is granted authorization to access resource R by algorithm $A_{\text{SPKI/SDSI}}$ iff algorithm A_{post} grants authorization to K_P to access R.

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 $\langle R, \square\rangle$ is accepted by the automaton for $pre^\star(Y)$.

The complexity of algorithm A_{pre} can be analyzed as follows: In case of a single resource, the number of states n_Q in the automaton A_Y is $n_K + 2$. There is a one-to-one correspondence between the transition rules and the certs in the set $\mathcal{C}$; therefore, n_Δ is equal to $|\mathcal{C}|$. Moreover, the number of transitions n_δ 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 Figure 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 Γ are given by the following sets:

$$\{R_H, K_0, K_1, K_2, K_3, K_B, K_4, K_A\}$$
$$\{\texttt{UW}, \texttt{CS}, \texttt{faculty}, \texttt{Bob}, \texttt{Alice}, \square, \blacksquare\}$$

$\langle R_H, \square\rangle \hookrightarrow \langle K_0, \texttt{UW CS faculty}\,\square\rangle$	(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)

Figure 7. The set of transition rules Δ in $\mathcal{P}_\mathcal{C}$.

The transition rules Δ are shown in Figure 7.

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

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

A configuration automaton $A_X = (\Gamma, Q, \delta, P, F)$ that accepts X can be defined as follows: $Q = P \cup \{s\}$, $\delta = \{(K_A, \square, s), (K_A, \blacksquare, s)\}$, and $F = \{s\}$. The automaton that would be constructed for $pre^\star(X)$ is shown in Figure 8. Note that the configuration $\langle R_H, \square\rangle$ is accepted by the automaton, and thus principal K_A (Alice) is authorized to login to host H.

Next, we describe in detail the algorithm A_{post} that uses the construction $post^\star$. Again, suppose that we are interested in determining whether a principal K_P is authorized to access resource R, 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:

   $$\{\langle R, \square\rangle\}$$

   Construct the automaton $A_S = (\Gamma, Q, \delta, P, F)$, where $Q = P \cup \{s\}$, $\delta = \{(R, \square, s)\}$, and $F = \{s\}$. ($Conf(A_Y)$ is S.) Before computing $post^\star(S)$, 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 $|w| \leq 2$. Then add a new state r for each transition rule of the form $\langle p, \gamma\rangle \hookrightarrow \langle p', \gamma'\gamma''\rangle$ and add a transition (p', γ', r) to the automaton A_S. Finally, complete the construction of the automaton for $post^\star(S)$ by repeatedly applying the saturation rule.

3. Grant authorization to K_P iff $\langle K_P, \square\rangle$ or $\langle K_P, \blacksquare\rangle$ is accepted by the automaton corresponding to $post^\star(S)$.

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

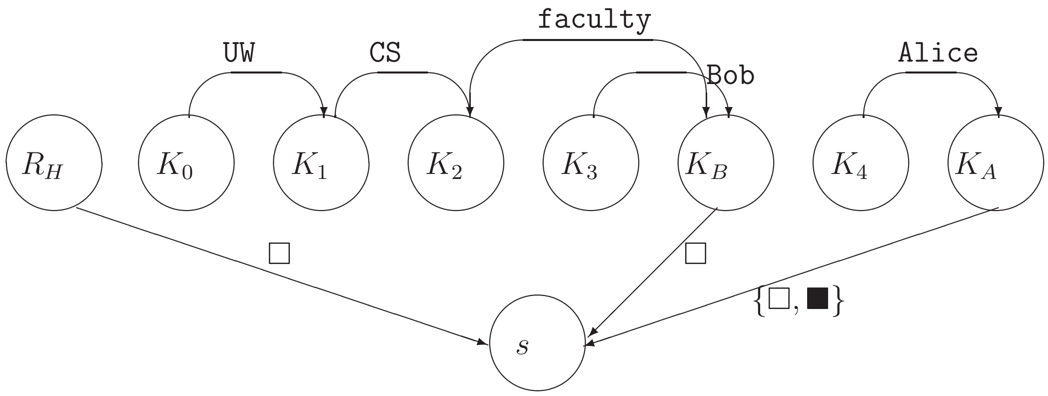

Figure 8. Automaton that accepts the set of configurations $pre^\star(\{\langle K_A, \Box\rangle, \langle K_A, \blacksquare\rangle\})$.

Example 5 Consider the set of certs $\mathcal{C}$ shown in Figure 1. The PDS $\mathcal{P}_\mathcal{C} = (P, \Gamma, \Delta)$ corresponding to the set of certs $\mathcal{C}$ was explicitly constructed in Example 4. Recall that the $post^\star$ algorithm assumes that every transition rule $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$ satisfies $|w| \leq 2$. The following rule in $\mathcal{P}_\mathcal{C}$ does not satisfy that constraint:

$$\langle R_H, \Box\rangle \hookrightarrow \langle K_0, \texttt{UW CS faculty}\ \Box\rangle$$

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.

$$\begin{array}{rrcl} m_0 : & \langle R_H, \Box\rangle & \hookrightarrow & \langle K_0^1, \texttt{faculty}\ \Box\rangle \\ m_1 : & \langle K_0^1, \texttt{faculty}\rangle & \hookrightarrow & \langle K_0^2, \texttt{CS faculty}\rangle \\ m_2 : & \langle K_0^2, \texttt{CS}\rangle & \hookrightarrow & \langle K_0, \texttt{UW CS}\rangle \end{array}$$

One of the original transition rules of the PDS has two stack symbols on the right-hand side: $m_3 : \langle K_B\ \Box\rangle \hookrightarrow \langle K_4, \texttt{Alice}\ \blacksquare\rangle$.

A configuration automaton $A_S = (\Gamma, Q, \delta, P, F)$ that accepts the set $S = \{\langle R_H, \Box\rangle\}$ can be defined as follows: $Q = P \cup \{s\}$, $F = \{s\}$, and $\delta = \{(R_H, \Box, s)\}$. After phase I of the construction from Section 3.2, the automaton has the following components $Q = P \cup \{K_0^1, K_0^2, m_0, m_1, m_2, m_3, s\}$, $F = \{s\}$, and δ is the set shown in Figure 9. The automaton that would be constructed for $post^\star(S)$ is as shown in Figure 10. Note that the configuration $\langle K_A, \blacksquare\rangle$ is accepted by this automaton, and thus principal K_A (Alice) is authorized to login to host H.

4.1 Certificate-Chain Reconstruction and Threshold Subjects

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.)

$$\begin{array}{l} (R_H, \Box, s) \\ (K_0^1, \texttt{faculty}, m_0) \\ (K_0^2, \texttt{CS}, m_1) \\ (K_0, \texttt{UW}, m_2) \\ (K_4, \texttt{Alice}, m_3) \end{array}$$

Figure 9. The set of transitions δ.

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

With each transition (p, γ, q) in the automaton, we associate a set of identifiers for transition rules. Initially, this set is empty for all of the transitions in the automaton. Intuitively, this set represents the transition rules that caused this transition to be added. If the transition rule $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$, together with the fact that $p' \xrightarrow{w} q$ holds in the automaton, causes the transition (p, γ, q) to be added to the automaton, the identifier of the transition rule $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$ is added to the set of identifiers associated with the transition (p, γ, q). After the completion of the $pre^\star$ construction, the certificate chain can be reconstructed by starting from an identifier associated with the transition $(R, \Box, s)$ and tracing back. The tracing-back procedure is conceptually simple, and details are provided in Appendix B.

Threshold subjects can be handled in our framework by introducing extra keys (see [13, Section 10]).

4.2 Certificate-Set Analysis Problems

This section discusses applications of model checking to specific certificate-set analysis problems; in particular,

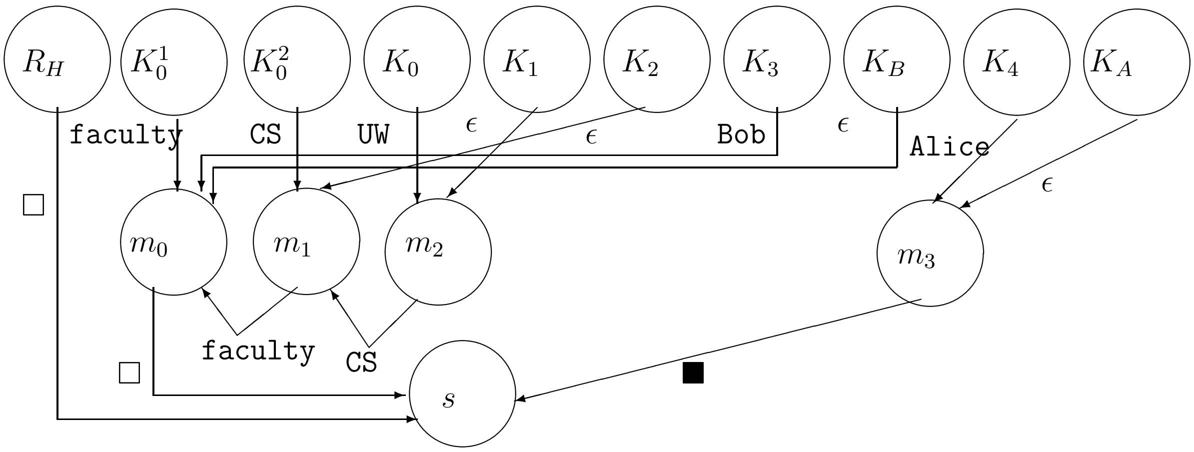

Figure 10. Automaton that accepts the set of configurations $post^\star(S)$.

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 $post^\star[\mathcal{C}]$.

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

$\langle R, \square\rangle \in pre^*(\{\langle K, \square\rangle, \langle K, \blacksquare\rangle\})$ or, alternatively, $\{\langle K, \square\rangle, \langle K, \blacksquare\rangle\} \cap post^*(\{\langle R, \square\rangle\}) = \emptyset$

Authorized access 2: Given resource R and name N, is N authorized to access R?

$\langle R, \square\rangle \in pre^*(\{c(N)\})$ or, alternatively, $\{c(N)\} \cap post^*(\{\langle R, \square\rangle\}) = \emptyset$

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

$$post^*(\{\langle R, \square\rangle\})$$

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

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

Shared access 2: For two given principals K_1 and K_2, what resources can be accessed by both K_1 and K_2?

$$\begin{aligned} & \{R \mid \langle R, \square\rangle \in pre^*(\{\langle K_1, \square\rangle, \langle K_1, \blacksquare\rangle\}) \\ \cap\ & pre^*(\{\langle K_2, \square\rangle, \langle K_2, \blacksquare\rangle\})\} \end{aligned}$$

Compromisation assessment 1: What resources could principal K have gained access to (solely) due to the presence of maliciously or accidentally issued certificate set $\mathcal{C}' \subseteq \mathcal{C}$?

$$\begin{aligned} & \{R \mid \langle R, \square\rangle \in (pre^\star[\mathcal{C}](\{\langle K, \square\rangle, \langle K, \blacksquare\rangle\}) \\ -\ & pre^\star[\mathcal{C} - \mathcal{C}'](\{\langle K, \square\rangle, \langle K, \blacksquare\rangle\}))\} \end{aligned}$$

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

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

Expiration vulnerability 1: What resources 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 $\langle R, \square\rangle$ satisfies the LTL formula

$$\square(\neg S \,\mathcal{U}\, (Q \vee \square\neg S)) \qquad (3)$$

Universally guarded access 2: Is it the case that all authorizations that grant a given principal K' access to some resource must involve a certificate signed by K?

We again use LTL formula (3). 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 $\langle K', \square\rangle$ and $\langle K', \blacksquare\rangle$ are labeled with atomic proposition S.

We then ask whether every surface configuration $\langle R, \square\rangle$, such that $R \in \mathcal{R}_{\mathcal{C}}$, satisfies LTL formula (3).

4.3 Efficiency of the Automaton Representation

Clarke et al [10] 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 $\mathcal{C}$:

$$\begin{array}{ll} K\ \mathtt{C} \rightarrow K_0\ \mathtt{A}^l\ \mathtt{B}_j & (\text{for } 0 \leq j < n) \\ K_0\ \mathtt{A} \rightarrow K_i & (\text{for } 0 \leq i < n) \\ K_i\ \mathtt{A} \rightarrow K_{(i+1) \bmod n}\ \mathtt{A} & (\text{for } 0 \leq i < n) \end{array}$$

In the rule 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:

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

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:

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

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 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, $\langle K_i, w\rangle$ 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 ϵ-edges and $\mathtt{A}$-edges pointing to those shared states.

4.4 Lattice Labellings

This section describes how annotating the PDS and configuration automaton with labels from a lattice can answer several useful questions, such as "How long does a specific authorization last?" and "What is the trust level associated with an authorization?". Definitions related to lattices can be found in Appendix C. Let each cert C in the set $\mathcal{C}$ be annotated with a label $l(C)$ from a lattice $\mathcal{L}$. Let $\mathcal{P}_{\mathcal{C}}$ be the PDS corresponding to $\mathcal{C}$. A transition rule r of the 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, γ, q) of the automaton will also be labeled with an element from the lattice $\mathcal{L}$ (denoted by $l(p, \gamma, q)$). Initially, all transitions in the automaton A_X are labeled with $\top$. We add transitions (p, γ, q) to the automaton A_X using the saturation rule given below:

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

The label $l(p, \gamma, q)$ of the transition (p, γ, q) is computed as follows:

$$\begin{array}{ll} l(r) \sqcap l(p' \xrightarrow{w} q) & \text{if } (p, \gamma, q) \text{ is a new transition} \\ (l(r) \sqcap l(p' \xrightarrow{w} q)) \ \sqcup\ l(p, \gamma, q) & \text{otherwise} \end{array}$$

(The labelings on individual transitions are extended to labelings on paths by taking the meet of the labels on the transitions along the path; the labeling for a composite transitions, such as $p' \xrightarrow{w} q$, is obtained from the join of the labels on all w paths from p' to q.)

The interval lattice. Consider the lattice of intervals of the form $[0, i_r]$, where i_r is a non-negative integers or ∞. The current time is 0. We will represent the interval $[0, i_r]$ as $[i_r]$. The top and bottom elements are the intervals $[0]$ and $[\infty]$, respectively. The meet and join of the two intervals $[i]$ and $[i']$ are $[\min\{i, i'\}]$ and $[\max\{i, i'\}]$, respectively. We label a cert C with the interval representing its validity period. Let $[i]$ be the label associated with the transition $(R, \square, s)$

in the automaton produced by algorithm A_{pre}. Then the authorization for R will be valid until time i. (In the case of Alice being authorized to login to host H, the reference monitor for H can use this information to log-off Alice after i time units.)

The lattice of trust levels. Consider the lattice $\{L, M, H\}$ ordered as $L \sqsubseteq M \sqsubseteq H$, where L and H are the bottom and top elements of the lattice. Intuitively, the lattice represents trust levels, where low, medium, and high trust levels are denoted by L, M, and H, respectively. The meet and join of elements e_1 and e_2, where $e_1 \sqsubseteq e_2$, are e_1 and e_2, respectively. Assume that each cert C is labeled with an element from the lattice. For example, if $K\ \mathtt{A} \rightarrow K'\ \sigma$ is assigned a label H, the principal K assigns a high level of trust to that cert. In this case, the label of the transition $(R_H, \square, s)$ in the automaton constructed by the algorithm A_{pre} represents the "trust level" of Alice's authorization. For example, if the label of $(R_H, \square, s)$ is H, then there exists a certificate chain, all of whose labels have the label H, that justifies granting authorization to Alice. The trust level can then be used by the reference monitor for H in making authorization decisions.

4.5 Distributed Authorization

The Computer Sciences Department (CS) at the UW-Madison is in the College of Letters and Sciences (L&S). There are many other departments, such as Biology, in L&S. Assume that there is a resource R that should only be accessible to the faculty in a department that belongs to L&S. A system administrator in L&S might issue the following set of certs $\mathcal{C}_{LS}$.

$$\begin{array}{rl} R\ \square \rightarrow & K_{LS}\ \mathtt{faculty}\ \square \\ K_{LS}\ \mathtt{faculty} \rightarrow & K_{CS}\ \mathtt{faculty} \\ K_{LS}\ \mathtt{faculty} \rightarrow & K_{Bio}\ \mathtt{faculty} \\ & \cdots \text{ (certs for other departments in } L\&S) \end{array}$$

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

$$\begin{array}{l} K_{CS}\ \mathtt{faculty} \rightarrow K_B \\ \qquad \cdots \text{ (certs for other faculty members in } CS) \\ K_{CS}\ \mathtt{students} \rightarrow K_A \\ \qquad \cdots \text{ (certs for other students in } CS) \end{array}$$

We want to determine whether principal K_B is authorized to access the 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. Using the algorithms that we presented before, the authorization question for K_B can be determined in a distributed manner, i.e., the work can be partitioned. We compute the following two sets:

$$\begin{array}{rcl} P_1 & = & post^\star[\mathcal{C}_{LS}](\{\langle R, \square \rangle\}) \\ P_2 & = & pre^\star[\mathcal{C}_{CS}](\{\langle K_B, \square \rangle, \langle K_B, \blacksquare \rangle\}) \end{array}$$

If the intersection of P_1 and P_2 is non-empty (a standard operation on automata), K_B is granted authorization.[5] A similar operation can be used to answer authorization questions about other departments, such as Bio, in $L\&S$. Notice that the name-reduction closure of $\mathcal{C}_{LS}$ and $\mathcal{C}_{CS}$ does not yield any new certificates; therefore, the procedure proposed by Clarke et al. does not provide the basis for a distributed authorization-resolution procedure.

5 Related Work

Clarke et al. considered the problem of discovering certificate chains for SPKI/SDSI [10]. Their algorithm was based on the idea of computing the name-reduction closure of the certificate set. The problem of model checking pushdown systems was addressed in [7, 14]. In the present paper, we have shown the techniques from the latter papers 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. The PDS-based authorization algorithms compute the *actual closure* of the certificate set, not just the name-reduction closure.

A fair amount of research exists on the formal semantics of SPKI/SDSI [1, 16, 17, 19]. 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 is not necessary to invent a new logic.
- 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 par-

[5] In general, 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.

ticular, one immediately obtains polynomial-time algorithms for a number of certificate-set analysis problems.

In [4], Benedikt et al. 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.) When [4] was submitted to ICALP, one of the reviews contained the following remark:

> [Among the submission's contributions,] one is conceptual: it identifies that the "Unrestricted" Hierarchical State Machines ... are the same as pushdown systems. This is valuable because so far, pushdown systems had "no right of existence" (other than the fact that they were a class of infinite-state systems for which model checking is decidable; i.e., nobody knew about a potential practical value).

The present paper demonstrates that pushdown systems do have a clear practical value: a set of SPKI/SDSI certificates *is* a pushdown system. Note that the construction given in Section 4 is merely a transliteration of SPKI/SDSI terminology into PDS terminology (i.e., "keys" and "resources" are "locations"; "identifiers" are "stack symbols"; "certificates" are "transition rules"). Thus, while [4] gave correspondence theorems that demonstrated that PDSs have an *indirect* "right to exist", the present paper shows that PDSs have a *direct* "right to exist".

The model-checking problem for "context-free processes" has been addressed in [8, 18]; context-free processes can be viewed as pushdown systems that have a single control location.

Benedikt et al. addressed the problem of LTL and CTL* model checking for unrestricted HSMs [4]. Similar algorithms for LTL model checking were developed independently and contemporaneously by Alur et al. [2].

References

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

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

[3] R. Alur and M. Yannakakis. Model checking of hierarchical state machines. volume 23, 6 of *Softw. Eng. Notes*, pages 175–188, New York, Nov. 3–5 1998. ACM Press.

[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, Sept. 1999.

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

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

[8] O. Burkart and B. Steffen. Model checking for context-free processes. In *Proc. CONCUR*, volume 630 of *Lec. Notes in Comp. Sci.*, pages 123–137, 1992.

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

[10] D. Clarke, J.-E. Elien, C. M. Ellison, M. Fredette, A. Morcos, and R. L. Rivest. Certficate chain discovery in SPKI/SDSI. Available at `http//theory.lcs.mit.edu/~rivest/`, Nov. 1999.

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

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

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

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

[15] 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*, pages 3–18, Warsaw, Poland, 1995. Chapman & Hall.

[16] J. Halpern and R. Meyden. A logical reconstruction of SPKI. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pages 59–70. IEEE Computer Society Press, 2001.

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

[18] J. Knoop. Demand-driven model checking for context-free processes. In P. Thiagarajan and R. Yap, editors, *Proc. Asian Comp. Sci. Conf.*, volume 1742 of *Lec. Notes in Comp. Sci.*, pages 201–213, Dec. 1999.

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

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

A Details on Model Checking PDSs

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 path 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 5 *[14]* 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 automata* $\mathcal{B} = (\Sigma, Q, \delta, q_0, F)$ is a 5-tuple, where Σ 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 Σ is denoted by Σ^ω. Let $\sigma = \alpha_1\alpha_2\cdots$ be an infinite word over the alphabet Σ. We say that σ is accepted by $\mathcal{B}$ if there exists a sequence of states $s_0 s_1 s_2 \cdots$, 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 $s_0 s_1 s_2 \cdots$ is called an *accepting run*. Let $\mathcal{L}(\mathcal{B})$ be the language accepted by the Büchi automata $\mathcal{B}$.

Assume that we are given a configuration c and an LTL formula ϕ. 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 ω-regular language. Moreover, there are efficient algorithms to translate an LTL formula into a Büchi automaton [15]. 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 $\mathcal{BP} = ((P \times Q), \Gamma, \Delta', G)$, 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 $q \in F$.

The LTL model-checking problem reduces to the *accepting-run problem*:

> Configuration $c = \langle p, \gamma\rangle$ satisfies ϕ iff there does not exist an accepting run in $\mathcal{BP}$ starting from the configuration $\langle (p, q_0), \gamma\rangle$—i.e.,a run that visits infinitely often the configurations with control locations in G.

Let R be the G-repeating heads in $\mathcal{BP}$. It is proved in [14] that, given a configuration $c = \langle p, \gamma\rangle$, there is an accepting run starting from $\langle (p, q_0), \gamma\rangle$ if the following set is non-empty:

$$post^\star(\{\langle (p, q_0), \gamma\rangle\}) \cap pre^\star(R\Gamma^\star)$$

The set of configurations that are reachable from $\langle (p, q_0), \gamma\rangle$ is given by $post^\star(\{\langle (p, q_0), \gamma\rangle\})$. The set $pre^\star(R\Gamma^\star)$ denotes 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 in $\mathcal{BP}$.

B Details on Certificate-Chain Reconstruction

With each transition (p, γ, 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, γ, q) is added due to the PDS transition rule $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$ and the fact that the composite transition $p' \xrightarrow{w} q$ holds in the automaton. Let the composite transition $p' \xrightarrow{w} q$ be comprised of the sequence of transitions $t_1 t_2 \cdots t_n$. The structure associated with the transition (p, γ, q) is

$$(id(\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle), [t_1 t_2 \cdots t_n]),$$

where the identifier of the PDS transition rule r is denoted by $id(r)$. We explain the construction using Example 2.2. The structures associated with the transitions are shown in Figure 11. An empty structure and an empty list are represented by ϵ and $NULL$, respectively. By tracing back from the structure associated with the transition $t_8 = (R_H, \Box, s)$, we obtain the following certificate chain, which proves the authorization for `Alice`:

$$(1), (2), (3), (4), (5), (6), (7)$$

idempotent:	$a \sqcap a = a$	$a \sqcup a = a$
commutative:	$a \sqcap b = b \sqcap a$	$a \sqcup b = b \sqcup a$
associative:	$(a \sqcap b) \sqcap c = a \sqcap (b \sqcap c)$	$(a \sqcup b) \sqcup c = a \sqcup (b \sqcup c)$
absorption:	$a \sqcap (a \sqcup b) = a$	$a \sqcup (a \sqcap b) = a$

Figure 12. Properties of the operators $\sqcap$ and $\sqcup$

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

Figure 11. Structures associated the transitions of the automaton shown in Figure 8.

C Background on Lattices

A lattice $\mathcal{L}$ is a set L with binary *meet* ($\sqcap$) and *join* operators ($\sqcup$), both of which are idempotent, commutative, and associative (see Figure 12). The elements of L form a partially ordered set, ordered by $\sqsubseteq$, which may be defined in either of two ways:

$$a \sqsubseteq b \quad \text{iff} \quad a \sqcap b = a$$

$$\text{or} \quad a \sqsubseteq b \quad \text{iff} \quad a \sqcup b = b.$$

We write $a \sqsubset b$ when $a \sqsubseteq b$ and $a \neq b$.

We will assume that our lattices have a bottom element ($\perp$) and a top element ($\top$) such that, for all $a \in L$,

$$\perp \sqcap a = \perp \qquad \perp \sqcup a = a$$

$$\top \sqcap a = a \qquad \top \sqcup a = \top$$

A *chain* is a sequence of elements $a_1, \ldots, a_n$ such that for $1 \leq i \leq n-1$, $a_i \sqsubset a_{i+1}$. The *height* of $\mathcal{L}$ is the length of the longest chain in $\mathcal{L}$.