Computing Minimum-Height Certificate Trees in SPKI/SDSI

Dejvuth Suwimonteerabuth

The Sirindhorn International Thai-German Graduate School of Engineering (TGGS)
dejvuth.s.sse@tggs-bangkok.org

Abstract: SPKI/SDSI is a framework that combines a simple public-key infrastructure and a simple distributed security infrastructure with a means of defining local name spaces. It allows principals, which can be a person or an organization, to locally create groups of principals and delegate rights to other principals or groups of principals by issuing certificates. To prove authorizations, principals need to search for necessary certificates that are, in general, in the form of certificate trees. This paper defines a framework based on SPKI/SDSI which allows principals to give weights to certificates. Weights can be used to address many authorization issues such as access control of limited resources. The paper shows a connection between SPKI/SDSI and the theory of pushdown systems, and presents an algorithm that solves the authorization problem by computing minimum-height certificate trees.

1 Introduction

In access control of shared resources, authorization systems allow to specify a security policy that assigns permissions to principals in the system. The *authorization problem* is, given a security policy, should a principal be allowed access to a given resource? SPKI/SDSI [EFL$^+$99] is a framework which allows a principal to locally create groups of principals by issuing so-called *name certificates*, and grant authorizations or delegate the right to grant authorizations to other principals or groups of principals (even without knowing individuals in the groups) by issuing so-called *authorization certificates*. In [CEE$^+$01], it has been shown that the authorization problem reduces to discovering a *certificate chain* to prove whether a given principal is allowed to access a given resource. The certificate chain might consists of one or more name definitions or authorization grants and delegations.

In general, however, a principal might need to find more than one certificate chain to prove his/her authorization. SPKI/SDSI allows, for instance, Alice to issue a certificate to give an authorization to her relatives who work in her company. Therefore, if Bob wants to prove his authorization, he must find a set of certificates which proves (i) that he is her relative *and* (ii) that he works in her company. This set of certificates forms a *certificate tree* in which each branch represents a certificate chain.

In this paper, we consider a more general system where certificates can have different *weights*. The meaning of weights depends on applications of interest. When proving an authorization, a principal does not look for an arbitrary certificate tree, but the tree that has the minimum height, i.e., the tree that involves certificates having the smallest weights. This system can be applied to many applications. For instance, different weights can be interpreted as different degrees of importance. When principals compete for a limited resource, one can control who has the right to access it based on the importance of his/her certificate tree.

Previous works have shown that SPKI/SDSI has a strong connection to the theory of pushdown systems [JR04, SJRS03]. A set of certificates can be seen as a pushdown system, and certificate-chain discovery reduces to pushdown reachability. This paper proceeds in a similar way. We propose an extension to SPKI/SDSI in which one can assign weights to certificates. Then, we present an efficient algorithm for finding minimum-height certificate trees.

We proceed as follows: Section 2 introduces SPKI/SDSI and formally defines the authorization problem. Section 3 presents alternating pushdown systems and other theoretical concepts used in the paper. Section 4 shows the connection between SPKI/SDSI and alternating pushdown systems, and presents an algorithm for solving the authorization problem. Section 5 concludes the paper.

Throughout the paper, we denote by $\mathbb{N}$ the set of non-negative integers and $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$. If n is a positive integer, then $[n] = \{i \in \mathbb{N} \mid 1 \leq i \leq n\}$.

2 SPKI/SDSI

A central notion of SPKI/SDSI are *principals*. A principal can be a person or an organization represented by a public key. Each principal defines his/her own namespace, which assigns *rôles* to (other) principals. For instance, principal *University* can define the rôle `staff` and associate principal `Alice` with its rôle. SPKI/SDSI makes such associations by issuing so-called *name certificates* (*name certs*, for short). Remarkably, principals may reference the namespace of other principals in their certificates. For instance, *University* may state that all *Engineering*'s staffs are also its staffs. In this way, SPKI/SDSI allows to associate a rôle with a group of principals described in a symbolic and distributed manner. SPKI/SDSI then allows to assign permissions to rôles using so-called *authorization certificates* (or *auth certs*).

More formally, a SPKI/SDSI system can be seen as a tuple $S = (P, A, C)$, where P is a set of *principals* (or public keys), A is a set of *rôle identifiers* (or identifiers, for short), and $C = Na \uplus Au$ is a set of certificates. Certificates can be either *name certs* (contained in Na), or *auth certs* (contained in Au).

A *term* is formed by a principal followed by zero or more identifiers, i.e., an element of the set PA^*. A name certificate is of the form $p\ \mathtt{a} \rightarrow t$, where p is a principal, a is an identifier, and t is a term. Notice that $p\ \mathtt{a}$ itself is a term. For all terms t, the sets $[\![t]\!]$ are the smallest sets of principals satisfying the following constraints:

- if $t = p$ for some principal p, then $[\![t]\!] = \{p\}$;
- if $t = t' \ \mathtt{a}$, then for all $p \in [\![t']\!]$ we have $[\![p \ \mathtt{a}]\!] \subseteq [\![t]\!]$;
- if $p \ \mathtt{a} \to t$ is name cert, then $[\![t]\!] \subseteq [\![p \ \mathtt{a}]\!]$.

For example, if *University*, *Engineering*, *Alice* are principals and `staff` is an identifier, then the certificate c_1 : *Engineering* `staff` $\to$ *Alice* expresses that Alice is an Engineering's staff, and the certificate c_2 : *University* `staff` $\to$ *Engineering* `staff` means that all Engineering's staffs are also staffs of the university.

An auth cert is of the form $p \, \square \to t \, b$, where p is a principal, t is a term, and b is either $\square$ or $\blacksquare$. It means that p grants some authorization to all principals in $[\![t]\!]$. If $b = \square$, then the principals in $[\![t]\!]$ are allowed to delegate the authorization to others; if $b = \blacksquare$, then they are not. Note that auth certs can also contain details about the authorization that they confer. We omit this detail in this paper due to the space constraint.

Formally, auth certs define a smallest relation $aut : P \times P$ between principals such that $aut(p, p')$ holds iff p grants an authorization to p':

- if there is an auth cert $p \, \square \to t \, b$, for $b \in \{\square, \blacksquare\}$, and $p' \in [\![t]\!]$, then $aut(p, p')$;
- if there is an auth cert $p \, \square \to t \, \square$, $p' \in [\![t]\!]$, and $aut(p', p'')$, then $aut(p, p'')$.

For example, the certificate c_3 : *University* $\square \to$ *University* `staff` $\blacksquare$ means that the university grants some right to all university's staffs. They, however, are not allowed to delegate that right to other principals.

The *authorization problem* in SPKI/SDSI is to determine, given a system (P, A, C) and two principals p and p', whether p' is granted authorization by p, i.e., whether $aut(p, p')$. The problem can be solved by finding a *certificate chain* that transforms $p \, \square$ into $p' \, \square$ or $p' \, \blacksquare$. In the example, if Alice wants to prove that she has the right from the university, she needs to find the chain c_3, c_2, c_1, which gives the following proof:

$$\mathit{University} \, \square \xrightarrow{c_3} \mathit{University} \ \mathtt{staff} \, \blacksquare \xrightarrow{c_2} \mathit{Engineering} \ \mathtt{staff} \, \blacksquare \xrightarrow{c_1} \mathit{Alice} \, \blacksquare$$

2.1 Intersection certificates

The SPKI/SDSI standard provides for so-called *threshold certificates*, which consists of, say, an auth cert of the form $p \, \square \to \{t_1 \, b_1, \ldots, t_n \, b_n\}$, where $b_1, \ldots, b_n \in \{\square, \blacksquare\}$, and an integer $k \leq n$. The meaning of such a cert is that p grants authorization to principal p' if there is a certificate chain to p' from at least k out of $t_1 \, b_1, \ldots, t_n \, b_n$. We restrict ourselves to the case where $k = n$ and use the more suggestive name *intersection certificate* instead. Formally, intersection certificates extend the relation aut as follows:

- if $p \, \square \to \{t_1 \, b_1, \ldots, t_n \, b_n\}$, then $aut(p, p')$, where $p' \in \bigcap_{i=1}^{n} [\![t_i]\!]$; moreover,
- if $b_j = \square$, $p' \in [\![t_j]\!]$, and $aut(p', p'')$ such that $p'' \in \bigcap_{i=1}^{n} [\![t_i]\!]$, then $aut(p, p'')$.

Notice that one could analogously define threshold name certificates in a similar way. However, in [CEE$^+$01, JR04] the use of threshold certificates is restricted to just authorization certificates, claiming that the use of threshold certificates in name certificates would make the semantics "almost surely too convoluted".

In the presence of intersection certificates, proofs of authorizations are in the form of certificate trees, where each branch corresponds to a certificate chain. Continuing the example, if the university instead delegates the right to Bob to grant the authorization to the staffs, we have c_3: $\mathit{University}\ \Box \rightarrow \{\mathit{University}\ \texttt{staff}\ \blacksquare, \mathit{Bob}\ \Box\}$ and if Bob grants the authorization to Alice c_4: $\mathit{Bob}\ \Box \rightarrow \mathit{Alice}\ \blacksquare$, the following certificate tree proves Alice's authorization:

$$\begin{tikzcd}
 & c_2 \arrow[r, dash] & c_1 \\
c_3 \arrow[ru, dash] \arrow[r, dash] & c_4 &
\end{tikzcd}$$

2.2 Min SPKI/SDSI

We extend SPKI/SDSI by assigning weights to certificates. A *min SPKI/SDSI* system is a tuple (S, f), where $S = (P, A, C)$ is a SPKI/SDSI system and $f : C \rightarrow \mathbb{N}^\infty$ is a function that assigns a natural number to each rule. We extend the function f to every node c in certificate trees to signify the height of the node: if c is a leaf, its height is $f(c)$; otherwise, its height is $f(c) + \max_{i=1}^{n} f(c_i)$, where c_i is a child node of c for all $i \in [n]$. The height of a certificate tree is the height of the root.

In the example, if the university issues a similar certificate for Carol c_5: $\mathit{University}\ \Box \rightarrow \{\mathit{University}\ \texttt{staff}\ \blacksquare, \mathit{Carol}\ \Box\}$, and gives this delegation more priority by assigning, say, $f(c_3) = 3$, $f(c_5) = 1$, and $f(c) = 0$ for any other c. Therefore, assuming that there is a chain from $\mathit{Carol}\ \Box$ to $\mathit{Alice}\ \blacksquare$, she would prefer the certificate tree where the certificate c_5 is the root, since it gives her more priority than the previous tree.

The rest of the paper deals with the problem of finding minimum-height certificate trees by using the theory of pushdown systems.

3 Pushdown systems

An *alternating pushdown system* (APDS) is a triplet $\mathcal{P} = (P, \Gamma, \Delta)$, where P is a finite set of *control locations*, Γ is a finite *stack alphabet*, and $\Delta \subseteq (P \times \Gamma) \times 2^{(P \times \Gamma^*)}$ is a set of *transition rules*. A *configuration* of $\mathcal{P}$ is a pair $\langle p, w \rangle$, where $p \in P$ is a control location and $w \in \Gamma^*$ is a *stack content*. If $((p, \gamma), \{(p_1, w_1), \ldots, (p_n, w_n)\}) \in \Delta$, we write $\langle p, \gamma \rangle \hookrightarrow \{\langle p_1, w_1 \rangle, \ldots, \langle p_n, w_n \rangle\}$ instead. If $n = 1$, we write $\langle p, \gamma \rangle \hookrightarrow \langle p_1, w_1 \rangle$ (braces omitted), and call the rule *non-alternating*. We call $\mathcal{P}$ a *pushdown system* (PDS) if Δ consists only of non-alternating rules.

A *min APDS* is an APDS, in which each rule is equipped with a *weight* which is a natural number; formally, $\mathcal{M} = (\mathcal{P}, f)$, where $\mathcal{P} = (P, \Gamma, \Delta)$ is an APDS and $f : \Delta \rightarrow \mathbb{N}^\infty$ is a function that assigns a value from $\mathbb{N}^\infty$ to each rule in Δ. If $f(\langle p, \gamma \rangle \hookrightarrow$

$\{\langle p_1, w_1\rangle, \ldots, \langle p_n, w_n\rangle\}) = a$, we often write $\langle p, \gamma\rangle \overset{a}{\hookrightarrow} \{\langle p_1, w_1\rangle, \ldots, \langle p_n, w_n\rangle\}$. We sometimes use the term APDS to refer to its min version when it is clear from the context.

Intuitively, a rule $\langle p, \gamma\rangle \overset{a}{\hookrightarrow} \{\langle p_1, w_1\rangle, \ldots, \langle p_n, w_n\rangle\}$ says that, from a configuration c where p is the control location and γ is the top of stack symbol, the computation of the system forks into n parallel computations, each of them starting from the configuration obtained from c by replacing p by p_i and γ by w_i, for all $i \in [n]$. Therefore, a run can be seen as a tree of computations. The height of a run is computed from the weights corresponding to the transition rules by applying $+$ between successive weights and $\max$ on the parallel ones.

Formally, we define the reachability relation $\Rightarrow \subseteq (P \times \Gamma^*) \times \mathbb{N}^\infty \times 2^{P \times \Gamma^*}$ to be the smallest relation such that

- $c \overset{0}{\Rightarrow} \{c\}$, for all $c \in P \times \Gamma^*$,
- if $\langle p, \gamma\rangle \overset{a}{\hookrightarrow} \{\langle p_1, w_1\rangle, \ldots, \langle p_n, w_n\rangle\}$ and $\langle p_i, w_i w\rangle \overset{b_i}{\Rightarrow} C_i$ for some $w \in \Gamma^*$, $b_i \in \mathbb{N}^\infty$, and $C_i \subseteq P \times \Gamma^*$, for each $i \in [n]$, then $\langle p, \gamma w\rangle \xRightarrow{a + \max_{i=1}^{n}\{b_i\}} \bigcup_{i=1}^{n} C_i$.

Given a configuration c and a set of configurations C, we define

$$H(c, C) = \min\{a \in \mathbb{N}^\infty \mid c \overset{a}{\Rightarrow} C\}$$

to be the minimum height of all runs starting from c and reaching (precisely) the set C. The *reachability problem* is to determine whether $T(c, C) < \infty$. If $T(c, C) < \infty$, we say that c is *backwards reachable* from C. We define $pre^*(C) = \{c \in P \times \Gamma^* \mid \exists C' \subseteq C : T(c, C') < \infty\}$ to be the set of all configurations that are backwards reachable from C.

An APDS is called *simple* if there is a set of bottom stack symbols $\Xi \subseteq \Gamma$, and all transition rules in Δ are in the following forms:

- $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$, where $p, p' \in P$, $\gamma \in \Gamma \setminus \Xi$, and $w \in (\Gamma \setminus \Xi)^*$,
- $\langle p, \perp\rangle \hookrightarrow \{\langle p_1, w_1 \perp_1\rangle, \ldots, \langle p_n, w_n \perp_n\rangle\}$, where $\perp, \perp_i \in \Xi$ and $w_i \in (\Gamma \setminus \Xi)^*$ for all $i \in [n]$.

As we shall see later, every APDS in this paper is simple.

Let us fix a min APDS $\mathcal{M} = (\mathcal{P}, f)$, where $\mathcal{P} = (P, \Gamma, \Delta)$. An $\mathcal{M}$-automaton (or simply automaton) is a quintuple $\mathcal{A} = (Q, \Gamma, \delta, P, q_f)$, where $Q \supseteq P$ is a finite set of *states*, $q_f \in Q$ is the *final state*, and $\delta \subseteq Q \times \Gamma \times \mathbb{N}^\infty \times Q$ is a set of transitions. Notice that the *initial states* of $\mathcal{A}$ are the control locations of $\mathcal{P}$. Analogously, a *min $\mathcal{M}$-automaton* equips each rule with a natural number; formally, $\mathcal{B} = (\mathcal{A}, l)$, where $\mathcal{A}$ is an $\mathcal{M}$-automaton and $l : \delta \to \mathbb{N}^\infty$.

We define the *transition relation* $\to \subseteq Q \times \Gamma^* \times \mathbb{N}^\infty \times Q$ as the smallest relation satisfying:

- $q \xrightarrow{\varepsilon(0)} q$ for all $q \in Q$, and
- if $t = (q, \gamma, q') \in \delta$, $l(t) = a$, and $q' \xrightarrow{w(b)} q''$, then $q \xrightarrow{\gamma w(a+b)} q''$.

Given an initial state p and a word w, we define $W(p, w) = \min\{a \in \mathbb{N}^\infty \mid p \xrightarrow{w(a)} q_f\}$ to be the *minimum weight* of the word w when starting from the state p. $\mathcal{B}$ *accepts* or *recognizes* a configuration $\langle p, w\rangle$ if $p \xrightarrow{w(a)} q_f$ such that $a < \infty$. The set of configurations recognized by $\mathcal{B}$ is denoted by $\mathcal{L}(\mathcal{B})$.

In [SSE06], it has been shown that given a set of configurations C of an (unweighted) simple APDS $\mathcal{P}$, recognized by an automaton $\mathcal{A}$, we can construct another automaton $\mathcal{A}_{pre^*}$ such that $\mathcal{L}(\mathcal{A}_{pre^*}) = pre^*(C)$. (Note that the definitions of reachability relation and transition relation in the unweighted case can be defined analogously.) The procedure has been extended in [Suw09] to handle weighted APDSs where weights are abstractly defined. Min APDSs can be seen as a special case in which weights are instantiated for particular applications. For brevity, we will not elaborate on these correspondences any further, and simply apply the appropriate pushdown theory to SPKI/SDSI. In the next section, however, we propose a novel efficient algorithm which can be directly applied to solve the authorization problem and compute minimum-height certificate trees.

4 Computing certificate trees

Certificates in SPKI/SDSI can be interpreted as prefix rewrite systems. For instance, if p `a` $\rightarrow p'$ `b c` and p' `b` $\rightarrow p''$ `d e` are two certificates interpreted as rewrite rules, then their concatenation rewrites p `a` to p'' `d e c`. In SPKI/SDSI with threshold certificates, a concatenation of two or more certificates forms a *certificate tree*. It is easy to see that the authorization problem, given a principal p and p', reduces to the problem of whether there exists a certificate tree that rewrites $p\ \square$ at the root into $p'\ \square$ or $p'\ \blacksquare$ at its leaves.

It has been observed that the type of rewrite systems induced by a set of SPKI/SDSI certificates is equivalent to that of pushdown systems [JR04] and with the presence of threshold certificates to that of simple alternating pushdown systems [SSE06]. Roughly speaking, principals are interpreted as control locations, rôle identifiers as stack alphabet, and certificates as transition rules. For example, a certificate like p `a` $\rightarrow p'$ `b c` is interpreted as a pushdown rule $\langle p, \mathtt{a}\rangle \hookrightarrow \langle p', \mathtt{bc}\rangle$. The SPKI/SDSI authorization problem then reduces to the pushdown reachability problem.

Notice that the corresponding APDSs are simple, since threshold certificates are limited to authorization certificates. Given a SPKI/SDSI system (P, A, C), we have $\Xi = \{\square, \blacksquare\}$. All alternating rules are of the form $\langle p, \square\rangle \hookrightarrow \{p_1, w_1 b_1, \ldots, p_n, w_n b_n\}$, where $p, p_i \in P$, $w_i \in A^*$, and $b_i \in \{\square, \blacksquare\}$ for each $i \in [n]$.

Reachability algorithms for PDSs have been extensively studied in e.g. [BEM97, EHRS00, Sch02]. The algorithms are based on saturation procedures which, given an automaton $\mathcal{A}$ that recognizes a set of configurations C, repeatedly add transitions to $\mathcal{A}$ according to certain conditions until no more transitions can be added, i.e. it is saturated. We propose a similar saturation procedure for min APDSs.

Let us fix a simple min APDS $\mathcal{M} = (\mathcal{P}, f)$, where $\mathcal{P} = (P, \Gamma, \Delta)$, and a min $\mathcal{M}$-automaton $\mathcal{B} = (\mathcal{A}, l_0)$, where $\mathcal{A} = (Q, \Gamma, \delta_0, P, q_f)$, such that $\mathcal{L}(\mathcal{B}) = C$ for some

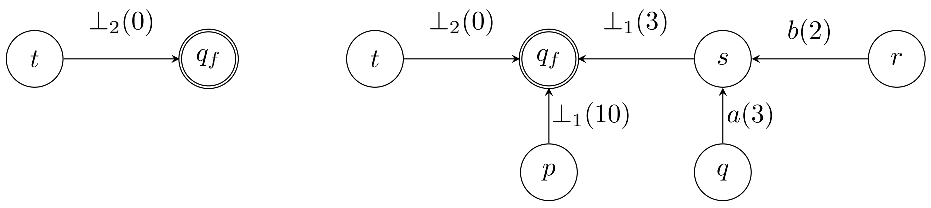

Figure 1: An example: the automata $\mathcal{B}$ (left) and $\mathcal{B}_{pre^*}$ (right)

$C \subseteq P \times \Gamma^*$. Without loss of generality, we assume that $\mathcal{A}$ has no transition leading to an initial state. The following procedure constructs $\mathcal{B}_{pre^*}$ accepting the set of configurations that are backwards reachable from any subset of configurations $C' \subseteq C$ with minimum weights. $\mathcal{B}_{pre^*}$ is defined as $(\mathcal{A}_{pre^*}, l)$, where $\mathcal{A}_{pre^*} = (Q, \Gamma, \delta, P, q_f)$. Initially, $\delta = \delta_0$ and $l(t) = l_0(t)$, if $t \in \delta_0$ and $l(t) = \infty$, otherwise. We iteratively update δ and l according to the following saturation rule until no values can be updated, i.e. until the automaton is saturated

> If $\langle p, \gamma\rangle \overset{a}{\hookrightarrow} \{\langle p_1, w_1\rangle, \ldots, \langle p_n, w_n\rangle\}$ and $p_i \xrightarrow{w_i(b_i)} q$ for all $i \in [n]$, add (p, γ, q) to δ and update $l(t) = \min(l(t), a + \max_{i=1}^{n}\{b_i\})$

Lemma 1 *Given a simple min APDS $\mathcal{M} = (\mathcal{P}, f)$, where $\mathcal{P} = (P, \Gamma, \Delta)$, and a min $\mathcal{M}$-automaton $\mathcal{B} = (\mathcal{A}, l_0)$, where $\mathcal{A} = (Q, \Gamma, \delta_0, P, q_f)$, the saturation procedure constructs $\mathcal{B}_{pre^*} = (\mathcal{A}_{pre^*}, l)$, where $\mathcal{A}_{pre^*} = (Q, \Gamma, \delta, P, q_f)$, such that $\mathcal{L}(\mathcal{B}_{pre^*}) = pre^*(\mathcal{L}(\mathcal{B}))$.*

Given a simple min APDS $\mathcal{M} = (\mathcal{P}, f)$, where $\mathcal{P} = (\{p, q, r, s, t\}, \{a, b, \perp_1, \perp_2\}, \Delta)$, as an example. The rules Δ and the weights f are defined as follows:

$$\langle p, \perp_1\rangle \overset{20}{\hookrightarrow} \langle t, \perp_2\rangle \qquad \langle p, \perp_1\rangle \overset{4}{\hookrightarrow} \{\langle q, a\perp_1\rangle, \langle s, \perp_1\rangle\}$$
$$\langle q, a\rangle \overset{1}{\hookrightarrow} \langle r, b\rangle \qquad \langle r, b\rangle \overset{2}{\hookrightarrow} \langle s, \varepsilon\rangle \qquad \langle s, \perp_1\rangle \overset{3}{\hookrightarrow} \langle t, \perp_2\rangle$$

Figure 1 shows the automaton $\mathcal{B}_{pre^*}$ on the right when applying the saturation procedure to the automaton $\mathcal{B}$ on the left. Notice that the configuration $\langle p, \perp_1\rangle$ is backwards reachable via two possible runs with weights 10 and 20. The weight of the transition $(p, \perp_1, q_f)$ is the minimum one.

An implementation of the saturation procedure is shown in Algorithm 1. The algorithm assumes without loss of generality two restrictions on every rule $\langle p, \gamma\rangle \hookrightarrow R$ in Δ:

(R1) if $R = \{\langle p', w'\rangle\}$, then $|w'| \leq 2$, and

(R2) if $|R| > 1$, then $\forall \langle p', w'\rangle \in R : |w'| = 1$.

In (R1), we call the rule pop rule, normal rule, and push rule when $|w'|$ is 0, 1, and 2, respectively. Note that any APDS can be converted into an equivalent one that satisfies (R1) and (R2) with only a linear increase in size.

Input: Min APDS $(\mathcal{P}, f)$, where $\mathcal{P} = (P, \Gamma, \Delta)$, and min automaton $(\mathcal{A}, l_0)$, where $\mathcal{A} = (Q, \Gamma, \delta_0, P, q_f)$

Output: The saturated min automaton $\mathcal{B}_{pre^*}$

```
procedure update($t, v$)
    $\delta := \delta \cup \{t\}$;
    $v := \min(v, l(t))$;
    if $v \neq l(t)$ then
        $trans := trans \cup \{t\}$;
        $l(t) := v$;

$\delta := \delta_0$; $trans := \delta_0$; $l := \lambda t.\infty$;
forall $t \in \delta_0$ do $l(t) := l_0(t)$;
$\Delta' := \Delta$; $f' := \lambda r.\infty$; $g := \lambda r.\infty$;
forall $r \in \Delta$ do  $f'(r) := f(r)$; $g(r) := 0$;
forall $r = \langle p, \gamma\rangle \hookrightarrow \langle p', \varepsilon\rangle \in \Delta$ do  update($(p, \gamma, p')$, $f'(r)$);
while $trans \neq \emptyset$ do
    remove $t = (q, \gamma', q')$ from trans;
    forall $r = \langle p, \gamma\rangle \hookrightarrow \langle q, \gamma'\rangle \in \Delta'$ do
        update($(p, \gamma, q')$, $f'(r) + \max(g(r), l(t))$);
    forall $r = \langle p, \gamma\rangle \hookrightarrow \langle q, \gamma'\gamma''\rangle \in \Delta'$ do
        add $r' := \langle p, \gamma\rangle \hookrightarrow \langle q', \gamma''\rangle$ to $\Delta'$;
        $f'(r') := \min(f'(r'), f'(r) + l(t))$;
        forall $t' = (q', \gamma'', q'') \in \delta$ do
            update($(p, \gamma, q'')$, $f'(r') + l(t')$);
    forall $r = \langle p, \gamma\rangle \hookrightarrow \{\langle q, \gamma'\rangle\} \cup R \in \Delta'$ s.t. $R \neq \emptyset$ do
        add $r' := \langle p, \gamma\rangle \hookrightarrow R$ to $\Delta'$;
        $f'(r') := \min(f'(r'), f'(r))$;
        $g(r') := \min(g(r'), \max(g(r), l(t)))$;
return $((Q, \Gamma, \delta, P, q_f), l)$;
```

Algorithm 1: A reachability algorithm for min APDSs

Lines 7–11 initialize the algorithm. Initially, $trans$ contains transitions from δ_0 (line 7), and all rules are copied to Δ' (line 9). The auxiliary functions f' and g are initialized to f and 0, respectively (line 10). Pop rules are dealt with first (line 11). The algorithm then repeatedly removes transitions from $trans$ (line 13) until it is empty (line 12), and examines whether they generate other transitions via the saturation rule (lines 14–24). The algorithm calls the procedure `update` (lines 1–6) when a new weight of a transition is computed. The new transition is added to δ (line 2) before computing the new minimum value (line 3). The if-statement at line 4 makes sure that the transition is added to $trans$ for further computation (line 5) only if its weight changes. As a result, line 6 can change $l(t)$ only to a smaller value.

The idea of the algorithm is to avoid unnecessary operations. Imagine that the saturation rule allows to add transition t if transitions t_1 and t_2 are already present. Now, if t_1 is taken from $trans$ but t_2 has not been added to $\mathcal{B}_{pre^*}$, we do not put t_1 back to $trans$ but store the following information instead: if t_2 is added, then we can also add t. It turns out that this can be done by adding new rules to Δ' and storing information in the auxiliary functions f' and g.

Let us now look at lines 14–24 in more detail. Line 14 handles normal rules where new transitions can be immediately added. Push rules (lines 16–20) and alternating rules (lines 21–24), however, require a more delicate treatment. At line 16 we know that (q, γ', q') is a transition of $\mathcal{B}_{pre^*}$ (because it has been removed from $trans$) and that $r = \langle p, \gamma \rangle \hookrightarrow \langle q, \gamma'\gamma'' \rangle$ is a push rule of $\mathcal{P}$. We create the "fake rule" $\langle p, \gamma \rangle \hookrightarrow \langle q', \gamma'' \rangle$ and add it to Δ' at line 17. Its f'-value is updated to be the minimum value of its old value (which is initialized to ∞ at line 9) and $f'(r) + l(t)$ at line 18. Later, when a transition (q', γ'', q'') is examined together with this fake rule at line 14, we update the transition (p, γ, q'') with weight $f'(r) + l(t) + l(q', \gamma'', q'')$. On the other hand, if the transition (q', γ'', q''), for any q'', is already in δ (line 19), we need to update the transition (p, γ, q'') accordingly (line 20).

At line 21 we know that $t = (q, \gamma', q'')$ is a transition of $\mathcal{B}_{pre^*}$ and $r = \langle p, \gamma \rangle \hookrightarrow \langle q', \gamma' \rangle \cup R$ is an alternating rule. Therefore, we add the fake rule $\langle p, \gamma \rangle \hookrightarrow R$ to Δ' (line 22), and minimize its f'-value to $f'(r)$ (line 23) and g-value to the maximum value of $g(r)$ and $l(t)$ (line 24). If the set R contains more than one element, then similar processes can take place which result in more fake rules with less elements in the right-hand sets. Line 14 handles the case when the fake rule is non-alternating. Notice that the auxiliary function g is used when constructing fake rules from alternating rules to store maximum values as specified by the saturation rule. The auxiliary function f' extends the function f by including weights of fake rules. Their values are used in line 15.

Lemma 2 *Algorithm 1 implements the saturation procedure.*

The complexity of the algorithm can be derived from the unweighted result in [EHRS00]. The difference is that Algorithm 1 can process transitions several times, while in [EHRS00] each transition is processed exactly once. Thus, the time complexity increases from $\mathcal{O}(|Q|^2|\Delta|)$ in [EHRS00] by a factor that is no more than the number of transitions and the maximum weight of all transitions. Note that weights of transitions can only decrease, and therefore limit the number of times they can be put in $trans$ and subsequently reprocessed.

Theorem 1 *Given a min SPKI/SDSI and two principal p and p', Algorithm 1 can be used to prove whether $aut(p, p')$ by computing the minimum-height certificate tree with $p\ \square$ at the root and $p'\ \square$ or $p'\ \blacksquare$ at its leaves.*

5 Conclusions

We have proposed an extension to the SPKI/SDSI authorization framework. The extension allows principals to assign weights to certificates, which permits principals to "prioritize" certificates they prefer. We have shown that the process of proving a principal's authorization turns out to be the problem of finding certificate trees. Finding the certificate tree with minimum height (or highest priority) is, however, a more difficult problem. We have given a connection between SPKI/SDSI and pushdown systems, and shown that the problem of computing minimum-height certificate trees reduces to the reachability problem of alternating pushdown systems. We have consequently proposed an algorithm for solving the reachability problem.

References

[BEM97] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *Proc. CONCUR*, 1997.

[CEE$^+$01] Dwaine Clarke, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Morcos, and Ronald L. Rivest. Certificate Chain Discovery in SPKI/SDSI. *Journal of Computer Security*, 9:285–322, 2001.

[EFL$^+$99] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylönen. *RFC 2693: SPKI Certificate Theory*. The Internet Society, 1999.

[EHRS00] Javier Esparza, David Hansel, Peter Rossmanith, and Stefan Schwoon. Efficient Algorithms for Model Checking Pushdown Systems. In *Proc. CAV*, LNCS 1855, pages 232–247, 2000.

[JR04] Somesh Jha and Thomas Reps. Model Checking SPKI/SDSI. *JCS*, 12(3–4):317–353, 2004.

[Sch02] Stefan Schwoon. *Model-Checking Pushdown Systems*. PhD thesis, Technische Universität München, 2002.

[SJRS03] Stefan Schwoon, Somesh Jha, Thomas Reps, and Stuart Stubblebine. On Generalized Authorization Problems. In *Proc. CSFW*, pages 202–218. IEEE, 2003.

[SSE06] Dejvuth Suwimonteerabuth, Stefan Schwoon, and Javier Esparza. Efficient Algorithms for Alternating Pushdown Systems with an Application to the Computation of Certificate Chains. In *Proc. ATVA*, LNCS 4218, pages 141–153, 2006.

[Suw09] Dejvuth Suwimonteerabuth. *Reachability in Pushdown Systems: Algorithms and Applications*. PhD thesis, Technische Universität München, 2009.