Certificate Chain Discovery in SPKI/SDSI

Dwaine Clarke*
Jean-Emile Elien[†]
Carl Ellison[‡]
Matt Fredette[§]
Alexander Morcos[¶]
Ronald L. Rivest[||]

July 30, 2001

Abstract

SPKI/SDSI is a novel public-key infrastructure emphasizing naming, groups, ease-of-use, and flexible authorization. To access a protected resource, a client must present to the server a proof that the client is authorized; this proof takes the form of a "certificate chain" proving that the client's public key is in one of the groups on the resource's ACL, or that the client's public key has been delegated authority (in one or more stages) from a key in one of the groups on the resource's ACL.

While finding such a chain can be nontrivial, due to the flexible naming and delegation capabilities of SPKI/SDSI certificates, we present a practical and efficient algorithm for this problem of "certificate chain discovery." We also present a tight worst-case bound on its running time, which is polynomial in the length of its input.

We also present an extension of our algorithm that is capable of handling "threshold subjects," where several principals are required to co-sign a request to access a protected resource.

*(Note: authors are listed alphabetically.) MIT Lab for Computer Science, `declarke@theory.lcs.mit.edu`

[†]Microsoft, `hbm@alum.mit.edu`

[‡]Intel, `carl.m.ellison@intel.com`

[§]BBN, `fredette@alum.mit.edu`

[¶]`morcos@alum.mit.edu`

[||]MIT Lab for Computer Science, `rivest@mit.edu`

Keywords: certificate, certificate chain, certificate chain discovery, public-key infrastructure, PKI, SPKI, SDSI, naming, local names, authorization, delegation, threshold subjects.

1 Introduction

This paper studies the problem of "certificate chain discovery" within the SPKI/SDSI ("s-p-k-i/sudsy") public-key infrastructure.

The problem addressed here is a fundamental one. Any security mechanism should be able to answer the basic authorization question, "Is principal X authorized to do Y?" The difficulty of answering this question depends primarily on the expressiveness of language used to make elementary security assertions.

If the language used to make security assertions is too flexible, then the authorization question may be undecidable. Harrison, Ruzzo, and Ullman [18] give such an undecidability result for a general protection system based on the access matrix model. (Speaking strictly, their undecidability result is about the more general question of safety rather than just authorization per se.)

On the other hand, Jones, Lipton, and Snyder [21] give an efficient (linear time) algorithm for deciding the authorization question in the *take-grant* model.

More recently, Blaze, Feigenbaum, and Strauss [6] show that "compliance checking" (their term for answering the authorization question) in their PolicyMaker model is in general undecidable, and that it remains NP-hard even when restricted in several natural ways. They also give a polynomial-time algorithm for a special case of their problem. The PolicyMaker scheme has evolved into their "KeyNote" trust management system [5].

We believe that SPKI/SDSI provides an elegant and simple framework for naming and authorization in a distributed environment. Its conceptual framework is natural and easy to understand; it is expressive enough for a large range of applications.

The basic point of the current paper is that the expressive power of SPKI/SDSI does not come at the expense of computational difficulty. We demonstrate here that there is an efficient algorithm for answering the authorization question within SPKI/SDSI.

We imagine the following typical scenario. A client (say, Alice) makes a request to access a resource which (unknown to her) is protected. The server replies that access can only be granted if Alice can prove that she is a member of one of the groups G_1, G_2, or G_3. That is, the access-control list (ACL) for the protected resource specifies that access may only be granted to members of those groups. Alice has a collection of certificates that she may use in her proof. She finds a first certificate C_1 that states that all members of group H are members of group G_2, and another certificate C_2 that states that she (actually, her public key) is a member of group H. The sequence (C_1, C_2) is a "certificate chain" proving that she may access the protected resource. She sends this sequence to the server, signs her request to the protected resource with her private key, and gains access.

Informally the technical problem is the following: given an access-control list for a protected resource, and a collection of SPKI/SDSI certificates, determine whether a given principal or set of principals, represented by their public keys, is authorized to access the protected resource. Because of the way that certificates can be chained in SPKI/SDSI, the problem is non-trivial; the fact that a polynomial-time algorithm exists for this problem is interesting.

The current paper is self-contained but brief, and the reader is encouraged to consult the references for additional background and motivation.

Section 2 gives a brief historical synopsis of the evolution of SPKI/SDSI.

The paper begins by treating the SPKI/SDSI naming subsystem. Section 3 introduces SPKI/SDSI names and gives our favorite representation of name certificates: as "rewrite rules" for transforming one string or certificate into another. Section 4 gives a simple graph-theoretic algorithm for evaluating the meaning of a SPKI/SDSI name in the absence of "extended" names. The case for extended names is made in Section 5, where it is shown how extended names can increase ease-of-use and modularity.

Section 6 then shows how two certificates can be composed to yield another one. This certificate composition operation is the fundamental "inference rule" of SPKI/SDSI. An efficient algorithm for computing the "name-reduction closure" of a given set of certificates is then described, proved correct, and analyzed.

Section 7 introduces authorization certificates, or "auth certs," and shows how they can also be represented as rewrite rules.

Section 8 gives an overview of the general certificate chain discovery problem, by way of a specific example. Section 9 then gives the details of our certificate-chain discovery algorithm, including an analysis of its running time.

Finally, Section 10 discusses how the certificate-chain discovery algorithm can be extended to handle "threshold subjects," where more than one party must sign an access request in order for it to be honored.

We assume the reader has a basic familiarity with public-key cryptography and digital signatures (see for example Menezes et al.[25]), although the details of particular signature schemes are not important here. For convenience and brevity, we say that a message was signed by a public key K_i when we really mean that it was signed by the secret key whose corresponding public key is K_i.

2 SPKI/SDSI History

In 1996 Lampson and Rivest[28] proposed a new public-key infrastructure, called "a Simple Distributed Security Infrastructure," abbreviated "SDSI" , and pronounced "sudsy." Its most interesting feature is probably its decentralized name space. In SDSI, the owner of each public key can create a local name space relative to that key. These name spaces can be linked together in a flexible and powerful manner to enable chains of authorization and define groups of authorized principals.

Concurrently, Carl Ellison, Bill Frantz, Brian Thomas, Tatu Ylonen and others developed a "Simple Public Key Infrastructure," or "SPKI," pronounced "s-p-k-i", which emphasized exceptional simplicity of design and a flexible means of specifying authorizations.

The SDSI and SPKI efforts were both motivated in part by the perceived complexity of the X.509 public-key infrastructure, and also by its perceived lack of power and flexibility.

In 1997 the SDSI and SPKI efforts were merged; the resulting synthesis has been called "SPKI/SDSI." Sometimes, for brevity, it has been called just "SPKI" or just "SDSI," but the reference is now always to the merged design.

A SPKI working group of the IETF was formed in 1996 that has continued to refine the design[20]. Various RFC's and Internet drafts[10, 12, 13, 14] document this work. Two web sites [27, 11] give further pointers to work on SPKI/SDSI.

Several MIT EECS Master's theses [16, 26, 9, 8, 24] have studied various algorithmic and implementation aspects of SPKI/SDSI. Of most relevance is Jean-Emile Elien's master's thesis[9], which focuses on the certificate chain discovery problem and gives an early version of the algorithm presented in this paper. Elien's thesis is especially recommended reading for further background and discussion both of SPKI/SDSI in general and the certificate chain discovery problem in particular. The algorithm presented here is an extension of the one presented in his thesis.

SDSI's naming scheme has generated some interest in its own right; for example, Abadi[1] has studied SDSI's naming scheme in some detail.

Halpern and van der Meyden [17] have also studied SDSI's naming scheme. They critique Abadi's treatment, and have produced a Logic of Local Name Containment and an associated semantics that explicates the operation of SDSI's local names, based on treating (as we also do) the meaning of a name as a set of keys, and treating a name certificate as asserting an inclusion relationship between two such sets.

Howell and Kotz [19] model SDSI's naming scheme within the framework of the Logic of Authentication due to Abadi, Lampson, and others [2, 22], with particular emphasis on the possible advantages and dangers of various proposed extensions to SDSI.

Li [23] shows how to interpret SPKI/SDSI's local names (including authorization certifications and threshold subjects) using logic programs and proves that his interpretation is equivalent to the original SPKI/SDSI definitions; he also shows how to interpret local names as distributed roles.

We note that the terminology used here may differ in small respects from that used in other SPKI/SDSI documentation; we do not expect this to cause the reader any difficulties.

3 SPKI/SDSI Names

We begin with a description of naming within SPKI/SDSI, leaving authorization for later. We do this for several reasons:

- The naming scheme within SPKI/SDSI is a fascinating object of study in its own right, with great flexibility and interesting computational problems.
- The SPKI/SDSI naming scheme is orthogonal to and conceptually separable from the authorization scheme.
- It will be easier to understand the issues arising in the full SPKI/SDSI scheme once the naming subsystem is fully understood.

In SPKI/SDSI there is a *local name space* associated with every public key. There are *no global names* in SPKI/SDSI. (The first version of SDSI [28] did have global names; these were eliminated in the merger of SDSI with SPKI.) A local name is a pair consisting of a public key and an arbitrary identifier.

A public key can sign statements (certificates) binding one of its local names to a value. Values can be specified indirectly in terms of other names, so the name spaces can become linked and interdependent in a flexible and powerful manner.

3.1 Keys.

In SPKI/SDSI, all principals are represented by their public keys. A principal is an individual, process, or active entity whose messages are distinctively recognizable because they are digitally signed by the public key that represents them. It is convenient to say that the principal *is* its public key.

Definition 1 *We let $\mathcal{K}$ denote the set of* public keys.

We typically use $K, K_A, K_B, K', K_1, K_2, \ldots$ to denote specific public keys. We omit discussion of the corresponding secret keys. In particular, as noted earlier, when we say that a message was signed by key K_i, we mean that it was signed by the secret key whose corresponding public key is K_i.

In practice, a key is represented by a data structure that specifies the algorithm name (e.g. RSA with MD5 hashing and OAEP formatting) and the associated parameters (e.g. modulus $n = 3871099\ldots8763$ and exponent $e = 17$). In this paper we use meta-symbols such as K_i to stand for such data structures.

3.2 Identifiers.

Because the most important function of a name is to serve as a mnemonic handle for some human user, it is important that users be able to create names rather freely using well-chosen identifiers.

Definition 2 *An* identifier *is a word over some given standard alphabet. We let $\mathcal{A}$ denote the set of all possible identifiers.*

Our examples use specific identifiers such as `A`, `B`, `Alice`, `Bob`, ..., usually in typewriter font.

3.3 Local names and local name spaces.

Each (public) key has its own associated local name space; there is no global name space or even a hierarchy of name spaces. SPKI/SDSI does not require a "root" or "root key"; it can be built "bottom-up" in a distributed manner from a collection of local name spaces.

Definition 3 *A* local name *is a sequence of length two consisting of a key K followed by a single identifier.*

Example. Typical local names might be "K `Alice`" or "K `project-team`." Here K represents an actual public key.

Notation 1 *We say that the local name "K `A`" belongs to the* local name space *of key K. We let $\mathcal{N}_L$ denote the set of all local names, and let $\mathcal{N}_L(K)$ denote the local name space of key K.*

The original SDSI syntax for the local name "K `A`" was "K's A"; the use of the possessive syntax emphasizes that this local name belongs to K's namespace. While this syntax is appropriately suggestive, we stick to the simpler syntax "K `A`" in this paper.

Local names in different name spaces are unrelated to each other, even if they use the same identifier. Local names may be chosen in an arbitrary manner. In one local name space the identifiers might be people's names, in another name space identifiers might be nicknames, social security numbers, phone numbers, IP addresses, credit-card numbers, organizational role names, committee names, or group names. The owner of the public key can decide arbitrarily what conventions he wishes to use when assigning names.

There are many reasons to use local names:

- To provide a convenient user-friendly handle for referring to another principal. For example, it is much simpler to refer to "`Bob`" than to refer to the Bob's specific public key "RSA-MD5 with parameters $n = 3549\ldots413$ and $e = 17$".

- To provide a level of abstraction that separates the name one uses to refer to the principal from the keys the principal uses, since the latter may change. If Bob changes his key, no certificates that refer to Bob's key by a local name need to change; only those certificates that give his actual public key need to be updated.
- To allow another party to provide the desired definition, by having one name defined in terms of a name defined by another party. For example, Alice can define her "`MIT`" in terms of VeriSign's "`Massachusetts Institute of Technology`".
- To have a name that refers to a collection (or *group*) of principals. Bob can conveniently define groups for various purposes; for example he may define groups "`friends`", "`personnel-committee`", "`EECS-faculty`", or "`sysadmins`".
- To have a name that can be used as an binary attribute—by defining the group of principals that possess that attribute. The state of California might define groups "`age-over-21`", "`state-employee`", "`registered-voter-for-2000`", or "`welfare-recipient`".

3.4 Extended names, names, and terms.

SPKI/SDSI has "extended names" as well as local names. (These are called "compound names" by Li [23].)

Definition 4 *An* extended name *is a sequence consisting of a key followed by* two or more *identifiers.*

A name is thus either a local name or an extended name. Extended names expand the expressive power of SPKI/SDSI, but do not have separate definitions; their meaning is defined in terms of the meaning of related local names.

Example. Typical extended names might be "K `Alice mother`", "K `microsoft engineering windows project-mgr`", or "K `MIT EECS personnel-committee`".

(In the syntax of SDSI 1.0, the first extended name would be represented as K's `Alice`'s `mother`.)

Notation 2 *We let $\mathcal{N}_E$ denote the set of all extended names. We let $\mathcal{N} = \mathcal{N}_L \cup \mathcal{N}_E$ denote the set of all names. We let $\mathcal{N}_E(K)$ denote the set of extended names beginning with key K, and let $\mathcal{N}(K)$, which we call the* name space *of key K, denote the set of all names (local or extended) beginning with key K.*

The SPKI/SDSI "expressions" that we will be dealing with will be called *terms*; intuitively, a term is something that may have a value. In SPKI/SDSI values are always sets of keys.

Definition 5 *We say that a* term *is either a key or a name. We let $\mathcal{T} = \mathcal{K} \cup \mathcal{N}$ denote the set of all terms.*

Section 5 discusses extended names in more detail, and describes their benefits.

3.5 Certificates.

SPKI/SDSI has two types of certificates, or "certs": *name certs*, which provide a definition for a local name, and *authorization certs*, or *auth certs*, which confer authorization on a key or a name.

Compared to X.509 public-key infrastructure schemes [15], our name cert is comparable to an "ID certificate," and to some forms of "attribute certificates", while our auth cert is comparable to an "attribute certificate" that conveys authorization. However, the details and semantics differ significantly, and the reader should not interpret these comments as more than a very crude approximation.

We defer further discussion of auth certs until Section 7, in order to focus for now on naming and name certificates within SPKI/SDSI.

3.6 Name Certificates.

A name cert provides a definition of a local name (e.g. K `A`) belonging to the issuer's (e.g. K's) local name space. Only key K may issue (that is, sign) certificates for names 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; the certificate is signed by K.
- The *identifier* `A` (together with the issuer) determines the local name "K `A`" that is being defined; this name belongs to the local name space $\mathcal{N}_L(K)$ of key K. We emphasize that name certs only define *local names* (with one identifier); extended names are never defined directly, only indirectly.
- The *subject* S is a term in $\mathcal{T}$. Intuitively, the subject S specifies an new additional meaning for the local name "K `A`".
- The *validity specification* V provides additional information allowing anyone to ascertain if the certificate is currently valid, beyond the obvious verification of the certificate signature. Normally, the validity specification takes the form of a validity period (t_1, t_2): the cert is valid from time t_1 to time t_2, inclusive. Sometimes, the validity specification takes the form of an on-line check to be performed. Certificates that are not currently valid can be ignored, so for this paper we presume that all certificates considered are currently valid, and we do not explicitly mention or discuss validity specifications further.

3.7 Valuation function.

We shall be concerned with the *value* of various terms. (Recall that a term is a key or a name.) In SPKI/SDSI, these values are *sets of public keys* (possibly the empty set). The value of a term T is defined relative to a set $\mathcal{C}$ of certificates.

Notation 3 *We let $\mathcal{V}_{\mathcal{C}}(T)$ denote the* value *of a term T with respect to a set $\mathcal{C}$ of certificates. When $\mathcal{C}$ may be understood from context, we may use the simpler notation $\mathcal{V}(T)$. The value of a term is a set of public keys, possibly empty.*

3.8 Value of a key.

A public key is the simplest kind of a SPKI/SDSI term–it is a constant expression evaluating to itself (as a singleton set).

Definition 6 *We define*

$$\mathcal{V}_{\mathcal{C}}(K) = \{K\}$$

for any public key K and any set $\mathcal{C}$ of certificates.

3.9 Value of a local name.

A local name has a value that is a set of public keys; this value may be the empty set, a set containing a single key, or a set containing many keys. This value is determined by one or more name certificates.

A local name, such as K `Alice` , need not have the same meaning as the local name K' `Alice` when $K \neq K'$; the owner of key K may define K `Alice` however he wishes, while the owner of key K' may similarly but independently define K' `Alice` in an arbitrary manner.

A name cert $C = (K, \mathtt{A}, S, V)$ (intuitively, defining local name K `A` in terms of subject S) should be understood as a signed statement by the issuer asserting that

$$\mathcal{V}(K\ \mathtt{A}) \supseteq \mathcal{V}(S)\ ; \qquad (1)$$

that is, every key in the value $\mathcal{V}(S)$ of subject S is also a key in the value $\mathcal{V}(K\ \mathtt{A})$ of local name K `A`.

One name certificate does not invalidate others for the same local name; their effect is cumulative. That is why the above equation says $\mathcal{V}(K\ \mathtt{A}) \supseteq \mathcal{V}(S)$ and not $\mathcal{V}(K\ \mathtt{A}) = \mathcal{V}(S)$; each additional name cert for K `A` may add new elements to $\mathcal{V}(K\ \mathtt{A})$. A local name in SPKI/SDSI may thus, without any special fanfare, represent a *group* of public keys.

We note that the semantics of SDSI local names provided by Halpern and van der Meyden [17] is very similar to our treatment here of the meaning of local names as sets of keys.

Value of an extended name.

Although a name certificate $C = (K, \mathtt{A}, S, V)$ has the explicit function of providing a definition for the local name K `A`, it also, as we now show, gives meaning to related extended names.

Conversely, we may need to utilize the meaning of an extended name in order to interpret a local name. If the subject S of a name certificate is an extended name, then it is necessary to have a definition for the value $\mathcal{V}(S)$ in order to interpret equation (1).

The value of an extended name is implied by the values of various related local names as follows.

Definition 7 *The value of an extended name $KA_1A_2\ldots A_n$ is defined recursively for $n \geq 2$ as:*

$$\mathcal{V}(KA_1A_2\ldots A_n) = \{K'' : K'' \in \mathcal{V}(K'A_n) \text{ for some } K' \in \mathcal{V}(KA_1A_2\ldots A_{n-1})\}\ . \quad (2)$$

An equivalent definition is:

$$\mathcal{V}(K\mathtt{A}_1\mathtt{A}_2\ldots\mathtt{A}_n) = \bigcup_{K'\in\mathcal{V}(K\mathtt{A}_1)} \mathcal{V}(K'\mathtt{A}_2\mathtt{A}_3\ldots\mathtt{A}_n)\ . \quad (3)$$

Example. Let K_0 denote the MIT public key, K_1 denote the EECS public key, and let K_2 denote Rivest's public key. Then

$$\begin{aligned}\mathcal{V}(K_0\ \mathtt{EECS\ rivest}) &\supseteq \mathcal{V}(K_1\ \mathtt{rivest})\\ &\supseteq \{K_2\}\ .\end{aligned}$$

assuming that $K_1 \in \mathcal{V}(K_0\ \mathtt{EECS})$ and $K_2 \in \mathcal{V}(K_1\ \mathtt{rivest})$.

Having taken the necessary step of showing how extended names acquire a meaning in a straightforward manner from the meanings of related local names, we now make precise our definition of the value of a term T.

Definition 8 *We define $\mathcal{V}_{\mathcal{C}}(T)$ for any term T to be the smallest set of public keys that is consistent with any constraints of the form of equations (1) and (2) implied by the name certificates in $\mathcal{C}$.*

Figure 1 gives a typical example; it presents a set $\mathcal{C}$ of name certs and gives $\mathcal{V}_{\mathcal{C}}(T)$ for various terms T.

We note that SPKI/SDSI has no "negative certs"; you can not issue a cert to remove some key from a group.

One can also think of $\mathcal{V}(K\ \mathtt{A})$ as the set of keys that may "speak for" that name—see Lampson et al.[22] for a definition of "speaks for." Any privileges or authorizations that have been given to the name are given to each key in its group. (See Howell and Kotz [19] for an expanded discussion of the relationship between SPKI/SDSI names and the "speaks for" relation, and see Halpern and van der Meyden [17] for a contrary view.)

3.10 Name Certs as Rewrite Rules

Here we explain how to represent a name certificate as a "rewrite rule" operating on strings of symbols. The symbols used are keys and identifiers. A rewrite rule allows one to replace a given sequence of symbols with another.

Rewrite rules are expressive enough to represent both the definitions given by name certs and the delegations expressed by auth certs.

By starting with a given name and performing rewrites in all possible ways (using a given set of certificates), one can determine the value of a name. One can use a similar procedure to find out which keys are authorized to perform a given action, as we shall see in Section 9.

Our representation of name certs as rewrite rules suppresses the validity specification. This omission is justified, since in practice as noted above any certificates that are not currently valid will be set aside initially, and ignored thereafter.

We represent a name certificate $C = (K, A, S, V)$ as the rewrite rule:

$$KA \longrightarrow S\ .$$

We may also write the syntax of a name rule as:

$$\mathcal{K}\mathcal{A} \longrightarrow \mathcal{T}$$

(as in a rule for a context-free grammar, where any key in $\mathcal{K}$ may be followed by any identifier in $\mathcal{A}$, etc.) or even

$$\mathcal{N}_L \longrightarrow \mathcal{T}\ .$$

3.11 A Typical Example

Figure 1 gives an example of a set of name certs and the values of the names it defines.

Name certs issued by Alice:

$$K_A \texttt{ Bob} \longrightarrow K_B \tag{4}$$
$$K_A \texttt{ Carol} \longrightarrow K_B \texttt{ CarolJones} \tag{5}$$
$$K_A \texttt{ Ted} \longrightarrow K_B \texttt{ CarolJones Ted} \tag{6}$$
$$K_A \texttt{ friends} \longrightarrow K_A \texttt{ Bob} \tag{7}$$
$$K_A \texttt{ friends} \longrightarrow K_A \texttt{ Carol} \tag{8}$$
$$K_A \texttt{ friends} \longrightarrow K_A \texttt{ Ted} \tag{9}$$
$$K_A \texttt{ friends} \longrightarrow K_A \texttt{ Bob my-friends} \tag{10}$$

Name certs issued by Bob:

$$K_B \texttt{ Alice} \longrightarrow K_A \tag{11}$$
$$K_B \texttt{ CarolJones} \longrightarrow K_C \tag{12}$$
$$K_B \texttt{ Frank} \longrightarrow K_F \tag{13}$$
$$K_B \texttt{ my-friends} \longrightarrow K_B \texttt{ Alice} \tag{14}$$
$$K_B \texttt{ my-friends} \longrightarrow K_B \texttt{ Frank} \tag{15}$$

Name certs issued by Carol:

$$K_C \texttt{ Ted} \longrightarrow K_T \tag{16}$$

Alice issues a name cert (4) binding her local name "K_A `Bob`" to Bob's key K_B. She defines her local name "K_A `Carol`" indirectly in terms of Bob's local name "K_B `CarolJones`" with cert (5). In (6) she defines her local name "K_A `Ted`" with an extended name "K_B `CarolJones Ted`", linking through both Bob and Carol's local name spaces (via certificates (12) and (16)). In (7)–(9) she defines the group "K_A `friends`" to include Bob, Carol, and Ted, and in (10) she includes everyone in Bob's group "`my-friends`" in her group "`friends`".
In (11)–(13) Bob gives symbolic names "`Alice`", "`CarolJones`", and "`Frank`" to Alice's key K_A, Carol's key K_C, and Frank's key K_F, respectively. In (14)–(15) he defines his group "`my-friends`" to include Alice's and Frank's keys.
In (16) Carol defines her local name "`Ted`" to refer to Ted's key K_T.
It follows that:

$$\mathcal{V}(K_A \texttt{ Bob}) = \{K_B\} \tag{17}$$
$$\mathcal{V}(K_A \texttt{ Carol}) = \{K_C\} \tag{18}$$
$$\mathcal{V}(K_A \texttt{ Ted}) = \{K_T\} \tag{19}$$
$$\mathcal{V}(K_A \texttt{ friends}) = \{K_B, K_C, K_T, K_A, K_F\} \tag{20}$$
$$\mathcal{V}(K_B \texttt{ Alice}) = \{K_A\} \tag{21}$$
$$\mathcal{V}(K_B \texttt{ CarolJones}) = \{K_C\} \tag{22}$$
$$\mathcal{V}(K_B \texttt{ Frank}) = \{K_F\} \tag{23}$$
$$\mathcal{V}(K_B \texttt{ my-friends}) = \{K_A, K_F\} \tag{24}$$
$$\mathcal{V}(K_C \texttt{ Ted}) = \{K_T\} \tag{25}$$

Figure 1: A typical example of name certs.

4 A Simple Case: No Extended Names

In this section we show that it is easy to find the value of a term given a collection of SPKI/SDSI name certs that have no extended names as subjects, that is, when every subject is either just a key or a local name. In practice we expect that many or most certificates will be of this form. In Section 9 we give an efficient algorithm for the general case.

The problem we are concerned with here is the problem of evaluating the meaning $\mathcal{V}_{\mathcal{C}}(T)$ of a term T, given a set of certificates $\mathcal{C}$ that contain local names but no extended names.

Without loss of generality, we assume that T is a local name appearing in some certificate in $\mathcal{C}$. (If T is a key K, or if T does not appear in the certificates at all, then the problem is trivial. In the first case $\mathcal{V}_{\mathcal{C}}(K) = \{K\}$; in the second case $\mathcal{V}_{\mathcal{C}}(T) = \emptyset$.)

The following simple algorithm solves our problem:

- Create a directed graph $G = (V, E)$ by creating a vertex v for each local name or key appearing in $\mathcal{C}$, and an edge from vertex L to vertex R if $\mathcal{C}$ contains a name cert of the form:

  $$L \longrightarrow R$$

  See, for example, Figure 2, which illustrates the graph G arising from the subset of certificates of Figure 1 that contain no extended names.

- Then

  $$\mathcal{V}_{\mathcal{C}}(T) = \{K : (K \in \mathcal{K}) \wedge (T \xrightarrow{*} K)\}$$

  where $T \xrightarrow{*} K$ means that there is a directed path from T to K in G.

The reason this algorithm works is that when there are no extended names present, there is no need to consider any names outside of those already present in the input set of certificates. Thus we need merely to trace dependencies between the local names and keys appearing in the input. When extended names are present, this reasoning no longer applies, as we shall see.

The running time of our algorithm above to find the meaning of a single term T is linear in the size of the input set $\mathcal{C}$ of certificates when the second step above is implemented using an efficient graph-searching algorithm such as depth-first search or breadth-first search. (See Cormen et al. [7] for details.) The same algorithm and running time applies for the simpler problem of determining whether a given K is a member of $\mathcal{V}_{\mathcal{C}}(T)$ for a given term T.

5 Extended names

Given the simplicity of the previous algorithm, one can reasonably ask: "why bother with extended names at all?" Although, as we shall see, extended names can be handled efficiently, it is nonetheless fair to ask if they are worth the extra bother.

For some applications it may indeed be the case that extended names are not really needed, and that by constraining certificates to have only keys or local names as subjects one can simplify things a bit without paying too severe a penalty in terms of expressive power. The system so implemented would be a proper subset of standard SPKI/SDSI.

As an example, consider the certificate set in Figure 1. Certificates (6):

$$K_A \texttt{ Ted} \longrightarrow K_B \texttt{ CarolJones Ted}$$

and (10):

$$K_A \texttt{ friends} \longrightarrow K_A \texttt{ Bob my-friends}$$

are the only certificates that have extended names as subjects. These could conceivably be rewritten as:

$$K_A \texttt{ Ted} \longrightarrow K_C \texttt{ Ted}$$

and:

$$K_A \texttt{ friends} \longrightarrow K_B \texttt{ my-friends} .$$

Yet we would argue that such a style is awkward and exhibits poor modularity. What if Bob or Carol should change their public keys? It would make more sense, from a human-engineering point of view, to

$$\begin{tikzcd}
K_A\ \texttt{friends} \arrow[r] \arrow[dr] \arrow[ddr] & K_A\ \texttt{Bob} \arrow[r] & K_B \\
& K_A\ \texttt{Ted} & \\
& K_A\ \texttt{Carol} \arrow[d] & \\
& K_B\ \texttt{CarolJones} \arrow[r] & K_C \\
K_B\ \texttt{my-friends} \arrow[r] \arrow[dr] & K_B\ \texttt{Alice} \arrow[r] & K_A \\
& K_B\ \texttt{Frank} \arrow[r] & K_F \\
& K_C\ \texttt{Ted} \arrow[r] & K_T
\end{tikzcd}$$

Figure 2: The graph corresponding to the certificates of Figure 1, except for the certificates (6) and (10) which have extended names as subjects. Each key and local name appearing in the certificates corresponds to a vertex, each certificate corresponds to an edge.

have *no* keys, other than the issuer's own key, appearing in local names. From this viewpoint, certificates (5) and (6), which have local names beginning with Bob's key, should instead be rewritten with *even longer* extended names as subjects:

$$\begin{array}{rcl} K_A\ \texttt{Carol} & \longrightarrow & K_A\ \texttt{Bob CarolJones} \\ K_A\ \texttt{Ted} & \longrightarrow & K_A\ \texttt{Bob CarolJones Ted} \end{array}$$

so that these certificates do not need to be re-issued should Bob change his key. (Of course, Bob needs to re-issue his certificates using the same naming conventions...)

Along the same lines, it is simpler to expect someone to write a symbolic ACL entry of the form:

$$K_A\ \texttt{VeriSign IBM Research DonCoppersmith}$$

than to have the actual public key of IBM's research division wired into the ACL entry.

Similarly, it is simpler to write a symbolic ACL entry of the form:

$$K_A\ \texttt{MIT faculty secretary}$$

than to spell out one ACL entry of the form

$$K_F\ \texttt{secretary}$$

for each key K_F of an MIT faculty member (assuming that the faculty can be persuaded to use the name "`secretary`" in a standard way).

Thus, we strongly endorse using extended names whenever possible to improve modularity and simplify the writing of ACL's.

In the next sections we see how to efficiently handle a set of certificates with extended names. This turns out to be an interesting problem, since the simple graph-theoretic model used above is no longer adequate. Instead, we will need to return to our view of certificates as "rewrite rules."

6 Composition of certs

In this section we define how a cert can be used to rewrite a string, or to rewrite another cert. This latter operation is also called the composition of certs (also called composition of rules), and is the fundamental operation of SPKI/SDSI.

In this section we define a "string" to be a term; later on when we are dealing with auth certs we shall expand this definition slightly.

6.1 Using a cert to rewrite a string.

Definition 9 *Suppose that S is a string, that $C = L \longrightarrow R$ is a rewrite rule, and that L is a prefix of S: that is, $S = LX$ for some (possibly empty) sequence X. Then we define $S \circ C$ to be the string $S' = RX$. We say say that we have* rewritten *the string S according to the rule C to obtain S'.*

Example.

$$(K_A\ \texttt{Bob my-friends}) \circ (K_A\ \texttt{Bob} \longrightarrow K_B) = (K_B\ \texttt{my-friends})$$

6.2 Using one cert to rewrite another (composition).

We can also apply a rule C_2 to rewrite another rule C_1 to obtain $C_3 = C_1 \circ C_2$, by using C_2 to rewrite the subject (right-hand side) of C_1.

Definition 10 *Suppose C_1 is a rule of the form*

$$L_1 \longrightarrow R_1 ,$$

and suppose C_2 is a rule of the form

$$L_2 \longrightarrow R_2 ,$$

where L_2 is a prefix of R_1. That is, $R_1 = L_2X$ for some (possibly empty) string X. Then we define the composition of rules *$C_3 = C_1 \circ C_2$ as*

$$\begin{array}{rcl} C_3 & = & C_1 \circ C_2 \\ & = & L_1 \longrightarrow (R_1 \circ C_2) \\ & = & L_1 \longrightarrow R_2X\ . \end{array}$$

We say that we have rewritten *C_1* (using *C_2*) to obtain *C_3. If L_2 is not a prefix of R_1 then $C_1 \circ C_2$ is undefined.*

As an example, we can compose the following name certs:

$$\begin{array}{rcl} K_A\ \texttt{friends} & \longrightarrow & K_A\ \texttt{Bob}\ \texttt{my-friends} \\ K_A\ \texttt{Bob} & \longrightarrow & K_B \end{array}$$

to obtain the name cert:

$$K_A\ \texttt{friends} \longrightarrow K_B\ \texttt{my-friends}\ .$$

That is, if K_A says that one definition of her name "friends" is the name "K_A Bob my-friends", and K_A says that one possible definition of her name "Bob" is K_B, then one definition of K_A's name "friends" is "K_B my-friends".

Definition 11 *We say that certs $C_1 = (L_1 \longrightarrow R_1)$ and $C_2 = (L_2 \longrightarrow R_2)$ are* compatible *if their composition $C_1 \circ C_2$ is defined, that is, if L_2 is a prefix of R_1. (More precisely, if $C_1 \circ C_2$ is defined, we say that C_1 is* left-compatible *with C_2, and that C_2 is* right-compatible *with C_1.)*

The definition implies C_1 is (left-)compatible with C_2 in the special case that $L_2 = R_1$. Note that the definition of compatibility really applies to the *ordered pair* (C_1, C_2), since $C_1 \circ C_2$ may be defined (so that C_1 and C_2 are compatible), but $C_2 \circ C_1$ may be undefined (so that C_2 and C_1 and not compatible). Thus the need for the more refined notions of left- and right-compatibility.

6.3 Properties of composition of certs.

The important point about the composition of certs is that composition is the only "rule of inference" needed for reasoning within SPKI/SDSI. Someone holding valid certs C_1 and C_2 may infer $C_3 = C_1 \circ C_2$ and treat it as a valid cert having the same status as any valid cert that had been actually issued. (As well shall see, these statements remain true even when we add authorization certs to the picture.)

We note that composition is not associative. For example, if

$$\begin{array}{rcl} C_1 & = & (K_1\ \texttt{A} \longrightarrow K_2\ \texttt{B}\ \texttt{C}) \\ C_2 & = & (K_2\ \texttt{B} \longrightarrow K_3) \\ C_3 & = & (K_3\ \texttt{C} \longrightarrow K_4)\ , \end{array}$$

then $(C_1 \circ C_2) \circ C_3 = (K_1\ \texttt{A} \longrightarrow K_4)$, whereas $C_1 \circ (C_2 \circ C_3)$ is undefined because $(C_2 \circ C_3)$ is undefined. However, it is easy to show that $(C_1 \circ C_2) \circ C_3$ is defined whenever $C_1 \circ (C_2 \circ C_3)$ is defined, and that these expressions have equal values when both are defined, so we may omit parentheses when desired and assume that "$\circ$" is left-associative:

$$C_1 \circ C_2 \circ C_3 \cdots C_n = (\cdots((C_1 \circ C_2) \circ C_3) \cdots C_n)\ .$$

We also note that in the composition $C_3 = C_1 \circ C_2$ where C_1 and C_2 were both issued (and not inferred), it may be the case that C_1 was issued before C_2 or the reverse. For example, Bob (controlling key K_B) may have issued the name cert

$$K_B\ \texttt{CarolJones} \longrightarrow K_C$$

either before or after Alice (controlling key K_A) issues the name cert

$$K_A\ \texttt{Carol} \longrightarrow K_B\ \texttt{CarolJones}$$

that specifies her name "Carol" in terms of Bob's name "CarolJones". This gives SPKI/SDSI a certain flexibility lacked by PKI systems that require a key to be created before it can be referred to.

6.4 Closure of a set of certs.

The notion of the closure of a set of certificates is fundamental; the closure contains all certs that can be derived by composition from the given set of certs.

Definition 12 *If $\mathcal{C}$ is a set of certificates, we define the set $\mathcal{C}^+$, called the (transitive)* closure *of $\mathcal{C}$, as the smallest set of certificates that includes $\mathcal{C}$ as a subset and that is closed under composition of certificates.*

Informally, the closure $\mathcal{C}^+$ contains all certificates that can be inferred from $\mathcal{C}$ using any finite number of compositions.

The closure $\mathcal{C}^+$ need not be a finite set, even if $\mathcal{C}$ is finite. For example,

$$\{(K\ \texttt{A} \longrightarrow K\ \texttt{A}\ \texttt{A})\}^+ = \{(K\ \texttt{A} \longrightarrow K\ \texttt{A}^i) : i \geq 2\}\ .$$

While the set $\mathcal{C}^+$ need not be finite, each rule in $\mathcal{C}^+$ has a finite-length derivation from the certificates in $\mathcal{C}$.

As we shall see in Theorems 1 and 2, the closure can be easily used to define the value of any term appearing in the given set of certificates.

We shall next define a *finite* subset of the closure, called the "name-reduction closure," that is easy to compute, and just as useful.

6.5 Reducing certificates.

Given the utility of the closure, as seen above, it is of interest to compute it efficiently, if possible. But since the closure is potentially infinite, to be efficient we need to compute just the relevant parts of it quickly.

This subsection defines the *name-reduction closure* $\mathcal{C}^{\#}$ (i.e. "C-sharp") of a set of certificates $\mathcal{C}$. The name-reduction closure is a *finite* subset of $\mathcal{C}^{+}$, and can be computed quickly. Intuitively, computing $\mathcal{C}^{\#}$ only performs compositions that are useful to compute the value $\mathcal{V}(S)$ of each subject S.

Definition 13 *We say that a cert $C = (L \longrightarrow R)$ is* reducing *if $|L| > |R|$, where $|X|$ denotes the length of sequence X.*

A reducing cert can only be a name cert of the form:

$$K \ \mathtt{A} \longrightarrow K' \ .$$

Fact 1 *If $C_1 = (L_1 \longrightarrow R_1)$ is an arbitrary certificate, and $C_2 = (L_2 \longrightarrow R_2)$ is a (right-)compatible reducing certificate, then $C_3 = C_1 \circ C_2 = (L_1 \longrightarrow R_3)$ satisfies*

$$|R_1| > |R_3| \ .$$

That is, rewriting C_1 with a reducing certificate C_2 gives a new certificate C_3 with a strictly shorter right-hand side.

For example, composing the cert:

$$K \ \mathtt{Alice} \longrightarrow K \ \mathtt{VeriSign} \ \mathtt{MIT} \ \mathtt{AliceSmith} \qquad (26)$$

with the reducing certificate

$$K \ \mathtt{VeriSign} \longrightarrow K_v$$

yields the reduced certificate

$$K \ \mathtt{Alice} \longrightarrow K_v \ \mathtt{MIT} \ \mathtt{AliceSmith}$$

which has a shorter right-hand side than (26).

6.6 Name-reduction closure

Definition 14 *If $\mathcal{C}$ is a set of certificates, then the* name-reduction closure *$C^{\#}$ of $\mathcal{C}$ is defined to be the smallest set of certificates containing $\mathcal{C}$ and closed under "name-reduction" (rewriting with reducing certificates). That is, if $C^{\#}$ contains a certificate C_1 and it also contains a (right-)compatible reducing certificate C_2, then $C^{\#}$ must also contain $C_1 \circ C_2$.*

Thus, to compute the name-reduction closure, we only perform rewritings that cause a reduction in the length of the right-hand side, until no more such rewritings can be done. This is clearly a finite process. More precisely, our algorithm for computing the name-reduction closure is the following:

Name-reduction closure algorithm:

1. Initialize $\mathcal{C}'$ to be the input set $\mathcal{C}$ of certificates.
2. As long as $\mathcal{C}'$ contains two compatible certificates C_1 and C_2 such that C_2 is a reducing certificate and $C_1 \circ C_2$ is not yet in $\mathcal{C}'$, add $C_1 \circ C_2$ to $\mathcal{C}'$.
3. Return $\mathcal{C}'$ as the computed value of $\mathcal{C}^{\#}$.

To illustrate the operation of this algorithm, Figure 3 shows the closure and name-reduction closure of the certificates from Figure 1.

Before studying the running time of this algorithm, we examine some of the properties of $\mathcal{C}^{\#}$.

6.6.1 Properties of the name-reduction closure.

The importance of the name-reduction closure of a set of certificates is given by the following theorems, which shows that the name-reduction closure explicitly computes the values of terms appearing on the right-hand side of the input certificates.

Theorem 1 *Suppose that $\mathcal{C}$ is a set of certificates, and that*

$$C = (L \longrightarrow R)$$

is a name cert in $\mathcal{C}$. Then, for any key $K \in \mathcal{V}_{\mathcal{C}}(R)$, the certificate

$$L \longrightarrow K$$

is a cert in $\mathcal{C}^{\#}$.

The name-reduction closure $\mathcal{C}^{\#}$ of the certificates in Figure 1 includes those certificates as well as the following:

K_A `Carol`	$\longrightarrow$	K_C	$(5) \circ (12) =$	(27)
K_A `Ted`	$\longrightarrow$	K_C `Ted`	$(6) \circ (12) =$	(28)
K_A `Ted`	$\longrightarrow$	K_T	$(28) \circ (16) =$	(29)
K_A `friends`	$\longrightarrow$	K_B	$(7) \circ (4) =$	(30)
K_A `friends`	$\longrightarrow$	K_C	$(8) \circ (27) =$	(31)
K_A `friends`	$\longrightarrow$	K_T	$(9) \circ (29) =$	(32)
K_A `friends`	$\longrightarrow$	K_B `my-friends`	$(10) \circ (4) =$	(33)
K_B `my-friends`	$\longrightarrow$	K_A	$(14) \circ (11) =$	(34)
K_B `my-friends`	$\longrightarrow$	K_F	$(15) \circ (13) =$	(35)
K_A `friends`	$\longrightarrow$	K_A	$(33) \circ (34) =$	(36)
K_A `friends`	$\longrightarrow$	K_F	$(33) \circ (35) =$	(37)

All of the preceding certs are also in the closure $\mathcal{C}^{+}$. The following certs are in $\mathcal{C}^{+}$ but not in the name-reduction closure $\mathcal{C}^{\#}$, since (14) and (15) are not reducing certs:

K_A `friends`	$\longrightarrow$	K_B `Alice`	$(33) \circ (14) =$	(38)
K_A `friends`	$\longrightarrow$	K_B `Frank`	$(33) \circ (15) =$	(39)

Figure 3: The closure and name-reduction closure of the example of Figure 1. The derivation of each certificate is given on the right. For example, certificate (27) is obtained by composing certificates (5) and (12).

Proof: (See Elien's thesis [9] for an earlier version of this theorem and proof.)

If R is a key, it must be the key K, and the theorem is trivial. So assume that R is not a key. Let K be any key in $\mathcal{V}_{\mathcal{C}}(R)$, and suppose that

$$C_1 \circ C_2 \circ \cdots \circ C_n \qquad (40)$$

is a shortest possible certificate chain whose result is $(L \longrightarrow K)$, where all of the certificates are from $\mathcal{C}^{\#}$. Such a chain must exist if $K \in \mathcal{V}_{\mathcal{C}}(R)$, since $\mathcal{C} \subseteq \mathcal{C}^{\#}$.

If $n = 1$ we are done, so assume that $n > 1$.

Note that if $n > 1$, then C_1 is not reducing, since there would be no (right-)compatible certificates C_2 (no name certificates have just a key as their left-hand side). Similarly, C_n must be a reducing certificate, since the right-hand side R_n of C_n is just the key K.

Let C_i be the last non-reducing certificate in the chain; thus $i < n$ and C_{i+1} must be reducing. Therefore the right-hand side R_i of C_i must satisfy $|R_i| \geq 2$, and so C_i and C_{i+1} must be compatible. Therefore $C_i \circ C_{i+1}$ is well defined, and an element of $\mathcal{C}^{\#}$. This implies that the certificate chain (40) is not shortest possible, a contradiction. Therefore $n = 1$ and we are done. ∎

The natural converse of this theorem also holds.

Theorem 2 *Suppose that $\mathcal{C}$ is a set of certificates, and that*

$$C = (L \longrightarrow K)$$

is a cert in $\mathcal{C}^{\#}$. Then there exists a cert

$$C' = (L \longrightarrow R)$$

in $\mathcal{C}$ such that $K \in \mathcal{V}_{\mathcal{C}}(R)$.

Proof: If $C = C_1 \circ C_2 \circ \ldots \circ C_n$, then it follows easily from the definitions and the fact that $C^{\#} \subseteq C^{+}$ that C_1 is the desired certificate C'. ∎

6.6.2 Running time of the name-reduction closure algorithm.

Since the name-reduction closure algorithm is the most critical portion of our certificate-chain discovery algorithm, we carefully analyze its running time in this section.

We believe that this algorithm is very practical, and that it will be exceptionally effective in practice.

In this subsection, we give a polynomial bound on the running time of the name-reduction closure algorithm.

We first give a worst-case bound on the running time, and show that it is tight. We then show that some realistic constraints on the input set of certificates make the running time of the algorithm much better. The algorithm does not change; it just runs faster when the input is not pathological.

Worst-case running time

Let $\mathcal{C}$ be the input set of certificates. Suppose that $\mathcal{C}$ contains n certificates, and that l is the length of the longest subject in any input certificate.

The first step in our analysis is to note that the maximum number of new certificates that can be produced during name-reduction closure is $O(n^2 l)$. (We recall that "O-notation" is used for stating worst-case upper bounds to within an unspecified constant factor; the actual number of new certificates produced in a particular instance may often be substantially less than this worst-case upper bound.)

We prove the bound as follows. A typical input certificate of the form

$$L \longrightarrow K_i A_1 A_2 \ldots A_m \ .$$

can be rewritten by reducing certificates to produce new certificates only of the form

$$L \longrightarrow K_j A_k A_{k+1} \ldots A_m$$

for some j and k. That is, the subject of the resulting certificate consists of some key K_j followed by some suffix of the subject of the input certificate. Since the choice of the starting input certificate, the choice of j, and the length of the suffix are arbitrary, the bound follows.

To see that this bound is tight to within constant factors, consider the following "worst-case" set of certificates:

$$\begin{array}{rcll} KC & \longrightarrow & K_0 A^l B_j \ \text{ for } 0 \leq j < n, & (41) \\ K_0 A & \longrightarrow & K_i \ \text{ for } 0 \leq i < n, & (42) \\ K_i A & \longrightarrow & K_{(i+1) \bmod n} A \ \text{ for } 0 \leq i < n, & (43) \end{array}$$

where A^l denotes l consecutive occurrences of A. Name reduction yields all rules of the form

$$KC \longrightarrow K_i A^k B_j$$

for $0 \leq i < n$, $0 \leq k \leq l$, and $0 \leq j < n$, as well as all reducing rules of the form

$$K_i A \longrightarrow K_j$$

for $0 \leq i < n$ and $0 \leq j < n$.

Theorem 3 *The running time of name reduction closure on an input set of n certificates, where l is the length of the longest subject in any input certificate, is $O(n^3 l)$.*

Proof: There are $O(n^2 l)$ certificates produced. The number of reducing certificates in $\mathcal{C}'$ that are (right-)compatible with any given certificate is $O(n)$: these reducing certificates all have the same issuer and identifier, but have different keys as subjects. Thus for each (input or derived) certificate there is work $O(n)$ to do. This yields our bound of $O(n^3 l)$ on the total work performed. (There are some data structure details required to make this actually work out, but they are relatively straightforward, such as hashing rules by their left-hand sides, or by the key and first symbol of their right-hand sides, etc.) ∎

Unambiguous sets of certificates

In practice, we expect that an input set of certificates will not be as pathological as the input set (41)–(43) above. For example, in practice we expect that an input set of certificates will be *unambiguous.*

Definition 15 *A set $\mathcal{C}$ of certificates is said to be* unambiguous *if any certificate C in $\mathcal{C}^+$ is expressible in at most one way as the result of a certificate chain $C_1 \circ C_2 \circ \cdots \circ C_m$ containing only certificates in $\mathcal{C}$.*

Example. If the certificates (43) are removed from the input set (41)–(43), the certificate set becomes unambiguous, but still generates $O(n^2 l)$ certificates.

Theorem 4 *The running time of name reduction closure on an* unambiguous *input set $\mathcal{C}$ of certificates is proportional to $|\mathcal{C}^\#|$, the total number of certificates in the name-reduction closure of $\mathcal{C}$.*

Proof: No certificate in $\mathcal{C}^\#$ is produced more than once, by the definition of unambiguity. ∎

The running time of our algorithm on an unambiguous set $\mathcal{C}$ of input certificates $\mathcal{C}$, where l is the length of the longest subject in any input certificate, is thus $O(n^2 l)$ (since there are only $O(n^2 l)$ certificates produced), a dramatic improvement of the $O(n^3 l)$ bound for the general case.

We expect even better behavior in practice, as we feel that it will often be the case that $|\mathcal{C}^\#|$ is proportional to $|\mathcal{C}|$, so that the running time of our algorithm will be linear.

6.7 Production of certificate chain

From the computation of the name-reduction closure, we can derive a chain of certificates that demonstrates explicitly how any given certificate is indeed in the closure. The process of reconstructing a certificate chain is primarily one of just working backwards through the computation.

For example, using the certificates of Figures 1 and 3, we have that

$$(38) = (33) \circ (14) = (10) \circ (4) \circ (14) \ ,$$

so the desired certificate chain is just the sequence of certificates $(10), (4), (14)$.

However, there is one issue that needs to be addressed, which is the representation of the certificate chain itself. There are two plausible choices for the format of a certificate chain: a *linear* format and a *compressed* format. Because the compressed format may be exponentially more compact than the linear format, and because it is just as easy to create and process as the linear format, we recommend the compressed format.

The linear format outputs the certificates from $\mathcal{C}$ in an order

$$C_1, C_2, \ldots, C_t$$

such that

$$C = C_1 \circ C_2 \circ \cdots \circ C_t$$

where C is the desired derived certificate.

The problem with this format is that the length t of the certificate chain may be exponential in the size n of the input certificate set.

Example. Consider the following set of certificates:

$$\begin{array}{rcl} KD & \longrightarrow & K_n A_n \\ K_i A_i & \longrightarrow & K_{i-1} A_{i-1} B_i \\ K_0 B_i & \longrightarrow & K_{i-1} A_{i-1} C_i \\ K_0 C_i & \longrightarrow & K_0 \\ K_0 A_0 & \longrightarrow & K_0 \end{array}$$

where $1 \leq i \leq n$. Then the length of the certificate chain proving that

$$KD \longrightarrow K_0 \tag{44}$$

is $2^{n+2} - 2$, by induction. However, this chain is highly repetitious, and can be represented much more compactly, as we now see.

Thus, it is of interest to have a compact format for certificate chains that will be of polynomial size. The following "compressed" format works. Assume that $C_1, \ldots, C_n$ are the input certificates.

$$\begin{array}{rcl} C_{n+1} & = & C_{i_1} \circ C_{j_1} \\ C_{n+2} & = & C_{i_2} \circ C_{j_2} \\ C_{n+3} & = & C_{i_3} \circ C_{j_3} \\ \ldots & & \\ C_{n+t} & = & C_{i_t} \circ C_{j_t} \end{array}$$

Here each i_k and each j_k is an integer in the range 1 to $n + k - 1$. In the compressed format the output is a sequence of lines, where each line shows how a new certificate can be derived by composing two previously input or derived certificates. The final certificate is the desired certificate. This format is the same as that given in Figures 1 and 3.

The size of the compressed format is always polynomial in the size of the input (indeed, it is at most $|\mathcal{C}^{\#}|$). The compressed format is easy to produce and easy to check. See Elien's thesis [9] for some implementation details. This representation is never longer than the linear format, and may be exponentially shorter.

The compressed form is the most logical one to use—it reflects the process we use to do the chain discovery. The linear form is thus unnatural but forced by requiring someone to use just original (verifiable) certificates. The compressed form lets us use derived rules.

We thus recommend the use of the compressed format for efficiency's sake; there is no reason why a polynomial-time computation should have to work for exponential time to produce its output, as it might have to do for the linear format.

7 Auth Certs.

Now that we have mastered SPKI/SDSI naming, we turn our attention to authorization certificates, or "auth certs," and see how to adapt our previous algorithms to handle an input set of certificates that contains both name and auth certs.

We will see how to represent auth certs as rewrite rules in such a way that the composition of certs remains well-defined and satisfies all of the desired properties (including delegation control).

Our final algorithm for determining authorization and computing certificate chains is then a combination of name-reduction closure and a graph-theoretic algorithm resembling that of Section 4.

The function of an auth cert is to grant or delegate a specific authorization from the issuer to the subject. An auth cert C is a signed five-tuple (K, S, d, T, V):

- The *issuer* K is a public key, which signs the cert. The issuer is the one granting a specific authorization.
- The *subject* S is a term in $\mathcal{T}$. The public keys in $\mathcal{V}(S)$ are receiving the grant of authorization.
- The *delegation bit* d, if true, grants each key in $\mathcal{V}(S)$ permission to further delegate to others the authorization it is receiving via this certificate.
- The *authorization specification* or *authorization tag* T specifies the specific permission or permissions being granted. For example, it may specify the right to access a particular web site, read a certain file, or login to a particular machine.
- The *validity specification* for an auth cert is the same as that for a name cert.

For example, we might have an auth cert specifying:

- K is the public key of Bill Gates. Bill Gates is the principal granting the authorization.
- S is "K_A `accounting`", where K_A is the public key of Aardvark Accounting Corporation. The set of public keys in $\mathcal{V}(K_A$ `accounting`$)$ is receiving the authorization.
- $d = 1$; any recipient of this authorization can further delegate this permission (by issuing another authorization cert).
- the authorization tag T = `''read 2000-tax-return''` specifies what operations are authorized by this auth cert.

One auth certificate does not invalidate others; their effect is cumulative. (Again, there are no "negative auth certs"; a permission granted is good until one of the relevant certs expires or becomes invalid.)

SPKI/SDSI auth certs integrate smoothly with SPKI/SDSI name certs; the name certs are used to give useful symbolic names to individual keys or groups of keys, and the auth certs can be used to authorize those keys or groups of keys for specific operations.

We digress for just a moment to talk about authorization tags. The syntax and details of authorization tags are not important to us here; we refer the reader to the relevant documents [11, 13, 14, 12]. We note the following important points about authorization tags:

- An authorization tag can be viewed as a representation of the the set of requests it authorizes.
- Given two authorization tags, it is simple for anyone to form an authorization tag that represents the intersection of the sets of requests the input tags authorize.
- The semantics of a request and of authorization tags are otherwise up to the owner of each protected resource.

To illustrate the second point, the authorization tag

```
(tag (ftp (* set read write)
  (* prefix //www.mit.edu/classes/)))
```

might authorize ftp read and write access to any file in the "`classes`" directory on the `www.mit.edu` server, while the authorization tag

```
(tag (ftp read (* prefix //www.mit.edu/)))
```

allows ftp read access to any file on the same server. The intersection of these two tags is the tag

```
(tag (ftp read
  (* prefix //www.mit.edu/classes/)))
```

When authorization certificates are composed, their authentication tags are intersected, so that the result represents only those rights that are authorized by *both* original certificates.

7.1 Access-control lists.

In a security system with discretionary access control each protected resource has an associated access-control list, or ACL, describing which principals have which permissions to access the resource.

The ease with which SPKI/SDSI allows one to describe groups of principals can make the writing of ACL's rather simple and straightforward. The ACL might typically list a single SPKI/SDSI group and its associated permissions. In some cases several groups might be listed, each with associated permissions.

Although the ACL may seem to be a new kind of data object, it can most naturally be interpreted as a convenient representation of one or more auth certs. We now describe this interpretation.

The issuer of an auth cert in an ACL is the owner of the protected resource. In SPKI/SDSI the special term "*Self*" is used to designate the key of the owner of the resource, although the owner's key could of course be used directly.

The subject of an auth cert in an ACL denotes recipients of the permission. More precisely, if S is the subject of an ACL auth cert, then any request for access to the protected resource that is signed by a key K in $\mathcal{V}(S)$ will be honored (assuming that the request matches that authorization tag as well).

Furthermore, if an ACL auth cert $\mathit{Self} \longrightarrow S$ had the delegation bit turned on, then any auth cert issued by $K \in \mathcal{V}(S)$ for the protected resource can be treated as if it were an original ACL auth cert issued by *Self*. (However, K can not grant access to resources to which it itself does not have access, and it may use auth certs to pass on only a subset of the access rights it has itself.)

7.2 Auth certs as rewrite rules

In this section we see how to represent an auth cert as a rewrite rule, so that we may compose auth certs with each other, or with name certs, in a way that precisely models the desired semantics of SPKI/SDSI. To accomplish this, we add to the auth rewrite rules special "ticket" symbols whose presence enforces the desired behavior.

Definition 16 *The special "ticket" symbols are "$\boxed{1}$" ("a live ticket") or "$\boxed{0}$" ("a dead ticket"). The meta-symbol "$\boxed{\text{z}}$" may be used to represent a "zombie" ticket that may be either live or dead.*

A ticket may be thought of as a convenient artifice to represent a particular authority or permission. (Elien [9] used the "turnstile" symbol "$\dashv$" instead.) Tickets ensure that the composition of certs will have the desired behavior, as we shall see. For the purpose of our rewrite rules, however, a ticket is just a symbol, different than any key or identifier. (Behind the scenes, the ticket represent both the delegation bit and the authorization tag.)

The ticket "$\boxed{1}$" is considered to be "live"—it represents a permission obtained with the delegation bit turned on, so it can be further delegated. The ticket "$\boxed{0}$" is considered to be "dead"—it represents a permission obtained with the delegation bit turned off, so it can not be delegated further.

To represent a ticket that may be either live or dead, we use the meta-symbol "$\boxed{\text{z}}$", the "zombie ticket." The zombie ticket does not actually appear in rewrite rules, but is used when discussing a rewrite rule having a ticket which may be either live or dead.

Definition 17 *A* string *is either a term or a term followed by a ticket.*

Examples of strings.

$$K_A$$
$$K_B \ \boxed{0}$$
$$K_B \ \texttt{Alice friends}$$
$$K_A \ \texttt{MIT EECS faculty} \ \boxed{1}$$

We can now represent an auth certificate $C = (K, S, d, T, V)$ as a rewrite rule. If the delegation bit d is on, allowing propagation, we have the rewrite rule:

$$K \ \boxed{1} \longrightarrow S \ \boxed{1} \ .$$

If the delegation bit d is off, so that delegation is forbidden, we have the rewrite rule:

$$K \ \boxed{1} \longrightarrow S \ \boxed{0} \ .$$

These two forms can be encompassed by one:

$$K \ \boxed{1} \longrightarrow S \ \boxed{\text{z}} \ .$$

The ticket on the left of an auth rewrite rule is always live. The ticket on the right is live if the delegation bit d of the auth cert is on (i.e. 1), otherwise the ticket is dead.

Such an auth cert can be interpreted as "K gives permission to S" (with the delegation bit turned on or off, according to whether the ticket on the right is live or dead).

In particular, an ACL entry is represented by a rewrite rule of the form:

$$\mathit{Self} \ \boxed{1} \longrightarrow S \ \boxed{\text{z}}$$

where the subject S is some term.

7.3 Why auth rules have tickets.

We can now see why auth certs are represented as rewrite rules with tickets. The presence of the tickets prevents the auth cert from being inappropriately used in a composition as a name cert. For example, it is *not* correct, according to the SPKI/SDSI composition rules, to compose the following name and auth certs:

$$\begin{array}{ccc} K_A \ \mathtt{C} & \longrightarrow & K_B \ \mathtt{C} \\ K_B \ \boxed{1} & \longrightarrow & K_B \ \mathtt{D} \ \boxed{1} \end{array}$$

to obtain

$$K_A \ \mathtt{C} \longrightarrow K_B \ \mathtt{D} \ \mathtt{C} \ .$$

Were the tickets not used, this might erroneously be considered a legal composition. With tickets, the two certs are not compatible. This restriction is consistent with the viewpoint that the purpose of an auth cert is to grant permission, and not to rewrite names. Only name certs can be used to rewrite names.

We now extend our previous discussion of the composition of name certs to consider the general composition of two certificates, where either one or both may be auth certs. Our definition of composition is unchanged. (This statement is true when we view certs as rewrite rules; behind the scenes, however, authorization tags are intersected when rules are composed. But we can ignore this detail here, since if two auth certs have tags authorizing a given request, then the intersection of those tags will also honor the given request.)

We note the following properties of the composition $C_3 = C_1 \circ C_2$:

1. The type of C_3, as an auth or name cert, is the same as the type of C_1. (Rewriting can not create or destroy tickets.)
2. If C_2 is an auth cert, then $L_2 = R_1$.
3. If C_1 is a name cert, then so is C_2. (Equivalently, if C_2 is an auth cert, then so is C_1.)
4. If R_1 contains a dead ticket, then C_2 must be a name cert.

To illustrate the second point: two auth certs can be composed, if the subject of the first auth cert is the same as the issuer of the second: composing

$$K_A \ \boxed{1} \longrightarrow K_B \ \boxed{1}$$

with

$$K_B \ \boxed{1} \longrightarrow K_C \ \boxed{1}$$

yields

$$K_A \ \boxed{1} \longrightarrow K_C \ \boxed{1} \ .$$

7.4 How tickets enforce delegation control.

It similarly follows that the distinction between a live ticket "$\boxed{1}$" and a dead ticket "$\boxed{0}$" represents and supports the SPKI/SDSI rules on delegation. A rule having a dead ticket on the right can only be rewritten by name certs, not by auth certs, whereas a rule having a live ticket on the right may be rewritten by either name certs or auth certs; effectively authority may be further delegated using auth certs. Thus, the presence of tickets enforces the SPKI/SDSI rules on delegation.

7.5 Using closure to define which keys are authorized.

The closure $\mathcal{C}^+$ of a set $\mathcal{C}$ of certificates is well-defined even if $\mathcal{C}$ contains auth certs. This closure intuitively captures all of the relevant inferences regarding which keys are authorized.

For example, suppose that the given set $\mathcal{C}$ of certificates includes the certificate $C = (\mathit{Self}\ \boxed{1} \longrightarrow R\ \boxed{\mathrm{z}})$; such a certificate may have been obtained from the ACL for a given resource.

Then every certificate $C' = (\mathit{Self}\ \boxed{1} \longrightarrow K\ \boxed{\mathrm{z}})$ in $\mathcal{C}^+$ that has a key K in its right-hand side and that is obtainable by iteratively rewriting C using certificates in $\mathcal{C}$ specifies a key K that is authorized using a certificate chain beginning with the certificate C.

As before, we can not work directly with $\mathcal{C}^+$, as it is (potentially) infinite. We work instead with the name-reduction closure $\mathcal{C}^\#$ (which is still well-defined) and make appropriate modifications and extensions.

The following theorem is similar to Theorem 1, and derives a key property for the name-reduction closure of a general set of certificates. Note that Theorem 1 remains valid even if $\mathcal{C}$ contains auth certs as well as name certs.

Theorem 5 *If*

$$C = (K \boxed{1} \longrightarrow S \boxed{z})$$

is an auth cert in $\mathcal{C}$, then

$$K \boxed{1} \longrightarrow K' \boxed{z}$$

is a cert in $\mathcal{C}^\#$ for every $K' \in \mathcal{V}_\mathcal{C}(S)$. (Here the two zombies must represent the same ticket.)

Proof: Essentially the same argument holds here as for Theorem 1. ▮

8 Illustration of the certificate-chain discovery problem

A typical instance of the problem we solve in this paper arises as follows. A user Alice tries to access some protected resource X. The guardian or reference monitor for X denies her access, since Alice has not demonstrated that she is authorized to access X. The denial is accompanied by a copy of the ACL for X: a set of auth certs that authorize access for certain keys or names. (If the ACL is itself is protected, Alice can invoke the entire process recursively in order to access the ACL.)

Alice must then use the certs in the ACL, together with certs she already possesses, to prove that she is authorized. She repeats her request, including a "certificate chain" demonstrating that her public key is authorized for X, and signs her request with (the secret key corresponding to) her public key.

As an example, suppose that the ACL for X contains the certs:

$$\begin{array}{rcll} \mathit{Self}\,\boxed{1} & \longrightarrow & K_0\ \texttt{engineering}\,\boxed{1} & (45) \\ \mathit{Self}\,\boxed{1} & \longrightarrow & K_0\ \texttt{finance}\,\boxed{1} & (46) \end{array}$$

and that Alice possesses the following certificates, among others:

$$\begin{array}{rcll} K_0\ \texttt{finance} & \longrightarrow & K_1\ \texttt{accounting} & (47) \\ K_1\ \texttt{accounting} & \longrightarrow & K_1\ \texttt{Bob} & (48) \\ K_1\ \texttt{Bob} & \longrightarrow & K_2 & (49) \\ K_2\,\boxed{1} & \longrightarrow & K_3\ \texttt{Alice}\,\boxed{0} & (50) \\ K_3\ \texttt{Alice} & \longrightarrow & K_4 & (51) \end{array}$$

The ACL (certs (45) and (46)) gives permission to `engineering` and `finance`, as defined by K_0. Name cert (47) states that `accounting` as defined by K_1 is part of `finance` as defined by K_0. Name cert (48) states that `Bob` as defined by K_1 is part of `accounting` as defined by K_1. Name cert (49), issued by K_1, states that one of `Bob`'s public keys is K_2.

Auth cert (50), issued by Bob's key K_2, gives permission to `Alice`, as defined by K_3. Here Bob, as a member of accounting, is passing on his permission to his friend Alice. This is permitted because the previous auth cert (46) had the delegation bit turned on, represented by the live ticket.

Finally, name cert (51), issued by K_3, defines `Alice` to include the public key K_4.

Alice can include the "certificate chain"

$$(46)(47)(48)(49)(50)(51)$$

in her subsequent request to access X, which she also signs with her public key K_4. The guardian for X can examine the certificate chain to conclude that Alice is indeed authorized.

The problem addressed in this paper is the problem of constructing a suitable certificate chain, given a collection of certificates. Some of the certificates correspond to the ACL (and are issued by "*Self*"), and some of the certificates belong to the user (or can be obtained by the user).

This problem bears a superficial resemblance to the problem of finding a path in a graph from "*Self*" to the user's public key, where each certificate corresponds to a single directed edge. The nodes of such a graph correspond to the names and keys occurring in the certificates. In simple examples such as the one above where there are no extended names, such an approach actually works fine, as noted in Section 4.

In other examples this simple graph-theoretic approach fails, because name certs can interact to produce new names not previously appearing in any certificate. For example, the two certificates:

$$\begin{array}{rcl} \mathit{Self}\,\boxed{1} & \longrightarrow & K_0\ \texttt{MIT faculty secretary}\,\boxed{1} \\ K_0\ \texttt{MIT} & \longrightarrow & K_5 \end{array}$$

can be composed to yield the new certificate:

$$\mathit{Self}\,\boxed{1} \longrightarrow K_5\ \texttt{faculty secretary}\,\boxed{1},$$

but "K_5 `faculty secretary`" may be a new name not appearing previously in any of the original certs.

9 Our certificate-chain discovery algorithm

In this section we present our algorithm for certificate chain discovery. It takes as input:

- an initial set $\mathcal{C}$ of certificates,
- an authorization that is desired, and
- a public key K_* for which it is desired to prove that authorization.

The proof to be produced consists of a chain of certificates that allow one to derive "$K_* \boxed{z}$" from "*Self* $\boxed{1}$"; that is, the proof consists of the derivation of the certificate *Self* $\boxed{1} \longrightarrow K_* \boxed{z}$.

It is worth digressing for a moment to discuss the question of where the set $\mathcal{C}$ of certificates comes from. Our working assumption is that the requestor (e.g. Alice) uses the set of certificates already in her possession. If that set is sufficient to prove her authorization, then our algorithm will find a proof of authorization and Alice is happy. If Alice's set of certificates does not imply the desired authorization, then Alice will be frustrated in her attempt to access the desired resource, since no suitable proof of authorization will be found by our algorithm (since none exists based on her certificate set). In this case, Alice may need to intervene personally to obtain sufficient additional certificates. For example, if Alice is frustrated in attempting to access her hospital medical records, she may naturally need to ask the hospital to issue her a certificate authorizing such access. Of course, if access to certificates is itself controlled, then the problem becomes much more complicated (we do not address such complexities here).

One could consider a distributed scenario where the input set of certificates resides on a variety of servers throughout the Internet. An algorithm based on depth-first search for this version of the problem has been presented by Ajmani in his Master's thesis [3]; it is however incomplete (it is not guaranteed to find a proof even if one exists). Aura [4] presents a related two-way distributed search algorithm for the case we discuss above in Section 4 when there are no extended names. We do not consider the distributed chain-discovery scenario or the problem of obtaining the relevant certs in a distributed environment further here, but it is a promising direction for future research.

We return to the presentation of our algorithm, assuming that a suitable auxiliary procedure has been used to obtain the input set of certificates. Our algorithm has the following steps.

1. **Remove useless certificates.**
   - Remove from $\mathcal{C}$ any certificate that is not currently valid, or which fails a required on-line check. That is, remove a certificate C from $\mathcal{C}$ if the validity specification V of C shows that the certificate is invalid.
   - Similarly, remove from $\mathcal{C}$ any auth cert C whose authorization tag T is not equal to, or does not include, the desired authorization. These certs are of no use in trying to derive the desired certificate chain.

2. **Name reduction.** Compute the name-reduction closure $\mathcal{C}^\#$ of $\mathcal{C}$.

   Recall that $\mathcal{C}^\#$ includes, for each auth certificate
   $$K \boxed{1} \longrightarrow S \boxed{z}$$
   in $\mathcal{C}$, where S is a name, a certificate of the form
   $$K \boxed{1} \longrightarrow K' \boxed{z} ,$$
   for each key $K' \in \mathcal{V}_{\mathcal{C}}(S)$.

3. **Remove all names and name certs.** Let $\mathcal{C}'$ be $\mathcal{C}^\#$ with all name certificates removed and all auth certs removed that do not have just a single key as their subject. The only certs remaining have the form:
   $$K_i \boxed{1} \longrightarrow K_j \boxed{z} \qquad (52)$$
   (where K_i may be "*Self*").

4. **Remove useless auth certs.** Let $\mathcal{C}''$ be $\mathcal{C}'$ after removing all certs of the form (52) having $K_j \neq K_*$ and a dead ticket on the right; such certs are useless for finding the desired certificate chain.

5. **Use depth-first search (DFS) to find a path.** Set up a graph with one vertex for each key, and an edge from K_i to K_j if there is an auth certificate C in $\mathcal{C}''$ of the form (52).

   Use depth-first search to determine if there is a path from *Self* to K_* . If not, terminate with failure.

6. **Reconstruct the certificate chain.** From the information computed in the previous steps, output the desired certificate chain.

We now give some details in the following subsections.

9.1 Using Depth-First-Search to find a path

After name and name cert elimination, we work with all certificates (original or derived) of the form (52) for various i and j. There are at most n^2 such certificates, since there are at most n keys appearing as issuer and n keys occurring in the subjects of the original set of n rules. Another bound on the number of such certificates is of course $|\mathcal{C}^{\#}|$; this may be considerably less than $O(n^2)$.

We wish to find if there is a path from *Self* (a particular distinguished key) to K_* (the user's key).

This graph problem can be solved by depth-first search in time proportional to the number of certificates of the form (52) that we are working with [7].

9.2 Running-time analysis

The running time of the certificate-chain discovery algorithm is bounded by the size $|\mathcal{C}^{\#}|$ of the name reduction closure $\mathcal{C}^{\#}$ computed in step 2. As derived in Section 6.6.2, this running time is bounded above $O(n^3 l)$ for a general set of n certificates with subjects having length at most l. However, the running time improves to $O(|\mathcal{C}^{\#}|)$ which is bounded by $O(n^2 l)$ if the certficate set $\mathcal{C}$ is unambiguous.

This completes our presentation of the basic certificate chain discovery algorithm and its analysis.

10 Threshold subjects

We now extend our algorithm to handle threshold subjects in auth certs. Threshold subjects can be used to specify a requirement that "k out of m" keys must sign a request in order that the request should be honored. (More precisely, the public keys signing the request must belong to k out of m groups; there may be fewer than k keys signing the request if some keys belong to more than one of the m groups.) The scenario is otherwise much as before: a set of parties Alice$_1$, Alice$_2$, ..., Alice$_n$ attempt to determine if they are authorized if they collectively sign an access request, based on a set of certificates that may contain auth certs with threshold subjects.

Definition 18 *A* threshold subject *is an expression of the form*

$$\theta_k(S_1, S_2, \ldots, S_m)\ \boxed{z}$$

where $1 \leq k \leq m$ and where each S_i is a term or another threshold subject.

Here k is the threshold value; at least k of the m subjects must sign an access request.

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

As an example of a threshold subject, consider the following certificate:

$$\mathit{Self}\ \boxed{1} \longrightarrow \theta_2(\begin{array}{l} K_0\ \texttt{mit faculty}, \\ K_0\ \texttt{intel researcher}, \\ K_0\ \texttt{Alice}\ \ )\ \boxed{1}\,. \end{array} \qquad (53)$$

This certificate requires that keys representing at least two of the three names sign an access request; equivalently with two of the three groups (MIT faculty, Intel researcher, or Alice) represented. (If Alice is an MIT faculty member, then her signature alone is good enough; otherwise two keys must be used to sign the request.)

Our previous algorithm can be adapted to handle threshold subjects; this technique follows closely the algorithm presented by Elien[9].

As noted, there is now not just a single signer K_* on the request, but a *set* $\mathcal{K}_*$ of signers; we want to determine if this set of signers is authorized.

The procedure is as follows.

1. **Rewrite threshold subjects.** Rewrite each auth cert with a threshold subject so that the threshold subject contains no names. This is done by introducing new dummy placeholder keys at each position in the threshold subject. For example, the auth cert (53) could be rewritten as

$$\mathit{Self}\ \boxed{1} \longrightarrow \theta_2(K_{201}, K_{202}, K_{203})\ \boxed{1} \qquad (54)$$

where K_{201}, K_{202} K_{203} represent new public keys that do not appear elsewhere in the set of certificates.

2. **Preserve semantics by adding new auth certs.** Add additional auth certs so to preserve the semantics of the original (now rewritten) auth cert. In this example we would add the certs:

$$K_{201}\ \boxed{1} \longrightarrow K_0\ \texttt{mit faculty}\ \boxed{1}$$
$$K_{202}\ \boxed{1} \longrightarrow K_0\ \texttt{intel researcher}\ \boxed{1}$$
$$K_{203}\ \boxed{1} \longrightarrow K_0\ \texttt{Alice}\ \boxed{1}$$

(If the original auth certificate had a dead ticket instead of a live one on the right-hand side, then these certificates would also have dead tickets on their right-hand sides.)

3. **Eliminate names and name certs.** Temporarily set aside the rewritten auth certs of the form (54), so that we have a set of certs containing no threshold subjects whatsoever. We now apply steps 1–3 of our standard algorithm of Section 9 to eliminate all names and name certs. Adding back the threshold certs originally set aside, we now have just a set of auth certs, each of which has as a subject either a key or a threshold subject on a list of keys.

4. **Remove useless auth certs.** Remove any auth cert that whose right-hand side is "$K_j\ \boxed{0}$" where K_j is not a member of the set $\mathcal{K}_*$ of keys that may participate in this access request.

5. **Label all keys.** Label each key in $\mathcal{K}_*$ as "marked"; label all others as "unmarked".

6. **Propagate labels.** Until no more progress can be made, iterate the following:

   - If the key K_j is marked, and there is an auth cert $K_i\ \boxed{1} \longrightarrow K_j\ \boxed{\text{z}}$, then mark K_i.
   - If there is an auth cert of the form

   $$K_i\ \boxed{1} \longrightarrow \theta_k(K_{i1}, K_{i2}, \ldots, K_{im})\ \boxed{\text{z}}$$

   where at least k of the keys $K_{i1}, K_{i2}, \ldots, K_{im}$ are marked, then mark K_i.

7. **Finish.** A certificate chain is found if *Self* is now marked.

With a bit of care, this algorithm can be implemented as a modified DFS algorithm running in linear time; the running time of this modified algorithm is unchanged in the worst case (if we let n denote the number of subjects appearing in the input set of certificates collectively), since it is linear in the number of vertices and edges of the graph.

The output format of the certificate chain needs to be extended slightly to handle the threshold subjects. We leave this detail to the SPKI/SDSI standards documents.

11 Conclusions

We have presented an efficient algorithm for computing certificate chains for SPKI/SDSI. Thus, SPKI/SDSI has an efficient procedure for answering the fundamental question, "Is A authorized to do X?". While SPKI/SDSI is very expressive, its expressiveness does not come at the price of intractability; sets of SPKI/SDSI certificates are easy to work with.

Acknowledgments
We thank DARPA (contract DABT63-96-C-0018) and NASA for their support.

References

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

[2] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, September 1993.

[3] Sameer Ajmani. A trusted execution platform for multiparty computation. Master's thesis, EECS Department, MIT, September 2000. See `http://www.pmg.lcs.mit.edu/~ajmani /projects.html#thesis`.

[4] Tuomas Aura. Fast access control decisions from delegation certificate databases. In *Proc. 3rd Australasian Conference on Information Security and Privacy ACISP '98*, volume 1438 of *LNCS*, pages 284–295, Brisbane, Australia, July 1998. Springer Verlag. See `http://www.tcs.hut.fi/Publications/papers /aura/aura-acisp98-abstract.html`.

[5] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. KeyNote: Trust management for public-key infrastructures. In *Proc. Cambridge 1998 Security Protocols International Workshop*, pages 59–63, 1998. See also IETF RFC 2704.

[6] Matt Blaze, Joan Feigenbaum, and Martin Strauss. Compliance checking in the PolicyMaker trust management system. In Ray Hirschfeld, editor, *Proc. Second International Conf. on Financial Cryptography, FC '98*, pages 254–274. Springer, 1998.

[7] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990.

[8] Gillian Elcock. Web-based user interface for a Simple Distributed Security Infrastructure (SDSI). Master's thesis, EECS Dept., Massachusetts Institute of Technology, June 1997. See `http://theory.lcs.mit.edu/~cis/theses/ elcock-masters.ps`.

[9] Jean-Emile Elien. Certificate discovery using SPKI/SDSI 2.0 certificates. Master's thesis, EECS Dept; Massachusetts Institute of Technology, May 1998. See `http://theory.lcs.mit.edu/~cis/ theses/elien-masters.pdf`.

[10] Carl M. Ellison. *RFC 2692: SPKI requirements*. The Internet Society, September 1999. See `ftp://ftp.isi.edu/in-notes/rfc2692.txt`.

[11] Carl M. Ellison. SPKI/SDSI certificate documentation. See `http://world.std.com/~cme/html/spki.html`., 2001.

[12] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. *RFC 2693: SPKI Certificate Theory*. The Internet Society, September 1999. See `ftp://ftp.isi.edu/in-notes/rfc2693.txt`.

[13] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. *Simple Public Key Certificate*. The Internet Society, March 1998. See `http://www.clark.net/pub/cme/spki.txt`; This is `draft-ietf-spki-cert-structure-05.txt`.

[14] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. *SPKI Examples*. The Internet Society, March 1998. See `http://www.clark.net/pub/cme/examples.txt`; This is `draft-ietf-spki-cert-examples-01.txt`.

[15] Warwick Ford and Michael S. Baum. *Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption*. Prentice-Hall, 1997.

[16] Matthew H. Fredette. An implementation of SDSI–the Simple Distributed Security Infrastructure. Master's thesis, EECS Dept., Massachusetts Institute of Technology, May 1997. See `http://theory.lcs.mit.edu/~cis/theses/ fredette-masters.ps`.

[17] Joseph Y. Halpern and Ron van der Meyden. A logic for SDSI's linked local name

spaces. In *Proceedings 12th IEEE Computer Society Security Foundations Workshop (CSFW-12)*, pages 111–122, 1999. Available at: `http://www.cs.cornell.edu/home/halpern/abstract.html#conf110`.

[18] M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8), 1976.

[19] Jon Howell and David Kotz. A formal semantics for spki. In *Proceedings of the Sixth European Symposium on Research in Computer Security (ESORICS 2000)*, pages 140–158, 2000. Lecture Notes in Computer Science #1895, Available at: `http://www.cs.dartmouth.edu/~jonh/research/delegation/esorics-abs.pdf`.

[20] Internet Engineering Task Force. Simple Public Key Infrastructure (spki). See `http://www.ietf.org/html.charters/spki-charter.html`., 1997.

[21] A. K. Jones, R. J. Lipton, and L. Snyder. A linear time algorithm for deciding security. In *Proc. FOCS '76*, pages 33–41. IEEE, 1976.

[22] Butler Lampson, Martín Abadi, Michael Burrows, , and Edward Wobber. Authentication in distributed systems: Theory and practice. *TOCS*, 10(4):265–310, November 1992.

[23] Ninghui Li. Local names in SPKI/SDSI. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW-13)*, pages 2–15. IEEE Computer Society Press, 2000.

[24] Andrew J. Maywah. An implementation of a secure web client using SPKI/SDSI certificates. Master's thesis, EECS Dept; Massachusetts Institute of Technology, June 2000. See `http://theory.lcs.mit.edu/~cis/theses/maywah-masters.ps`.

[25] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[26] Alexander Morcos. A Java implementation of Simple Distributed Security Infrastructure. Master's thesis, EECS Dept., Massachusetts Institute of Technology, May 1998. See `http://theory.lcs.mit.edu/~cis/theses/morcos-masters.ps`.

[27] Ronald L. Rivest. Cryptography and Information Security Group Research Project: A Simple Distributed Security Infrastructure (SDSI). See `http://theory.lcs.mit.edu/~cis/sdsi.html`., 1996.

[28] Ronald L. Rivest and Butler Lampson. SDSI–a simple distributed security infrastructure. See `http://theory.lcs.mit.edu/~rivest/sdsi10.ps`., August 1996.