# Reducing the Dependence of Trust-Management Systems on PKI

Hao Wang, Somesh Jha, Thomas Reps
*Computer Science Department*
*University of Wisconsin*
{hbwang, jha, reps}@cs.wisc.edu

Stefan Schwoon
*Institut für Formale Methoden der Informatik*
*Universität Stuttgart*
schwoosn@fmi.uni-stuttgart.de

Stuart Stubblebine
*Stubblebine Research Labs*
stuart@stubblebine.com

Technical Report
UW-CS-TR-1527

### Abstract

Trust-management systems address the authorization problem in distributed systems by defining a formal language for expressing authorization and access-control policies, and relying on an algorithm to determine when a specific request can be granted. For authorization in distributed systems, trust-management systems offer several advantages over other approaches, such as support for delegation and making authorization decisions in a decentralized manner. This paper focuses on a popular trust-management system SPKI/SDSI. Although SPKI/SDSI is an attractive system for authorization in distributed systems, it has seen limited deployment. One of the major hurdles in deploying SPKI/SDSI is that it is PKI-based, i.e., every principal is required to have a public-private key pair. We present an approach that combines SPKI/SDSI with a widely-deployed authentication system, Kerberos, to reduce reliance of SPKI/SDSI on PKI. In our approach, only sites need public-private key pairs. We believe that reducing the reliance of SPKI/SDSI on PKI will facilitate its wider deployment. We also have implemented a prototype of our technique.

## 1 Introduction

Systems with shared resources use access-control mechanisms for protection. There are three fundamental problems in access control: *authentication*, *authorization*, and *enforcement*. Authentication deals with verifying the identity of a principal. Authorization addresses the following problem: should a request $r$ by a specific principal $A$ be allowed? Enforcement addresses the problem of implementing the authorization during an execution. In a centralized system, authorization is based on the closed-world assumption, i.e., all of the parties are known or their identity can be established using an authentication system. In a distributed system, the closed-world assumption is not valid. Trust-management systems [2] solve the authorization problem in distributed systems by defining a formal language for expressing authorization and access-control policies, and relying on an algorithm to determine when a specific request is allowable. Hence, trust-management systems decouple specification of the security policy from enforcement. Therefore, in the context of authorization in distributed systems, trust-management systems offer several advantages, such as support for delegation, no (conceptual) requirement for a central authority, and the ability to make authorization decisions in a truly distributed fashion. A survey of trust-management systems, along with a formal framework for understanding them, is presented in [25]. Two prominent trust-management systems are Keynote [1] and SPKI/SDSI [8].

In this paper, we focus on the trust-management system SPKI/SDSI; however, the basic ideas introduced in this paper are applicable to other trust-management systems. In SPKI/SDSI, *name certificates* define the names available in an issuer's local name space; *authorization certificates* grant authorizations, or delegate the ability to grant authorizations. SPKI/SDSI has been investigated by several researchers. Authorization decisions in SPKI/SDSI are based on *certificate chains*, which are proofs that a client's public key is one of the keys that has been authorized to access a given resource—either directly or transitively, via one or more name-definition or authorization-delegation steps. Despite its many advantages, SPKI/SDSI has seen limited deployment. The two major hurdles in deploying SPKI/SDSI are:

- The need for a distributed certificate-chain discovery algorithm.
- The need for every principal to have a public-private key pair.

The first hurdle has been partially addressed by work on distributed certificate-chain discovery algorithms for trust-management systems.[1] In this paper we focus on the second hurdle. The goal of this paper is to *reduce the dependence of SPKI/SDSI on public-key infrastructure (PKI).* Reducing the reliance of SPKI/SDSI on PKI will hopefully lead to its wider deployment.

Our main approach is to leverage an existing widely deployed authentication system, namely, Kerberos [20]. Specifically, we demonstrate that using SPKI/SDSI in conjunction with Kerberos reduces the reliance on PKI by requiring only one public-private key pair per site, whereas "vanilla" SPKI/SDSI requires each principal to have a public-private key pair. Our approach maintains all the advantages of SPKI/SDSI, such as support for delegation and the ability to make authorization decisions in a distributed fashion. In our approach, there are two levels of certificates; each level resembles vanilla SPKI/SDSI. We call these K-SPKI/SDSI (for SPKI/SDSI with Kerberos), and E-SPKI/SDSI (for extended SPKI/SDSI). Users work at the K-SPKI/SDSI level; E-SPKI/SDSI is the implementation level. The distinction between these levels is discussed further in Section 3.1. In our solution, we allow authenticated Kerberos users to issue K-SPKI/SDSI certificates, and therefore eliminate the requirement that every user possess a public/private key pair. Our K-SPKI/SDSI server accepts certificates from authenticated Kerberos users and generates corresponding E-SPKI/SDSI certificates on behalf of the users. These certificates are used by the K-SPKI/SDSI server for answering authorization queries (by invoking certificate-chain discovery at the E-SPKI/SDSI level). By providing a solution that is built on top of Kerberos, an authentication system that is widely deployed in various organizations, ranging from research institutes to corporations, we hope that our approach will be easier to adopt.

The contributions of this paper are as follows:

- We show how to reduce the dependence of SPKI/SDSI on PKI by leveraging Kerberos.
- This insight behind the work is that when a user authenticates using Kerberos, they acquire a session key that serves as evidence of who they are. This property is uses as a substitute for the signing actions that vanilla SPKI/SDSI requires, which provides the ability to participate in a SPKI/SDSI-like scheme without the requirement to have a public-private key pair.

[1] Prior work (i.e., [15, 17]) has addressed systems similar to, but not identical to, SPKI/SDSI. In our work, we used a distributed certificate-chain discovery algorithm that generalizes the non-distributed algorithm of Jha and Reps [12].

- We have created a prototype that implements the technique. Our measurements show that performance depends on how K-SPKI/SDSI certificates are distributed among sites.

Background on SPKI/SDSI is given in Section 2. Our method for combining SPKI/SDSI and Kerberos is described in Section 3. Some applications of our system are discussed in Section 4. Section 5 discusses deployment and performance issues of our prototype. Section 6 discusses related work.

## 2 Background on SPKI/SDSI

In SPKI/SDSI, all *principals* are represented by their public keys, i.e., the principal *is* its public key. A principal can be an individual, process, host, or any other entity. $\mathcal{K}$ denotes the set of public keys. Specific keys are denoted by $K, K_A, K_B, K'$, etc. An *identifier* is a word over some alphabet $\Sigma$. The set of identifiers is denoted by $\mathcal{A}$. Identifiers will be written in typewriter font, e.g., $\mathtt{A}$ and $\mathtt{Bob}$. A *term* is a key followed by zero or more identifiers. Terms are either keys, local names, or extended names. A *local name* is of the form $K\ \mathtt{A}$, where $K \in \mathcal{K}$ and $\mathtt{A} \in \mathcal{A}$. For example, $K\ \mathtt{Bob}$ is a local name. Local names are important in SPKI/SDSI because they create a decentralized name space. The local name space of $K$ is the set of local names of the form $K\ \mathtt{A}$. An *extended name* is of the form $K\ \sigma$, where $K \in \mathcal{K}$ and $\sigma$ is a sequence of identifiers of length greater than one. For example, $K\ \mathtt{UW}\ \mathtt{CS}\ \mathtt{faculty}$ is an extended name.

### 2.1 Certificates

SPKI/SDSI has two types of certificates, or "certs":

**Name Certificates** (or *name certs*): A name cert provides a definition of a local name in the issuer's local name space. Only key $K$ may issue or sign a cert that defines a name in its local name space. A name cert $C$ is a signed four-tuple $(K, \mathtt{A}, S, V)$. The issuer $K$ is a public key and the certificate is signed by $K$. $\mathtt{A}$ is an identifier. The subject $S$ is a term. Intuitively, $S$ gives additional meaning for the local name $K\ \mathtt{A}$. $V$ is the *validity specification* of the certificate. Usually, $V$ takes the form of an interval $[t_1, t_2]$, i.e., the cert is valid from time $t_1$ to $t_2$ inclusive.

**Authorization Certificates** (or *auth certs*): An auth cert grants or delegates a specific authorization from an issuer to a subject. Specifically, an auth cert $c$ is a five-tuple $(K, S, D, T, V)$. The *issuer* $K$ is a public key, which is also used to sign the cert. The *subject* $S$ is a term. If the *delegation bit* $D$ is turned on, then a subject receiving this authorization can delegate this authorization to other keys. The *authorization specification* $T$ specifies the permission being granted; for example, it may specify a permission to read a specific file, or a permission to login to a particular host. The *validity specification* $V$ for an auth cert is the same as in the case of a name cert.

A *labeled rewrite rule* is a pair $(L \longrightarrow R, T)$, where the first component is a rewrite rule and the second component $T$ is an authorization specification. For notational convenience, we will write the labeled rewrite rule $(L \longrightarrow R, T)$ as $L \xrightarrow{T} R$. We will treat certs as labeled rewrite rules:[2]

- A name cert $(K, \mathtt{A}, S, V)$ will be written as a labeled rewrite rule $K\ \mathtt{A} \xrightarrow{\top} S$, where $\top$ is the authorization specification such that for all other authorization specifications $t$, $\top \cap t = t$, and $\top \cup t = \top$.[3] Sometimes we will write $\xrightarrow{\top}$ as simply $\longrightarrow$, i.e., a rewrite rule of the form $L \longrightarrow R$ has an implicit label of $\top$.

[2] In authorization problems, we only consider valid certificates, so the validity specification $V$ for a certificate is not included in its rule.

[3] The issue of intersection and union of authorization specifications is discussed in detail in [8, 10].

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

## 2.2 Authorization

Because we only use labeled rewrite rules in this paper, we refer to them as rewrite rules or simply rules. A term $S$ appearing in a rule can be viewed as a string over the alphabet $\mathcal{K} \cup \mathcal{A}$, in which elements of $\mathcal{K}$ appear only in the beginning. For uniformity, we also refer to strings of the form $S\ \square$ and $S\ \blacksquare$ as terms. Assume that we are given a labeled rewrite rule $L \xrightarrow{T} R$ corresponding to a cert. Consider a term $S = LX$. In this case, the labeled rewrite rule $L \xrightarrow{T} R$ applied to the term $S$ (denoted by $(L \xrightarrow{T} R)(S)$) yields the term $RX$. Therefore, a rule can be viewed as a function from terms to terms that rewrites the left prefix of its argument, for example,

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

Consider two rules $c_1 = (L_1 \xrightarrow{T} R_1)$ and $c_2 = (L_2 \xrightarrow{T'} R_2)$, and, in addition, assume that $L_2$ is a prefix of $R_1$, i.e., there exists an $X$ such that $R_1 = L_2X$. Then the *composition* $c_2 \circ c_1$ is the rule $L_1 \xrightarrow{T \cap T'} R_2X$. For example, consider the two rules:

$$\begin{aligned} c_1 :&\quad K_A\ \texttt{friends} \xrightarrow{T} K_A\ \texttt{Bob}\ \texttt{myFriends} \\ c_2 :&\quad K_A\ \texttt{Bob} \xrightarrow{T'} K_B \end{aligned}$$

The composition $c_2 \circ c_1$ is $K_A\ \texttt{friends} \xrightarrow{T \cap T'} K_B\ \texttt{myFriends}$. Two rules $c_1$ and $c_2$ are called *compatible* if their composition $c_2 \circ c_1$ is well defined.[4]

## 2.3 The Authorization Problem in SPKI/SDSI

Assume that we are given a set of certs $\mathcal{C}$ and that principal $K$ wants access specified by authorization specification $T$. The authorization question is: “Can $K$ be granted access to the resource specified by $T$?”

A *certificate chain* $ch$ for $\mathcal{C}$ is of a sequence of certificates $[c_1, c_2, \cdots, c_k]$ in $\mathcal{C}$. The label of a certificate chain $ch = [c_1, \cdots, c_k]$ (denoted by $L(ch)$) is the label obtained from $c_k \circ c_{k-1} \cdots \circ c_1$ (denoted by $compose(ch)$). We assume that the authorization specification $T$ is associated with a unique principal $K_{owner[T]}$ (the resource to which $T$ refers). Given a set of certificates $\mathcal{C}$, an authorization specification $T$, and a principal $K$, a *certificate-chain-discovery* algorithm looks for a finite set of certificate chains that “prove” that principal $K$ is allowed access specified by $T$.

Formally, certificate-chain discovery attempts to find a finite set $\{ch_1, \cdots, ch_m\}$ of certificate chains such that for all $1 \leq i \leq m$

$$compose(ch_i)(K_{owner[T]}\ \square) \in \{K\ \square, K\ \blacksquare\}\ ,$$

and $T \subseteq \bigcup_{i=1}^{m} L(ch_i)$.

[4] In general, the composition operator $\circ$ is not associative. For example, $c_3$ can be compatible with $c_2 \circ c_1$, but $c_3$ might not be compatible with $c_2$. Therefore, $c_3 \circ (c_2 \circ c_1)$ can exist when $(c_3 \circ c_2) \circ c_1$ does not exist. However, when $(c_3 \circ c_2) \circ c_1$ exists, so does $c_3 \circ (c_2 \circ c_1)$; moreover, the expressions are equal when both are defined. Thus, we allow ourselves to omit parentheses and assume that $\circ$ is right associative.

Clarke et al. [6] presented an algorithm for certificate-chain discovery in SPKI/SDSI with $O(n_K^2|\mathcal{C}|)$ time complexity, where $n_K$ is the number of keys and $|\mathcal{C}|$ is the sum of the lengths of the right-hand sides of all rules in $\mathcal{C}$. However, this algorithm only solved a restricted version of certificate-chain discovery: a solution could only consist of a *single* certificate chain. For instance, consider the following certificate set:

$$\begin{array}{ll} c_1: & (K,\ K_A,\ 0,\ \texttt{((dir /etc) read)},\ [t_1, t_2]) \\ c_2: & (K,\ K_A,\ 0,\ \texttt{((dir /etc) write)},\ [t_1, t_2]) \end{array}$$

Suppose that Alice makes the request

$$(K_A, \texttt{((dir /etc) (* set read write))}).$$

In this case, the chain $[c_1]$ authorizes Alice to read from directory `/etc`, and a separate chain $[c_2]$ authorizes her to write to `/etc`. Together, the set $\{[c_1], [c_2]\}$ proves that she has both read and write privileges for `/etc`. However, both of the certificates $c_1$ and $c_2$ would be removed from the certificate set prior to running the certificate-chain discovery algorithm of Clarke et al., because `read` $\not\supseteq$ `(* set read write)` and `write` $\not\supseteq$ `(* set read write)`. Consequently, no proof of authorization for Alice's request would be found. Schwoon et al. [22] presented algorithms for the full certificate-chain-discovery problem, based on solving reachability problems in weighted pushdown systems (WPDSs). Their formalization allows a proof of authorization to consist of a set of certificate chains. This paper uses the WPDS-based algorithm for certificate-chain-discovery introduced by [22].

# 3 SPKI/SDSI and Kerberos

We describe the authorization scenario in SPKI/SDSI. In Section 3.1 we describe how the reliance of SPKI/SDSI on PKI can be reduced by using Kerberos. First, we introduce a small example that will be used throughout this section.

**Example 3.1.** Imagine that there are two sites, $Bio$ and $CS$, which correspond to the biology and the computer science department respectively. Let us say that professor $Bob$ in the biology department wants to provide access to a server $V$ to all his students and students of professor $Alice$ in the computer science department.

Assume that there are two sites $st_1$ and $st_2$ that have SPKI/SDSI servers $S_{st_1}$ and $S_{st_2}$, respectively. In the context of our example, sites $CS$ and $Bio$ have SPKI/SDSI servers $S_{CS}$ and $S_{Bio}$. There are three components to a SPKI/SDSI authorization scenario.

**Certificate issuance.** Each user sends signed auth and name certs to the SPKI/SDSI server at their site. The SPKI/SDSI server verifies the signatures on the certs. If signature verification fails on a cert, it is rejected; otherwise it is stored by the SPKI/SDSI server. In our example, $Alice$ sends to $S_{CS}$ the following name certs, which are signed by $Alice$:

$$\boxed{\begin{array}{l} K_{Alice}\ \texttt{students} \longrightarrow K_X \\ K_{Alice}\ \texttt{students} \longrightarrow K_Y \\ K_{Alice}\ \texttt{students} \longrightarrow K_Z \end{array}}$$

The three name certs essentially state that $X$, $Y$, and $Z$ are students of $Alice$. $Bob$ sends to $S_{Bio}$ the following signed auth certs, which are signed by $Bob$:

$$\boxed{\begin{array}{l} K_{Bob}\,\Box \xrightarrow{T_V} K_{Bob}\ \texttt{students}\ \blacksquare \\ K_{Bob}\,\Box \xrightarrow{T_V} K_{Alice}\ \texttt{students}\ \blacksquare \end{array}}$$

The two auth certs state that students of $K_{Alice}$ and $K_{Bio}$ can access server $V$ (denoted by authorization specification $T_V$), but the students cannot delegate this right.

**Certificate-chain discovery.** Suppose a user $U$ (with public key $K_U$) at site $A$ wants to access a resource at site $B$ according to authorization specification $T$. User $U$ sends a certificate-chain-discovery request for $T$ (denoted by $CCDrequest(K_U, T)$) to the SPKI/SDSI server $S_{CS}$. The SPKI/SDSI server $S_{CS}$ executes a distributed certificate-chain-discovery algorithm and returns a finite set of certificate chains $\{ch_1, \cdots, ch_m\}$ to $U$. In Example 3.1, suppose that user $X$ sends the certificate-chain discovery request $CCDrequest(K_X, T_V)$ to server $S_{CS}$. Server $S_{CS}$ executes a distributed certificate-chain discovery algorithm and returns the set of chains $\{ch_1\}$, where $ch_1 = [c_1, c_2]$ ($c_2$ and $c_1$ are shown below.)

$$\begin{array}{rcl} c_2 & = & K_{Bob}\,\Box \xrightarrow{T_V} K_{Alice}\ \texttt{students}\ \blacksquare \\ c_1 & = & K_{Alice}\ \texttt{students} \longrightarrow K_X \end{array}$$

**Requesting a resource.** Assume that user $U$ wants to access a resource according to authorization specification $T$. First, $U$ requests that certificate-chain discovery be carried out by sending a request $CCDrequest(K_U, T)$ to the SPKI/SDSI server at its site, and obtains back a set of certificate chains $SCH = \{ch_1, \cdots, ch_m\}$. User $U$ presents the set of certificate chains $SCH$ to the principal $K_T$ (recall that $K_T$ is the owner of the resource to which $T$ refers). The principal $K_T$ authorizes $U$ iff $T \subseteq \bigcup_{i=1}^{m} L(ch_i)$ (this step is usually called *compliance checking*). The label $L(ch_i)$ of a chain $ch_i$ is described in Section 2.2.

In Example 3.1, user $X$ wants access to server $V$ according to the authorization specification $T_V$. After making a certificate-chain discovery request, $X$ obtains the set $\{ch_1\}$, where $ch_1 = [c_1, c_2]$, $compose(ch_1)(K_{Bob}\Box) \in \{K_X\,\Box, K_X\,\blacksquare\}$, and $T_V \subseteq L(ch_1)$. $X$ presents $\{ch_1\}$ to server $V$. $V$ checks that $T_V \subseteq L(ch_1)$, which is true, and hence $V$ grants $U$ access.

## 3.1 SPKI/SDSI and Kerberos

Notice that, to use SPKI/SDSI, *every user* needs to have public/private key pair. In this section, we describe an authorization protocol that uses a distributed authentication system, such as Kerberos, but only requires a public/private key pair *per site*. Our new authorization system is called K-SPKI/SDSI.

We assume that the reader is familiar with Kerberos (for a detailed description of Kerberos see [20]). We make the following assumptions:

- Each site is a Kerberos realm. The KDC at site $st$ is denoted by $KDC_{st}$.
- The K-SPKI/SDSI server at each site is Kerberoized.
- The KDC and the K-SPKI/SDSI server at a site $st$ share a public/private key pair. The public key of site $st$ is denoted by $K_{st}$.

Next we describe all three components of our authorization scenario in the new context.

**Certificate issuance.** To issue K-SPKI/SDSI certificates, a Kerberos user first authenticates with the local KDC using the standard Kerberos authentication protocol and receives a Ticket Granting Ticket (TGT) from the KDC. Using the TGT, the client requests a Service Granting Ticket (SGT) for accessing the Kerberoized SPKI/SDSI (K-SPKI/SDSI) server. Throughout the rest of the section, assume that the user has obtained an SGT for the K-SPKI/SDSI server at its site. Using the SGT, the client issues requests for generating SPKI/SDSI name certs or auth certs. Communication on the channel over which the requests are sent is encrypted using the session key $K_s$ provided in the SGT. To issue a name cert, a user $U$ at site $st$ sends an encrypted name cert request to the SPKI/SDSI server:

$$E_{K_s}[U, \mathtt{A}, S, V] \ ,$$

where $U$ is the name of the user, $\mathtt{A}$ is an identifier, $S$ is a subject, and $V$ is a validity specification. As before, we will write the name cert $E_{K_s}[U, \mathtt{A}, S, V]$ as $U \ \mathtt{A} \longrightarrow S$. Upon receiving the encrypted name cert $E_{K_s}[U, \mathtt{A}, S, V]$ the local K-SPKI/SDSI server ascertains its validity, and if the name cert is valid, it creates a new name cert of the form $[K_{st} \ U, \mathtt{A}, K_{st} \ S, V]$, signs it with its private key, and stores it in the database of certificates. Notice that in the new name cert the public key $K_{st}$ of site $st$ is added before $U$ and $S$. In our example, $Alice$ sends the following name certs encrypted with the session key $K_s$ to the K-SPKI/SDSI server at its site.

$$\boxed{\begin{array}{l} \mathtt{Alice\ students} \longrightarrow \mathtt{X} \\ \mathtt{Alice\ students} \longrightarrow \mathtt{Y} \\ \mathtt{Alice\ students} \longrightarrow \mathtt{Z} \end{array}}$$

The K-SPKI/SDSI server verifies the encrypted name certs shown above and creates the following E-SPKI/SDSI name certs and signs them.

$$\boxed{\begin{array}{l} K_{CS} \ \mathtt{Alice\ students} \longrightarrow K_{CS} \ \mathtt{X} \\ K_{CS} \ \mathtt{Alice\ students} \longrightarrow K_{CS} \ \mathtt{Y} \\ K_{CS} \ \mathtt{Alice\ students} \longrightarrow K_{CS}\mathtt{Z} \end{array}}$$

A user $U$ at site $st$ sends an auth cert $E_{K_s}[U, S, D, T, V]$ encrypted with the session key from the TGT to the K-SPKI/SDSI server. Upon receiving the encrypted auth cert $E_{K_s}[U, S, D, T, V]$ the K-SPKI/SDSI server ascertains its validity, and if the auth cert is valid, it creates a new E-SPKI/SDSI auth cert of the form $[K_{st} \ U, K_{st} \ S, D, T, V]$ signs it with its private key, and stores it in the database of certificates. In our example, $Bob$ sends the following auth certs encrypted with the session key from the TGT to the K-SPKI/SDSI server $S_{Bio}$.

$$\boxed{\begin{array}{l} \mathtt{Bob} \ \square \xrightarrow{T_V} \mathtt{Bob\ students} \ \blacksquare \\ \mathtt{Bob} \ \square \xrightarrow{T_V} \mathtt{CS\ Alice\ students} \ \blacksquare \end{array}}$$

The two auth certs state that students of $Bob$ (at the current site) and $Alice$ (at site $CS$) can access server $V$ (denoted by authorization specification $T_V$), but the students cannot delegate this right. The K-SPKI/SDSI server $S_{Bio}$ verifies the encrypted auth certs shown above, and creates the following E-SPKI/SDSI auth certs, and signs them.

$$\boxed{\begin{array}{l} K_{Bio} \ \mathtt{Bob} \ \square \xrightarrow{T_V} K_{Bio} \ \mathtt{Bob\ students} \ \blacksquare \\ K_{Bio} \ \mathtt{Bob} \ \square \xrightarrow{T_V} K_{Bio} \ \mathtt{CS\ Alice\ students} \ \blacksquare \end{array}}$$

The K-SPKI/SDSI servers also adds name certs corresponding to the K-SPKI/SDSI servers of other sites. In our example, $S_{CS}$ signs and adds the name cert $K_{CS} \ \mathtt{Bio} \longrightarrow K_{Bio}$, which states that the public key of the site $Bio$ is $K_{Bio}$. Similarly, $S_{Bio}$ signs and adds the name cert $K_{Bio} \ \mathtt{CS} \longrightarrow K_{CS}$.
**Note:** K-SPKI/SDSI servers must support an extended version of SPKI/SDSI: the left-hand sides of extended auth and name certs have three symbols; the left-hand side of an extended auth cert is of the form

$K_\alpha\mathtt{U}\square$ or $K_\alpha\mathtt{U}\blacksquare$, where $K_\alpha$ is the public key of site $\alpha$ and $U$ is a user; the left-hand side of an extended name cert if of the form $K_\alpha\mathtt{U}\ \mathtt{A}$, where both U and A are identifiers. However, in SPKI/SDSI the left-hand sides of auth and name certs have just two symbols. Various SPKI/SDSI algorithms must be extended to implement E-SPKI/SDSI; however, this is possible because E-SPKI/SDSI is a special case of left-prefix rewriting, and the primitives generalize to arbitrary left-prefix rewriting systems [4].

**Requesting a resource.** Using the SGT, a user $U$ at site $st_1$ sends a request to the local K-SPKI/SDSI server, asking to access the remote server $V$ located in a different site $st_2$. This request is encrypted using the session key $K_s$ provided by the SGT.

$$E_{K_s}[st_2, V, T]$$

The K-SPKI/SDSI server $S_{st_1}$ at site $st_1$ initiates a distributed certificate-chain discovery request $CCDrequest(K_{st_1}\ U, T)$ on behalf of $U$. This process involves K-SPKI/SDSI servers, both local and remote, that contain related E-SPKI/SDSI certificates. If the request $CCDrequest(K_{st_1}\ U, T)$ is successful and returns a set of certificate chains SCH, user $U$ receives the following token from $S_{st_1}$.

$$\begin{aligned} Token_U &= E_{K_s}(K_1)\, Ticket_U \\ Ticket_U &= E_{K_{st_2}}(K_2)\ E_{K_2}[st_2, K_1, V, T, \mathrm{SCH}, TS_1, Lifetime_1] \end{aligned}$$

In Example 3.1, user $X$ receives a token with the set of certificate chains SCH $=$ $\{ch_1\}$, where $ch_1$ is the certificate chain $[c_1, c_2, c_3]$. Certificates $c_1$, $c_2$, and $c_3$ are shown below.

$$\begin{aligned} c_1 &= K_{Bio}\ \mathtt{Bob}\ \square \xrightarrow{T_V} K_{Bio}\ \mathtt{CS}\ \mathtt{Alice}\ \mathtt{students}\ \blacksquare \\ c_2 &= K_{Bio}\ \mathtt{CS} \longrightarrow K_{CS} \\ c_3 &= K_{CS}\ \mathtt{Alice}\ \mathtt{students} \longrightarrow K_{CS}\ \mathtt{X} \end{aligned}$$

Notice that $compose([c_1, c_2, c_3])(K_{Bio}\mathtt{Bob}\ \square) \in \{K_{CS}\ \mathtt{X}\ \square, K_{CS}\ \mathtt{X}\ \blacksquare\}$ and $T_V \subseteq L([c_1, c_2, c_3])$.

Upon receiving $Token_U$, user $U$ decrypts $E_{K_s}(K_1)$ and retrieves the key $K_1$ (recall that $K_s$ is the session key in the TGT for the K-SPKI/SDSI server at site $st_1$). User $U$ constructs the following authenticator:

$$Authenticator_U = E_{K_1}[ID_U \| AD_U \| TS_2 \| Lifetime_2]$$

User $U$ sends the following message to the server $V$ at site $st_2$:

$$Ticket_U\ \ Authenticator_U$$

Server $V$ requests its local K-SPKI/SDSI server to verify the message. The K-SPKI/SDSI server at site $st_2$ performs the following steps:

- Decrypts the message $E_{K_{st_2}}(K_2)$ with its private key, and retrieves the session key $K_2$.
- Decrypts the message $E_{K_2}[st_2, K_1, V, T, \mathrm{SCH}, TS_1, Lifetime_1]$ and ascertains its freshness using the time-stamp $TS_1$. Moreover, the server verifies using $Lifetime_1$ that the token has not expired. The K-SPKI/SDSI server also performs the compliance-checking step on the set of certificate chains SCH.
- Similarly, the K-SPKI/SDSI server ascertains the validity of the authenticator $E_{K_1}[ID_U \| AD_U \| TS_2 \| Lifetime_2]$. Notice that the server knows the session key $K_1$ from $Ticket_U$.

If all the steps given above are successful, then the K-SPKI/SDSI server sends a message to $V$ indicating that $U$ should be granted access.

### 3.2 Threat Analysis

The message exchange for requesting a resource described earlier is very similar to the exchange of messages between the client and KDC in Kerberos. In essence, the authenticator $Authenticator_U$ states that "anyone who uses $K_1$ is $U$". Notice that since in the token $Token_U$ the session key $K_1$ is encrypted with $K_s$, which can only be known by the user $U$ (because $K_s$ is in the SGT issued to $U$). Therefore, assuming the authentication in Kerberos is correct, only $U$ could have known $K_1$. An adversary can still replay the message $Ticket_U$ $Authenticator_U$ to the server $V$ and masquerade as $U$. Since the authenticator is intended for use only once, it can have a very short lifetime, and hence the risk of a replay attack is minimal.

## 4 Applications

Our work is only a first step towards building a practical distributed authorization system. In this section, we discuss how existing applications can benefit from the K-SPKI/SDSI approach.

### 4.1 Authorization for Distributed File Systems

AFS [21] is a popular distributed file system in active use because it provides users with a consistent name space, regardless of the users's physical location. Authorization in AFS is an important issue because all AFS users share the same view of the entire AFS. Currently, AFS relies on Kerberos for authentication, and uses Access Control Lists (ACLs), which may contain user names, group names, or both, to control who can access the data inside each AFS directory. The ACL system works well for managing permissions within one site; however, if users from different sites plan to share files, the ACL system becomes less efficient to use and is difficult to maintain. This is because in AFS, authorization is performed locally at each AFS cell.

To illustrate this, we use a concrete example to show how AFS ACLs work in a cross-site environment. Then we explain how our approach can simplify cross-site authorization in AFS.

**Cross-site ACLs in AFS.** Let us assume that Professor Bob, from the site `Bio`, plans to grant access to directory `data` to students of Professor Alice from the site `CS`. Using existing AFS, Alice and Bob must follow these steps to accomplish this goal: [5]

1. *Initial authentication:* At site `Bio`, Bob authenticates with the AFS system (through Kerberos) to obtain an AFS token that is used to access AFS and create ACLs.
2. *Alice creates a list of her students:* At site `CS`, Alice creates a list of her students, as shown in Figure 1 (a). [6]
3. *Bob sets up a group for Alice's students:* Bob creates an AFS group, called `Alice@CS:students`, for Alice's students in `Bio`. He then populates the group with Alice's students. This is shown in Figure 1 (b).
4. *Bob grants access to Alice's students:* Bob grants access to directory `data` to the group that he created in the previous step, as shown in Figure 1 (c).

The problem with the above approach is that the group `Alice@CS:students`, maintained by Bob, is redundant and must be kept in sync with Alice's own list. As a result, this approach is not efficient and does not scale. Consider the following two scenarios:

[5] In addition, the AFS server from `CS` must set up a special group called `system:authuser@cs` for this to work.

[6] We are not concerned with how this list is created and maintained. The relevant issue here is that someone from `CS` needs to create and maintain a list of Alice's students.

| Alice’s students (CS) |
|---|
| studentX |
| studentY |
| studentZ |
| ⋮ |

**(a.) Alice’s list of her students**

| Alice@CS:students |
|---|
| studentX@cs |
| studentY@cs |
| studentZ@cs |
| ⋮ |

**(b.) Bob’s ACL group**

⇒

| **ACL list for** `data` | |
|---|---|
| system:administrators | rlidwka |
| system:anyuser | l |
| Alice@CS:students | rl |
| Bob | rlidwka |

**(c.) Bob grants access**

Figure 1: An example of using AFS ACLs for cross-site access control. In (a), Alice maintains a list of her students at site `CS`. In (b), at site `Bio`, Bob creates the group `Alice@CS:students` and adds Alice’s students to the group. In (c), Bob grants access to directory `data` to the group `Alice@CS:students` (third entry); the other three ACL entries are managed by AFS.

- If Alice adds or removes a student from her list, then Bob must also update his list accordingly.
- If three more professors from three different sites plan to grant access permissions to Alice’s students, then each of these professors must also create and maintain a copy of Alice’s student list.

**AFS Authorization Using K-SPKI/SDSI.** In contrast, the K-SPKI/SDSI approach greatly simplifies cross-site authorization. Here are the steps:

1. *Initial authentication:* Alice and Bob authenticate with their KDCs to obtain SGTs for their respective K-SPKI/SDSI servers.
2. *Alice issues name certs:* At site `CS`, using the K-SPKI/SDSI server, Alice issues a name cert for each one of her students:
$$K_{CS}\ \texttt{Alice students} \longrightarrow K_{CS}\ \texttt{studentX}$$
This essentially creates a group called $K_{CS}$ `Alice students` at site `CS`.
3. *Bob grants access to Alice’s students:* At site *Bio*, through the K-SPKI/SDSI server, Bob issues *one* auth cert granting access to Alice’s students, using the group created at `CS`:
$$K_{Bio}\ \texttt{Bob}\ \Box \xrightarrow{\texttt{data rl}} K_{Bio}\ \texttt{CS Alice students}\ \blacksquare$$

Here, because Alice is the only one who manages the list of students, and Bob refers to the symbolic name `CS Alice students`, Bob no longer needs to create a copy of Alice’s list of students. As a result, this approach is simpler to use and is also more scalable in cross-site environments:

- When Alice adds or removes a student from her list, Bob does not need to make any changes on his side.
- If three more professors from three different sites want to grant access permissions to Alice’s students, all they need to do is to issue auth certs at their sites, and no duplicate lists of students need to be created.

### 4.2 Accessing Kerberos Services through Web Services

Web services are traditionally built on top of public-key cryptography, such as SSL. Kerberos services, on the other hand, rely on secret-key cryptography. As a result, it has been a challenge to integrate these two systems so that users can access Kerberos services through web services. In this example, we consider a case where a user, say Alice, wants to access her data through a web service (usually running inside a web

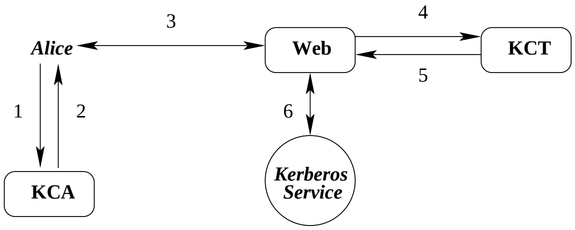

Figure 2: Accessing a Kerberos service using K-PKI.

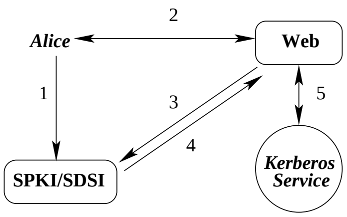

Figure 3: Accessing a Kerberos service using K-SPKI/SDSI.

server). We compare two approaches, one based on the K-PKI project [16], and the other based on our approach.

**A previous approach.** The K-PKI [16] project addresses the integration problem by providing a mechanism that translates Kerberos credentials to X.509 certificates, and vice versa. Figure 2 illustrates how a client, say Alice, accesses a Kerberos service through a web service. For the remainder of the example, it is assumed that Alice already has a TGT, and all communications are secure.

1. *Alice requests for an X.509 certificate:* Alice first generates a public/private key pair for herself. She then obtains an SGT for the *Kerberos Certification Authority*, or KCA, and authenticates with the KCA. As part of the authentication process, she sends the public key to the KCA to be signed.
2. *KCA generates an X.509 certificate:* The KCA, after authenticating Alice, signs Alice's public key and returns the signed certificate back to Alice.
3. *Alice authenticates with web service:* Alice uses the newly generated certificate to authenticate herself with the web service.
4. *Web service matches a request for a Kerberos ticket:* The web service authenticates itself with the *Kerberos Credential Translator*, or KCT, and provides KCT with evidence that it has properly authenticated with Alice.
5. *KCT generates a Kerberos ticket:* KCT, upon validating the identity and the request of the web service, generates a Kerberos ticket *using Alice's identity* for the web service and returns the ticket to the web service.
6. *Accessing the Kerberos service:* The web service, using the Kerberos ticket returned by KCT, can now access the Kerberos service on behalf of Alice.

The drawback of this approach is that it has to translate Kerberos credentials to PKI certificates, and vice versa. We achieve the same objective, but in a simpler way, using our K-SPKI/SDSI approach.

**K-SPKI/SDSI approach.** Using K-SPKI/SDSI, Alice can grant access permission to the web service directly, thereby avoiding the credential conversions that are required in the K-PKI approach. Figure 3 shows the setup of our system. In our system, Alice uses the following steps to access a Kerberos service through the web:

1. *Alice grants access to the web service:* Alice first obtains an SGT for the K-SPKI/SDSI server. Using the SGT, she issues an auth cert through the K-SPKI/SDSI server, granting rights to the web service, along with validity information.

$$K_{CS}\ \texttt{Alice}\,\Box \xrightarrow{T_V} K_{\texttt{CS}}\texttt{Web}\ \blacksquare$$

2. *Alice authenticates with the web service.*
3. *Web service requests for access token:* After authenticating Alice, the web service queries the K-SPKI/SDSI server for a token for accessing data on behalf of Alice.
4. *K-SPKI/SDSI server checks for permission:* The K-SPKI/SDSI server performs a distributed certificate-chain discovery to check the requested access permission; if the check is successful, the K-SPKI/SDSI server returns an access token to the web service.
5. *Accessing the Kerberos service:* The web service, using the token returned by the K-SPKI/SDSI server, can now access the Kerberos service.

This approach is also flexible to use. Because Alice is issuing auth certs directly, she may limit the access privileges granted to the web service according to her security objectives. For example, she may use a short validity period to limit the time window during which the web service can access the data she specifies; she may also restrict the web service's access rights by granting specific access privileges.

## 5 Implementation and Evaluation

We have built a prototype system to evaluate our approach. The implementation uses MIT's *Kerberos* distribution (version 1.3.1 [20]) and the *Distributed SPKI/SDSI* library that is based on a model checker for pushdown systems [22]. We evaluated our approach using two criteria: *ease of deployment* and *performance*. Because our implementation is still a prototype, and we have not deployed the system in a real-world environment, we evaluated the prototype in a simulated environment, using synthetic data. However, as the experimental results demonstrate, we believe that our approach does achieve the goal of reducing SPKI/SDSI's dependence on public-key infrastructure, and that it is easy to implement and deploy such a system. We summarize the results based on these two criteria:

- **Ease of deployment:** Three steps are required to deploy our system, assuming that Kerberos is already installed.
    1. *Install a public/private key pair:* In our approach, only one public/private key pair is needed for each Kerberos site. In comparison, PKI's systems require every user to have a key pair. In addition, sites need to exchange their public keys. However, we believe that this is a reasonable requirement because the exchange is done only once.
    2. *Install the K-SPKI/SDSI server:* Each Kerberos site must have its own logical K-SPKI/SDSI server. Because K-SPKI/SDSI server is implemented as a Kerberos service, this does not require any changes to Kerberos besides setting up the secret key between the KDC and the K-SPKI/SDSI server.
    3. *Update Kerberos clients:* Kerberos clients must be updated to take advantage of the K-SPKI/SDSI server. However, all clients need to do is to use a new library call to access the K-SPKI/SDSI server.
- **Performance:** The experimental results demonstrate that the performance of distributed authorization is highly dependent on how E-SPKI/SDSI certificates are distributed among the sites: the more distributed the certs are, the more sites are needed to resolve authorization queries, and the longer it takes to process an authorization query. In our study, distributed authorization performed well: in a test environment with about 1,500 certificates and eight Kerberos sites, it took about 1 second to process a complex authorization request, and took half as long to process a simple one. Because this is only a prototype implementation, there is still plenty room for optimizations that would improve the perfor-

mance.

### 5.1 Ease of deployment

The objective of this work is to make SPKI/SDSI, and potentially other trust-management systems, less reliant on PKI and hence easier to deploy in the real world. We achieve this goal by two means. First, we reduce SPKI/SDSI's reliance on PKI by relying on authentication provided by existing infrastructures, such as Kerberos, that are proven and in use. The approach tries to make SPKI/SDSI fit into existing systems seamlessly, instead of introducing drastic changes that would be hard to have accepted. Deploying our system in environments where Kerberos is installed only requires a few small changes.

Second, in terms of implementation, we tried to make sure that our approach does not introduce too many changes to Kerberos, because changes usually result in more complications for deployment. We achieved this goal by implementing the K-SPKI/SDSI server as an independent unit, instead of changing the KDC. Clients can simply interact with the K-SPKI/SDSI server through the standard request/reply model. As a result, our implementation requires no changes to the KDC, and only one minor modification to the Kerberos library. [7]

However, this approach also has some drawbacks. First, by using a separate server, clients must be modified to use the provided features—even though the change is very simple. The alternative is to provide these functionalities inside the KDC. When a Kerberos client requests an SGT for a service, the KDC automatically performs the necessary authorization query on behalf of the client and stores the authorization token as part of the SGT. This approach makes the authorization process transparent to the clients, but it does require changes to the KDC. This technique is also used by others for adding authorization support inside Kerberos [9, 7, 16, 3]. We are currently evaluating both approaches.

In addition to the changes above, when deploying our system, each site must install a public/private key pair. Furthermore, each site needs to send its public key to other sites with which it plans to collaborate. However, we believe that this is a reasonable requirement because setting up collaboration is an administrative task that only needs to be done once for each collaborator. For example, in Kerberos, cross-site authentication requires participating sites to exchange their secret keys in advance.

### 5.2 Performance

We also evaluated the performance of our system in a simulated distributed environment. We only considered the performance for distributed authorization because issuing certificates is an infrequent administrative task. The simulated test environment consists of eight Kerberos sites, as shown in Figure 4. Each node in the graph represents a Kerberos site; nodes with a symbol `R` represent a resource/service that Kerberos users can access. To illustrate what goes on, some of the certificates used in the experiments are shown next to each site. Because in a distributed environment every Kerberos site stores its own certificates, distributed authorization must involve two or more sites, depending on how the E-SPKI/SDSI certificates are distributed. For instance, in Figure 4 when `Manager` from the site `GOV` attempts to access the resource `R` from `NSF`, only these two sites are involved in distributed authorization, as denoted by the solid arrow. In contrast, when `Alice`, from `CS`, wants to access the same resource `R`, multiple sites (along the dashed arrows) must participate in the distributed authorization. Therefore, we expect the number of sites involved in distributed authorization to be an important factor in performance. For this reason, we tested distributed authorization using three different scenarios, shown by the three types of arrows in Figure 4.

[7] We changed the function `kuserok`, which, upon called, evaluates whether a Kerberos principal is allowed to login to a host. Our change provides an option for callers of this function to use the K-SPKI/SDSI server to check for authorization.

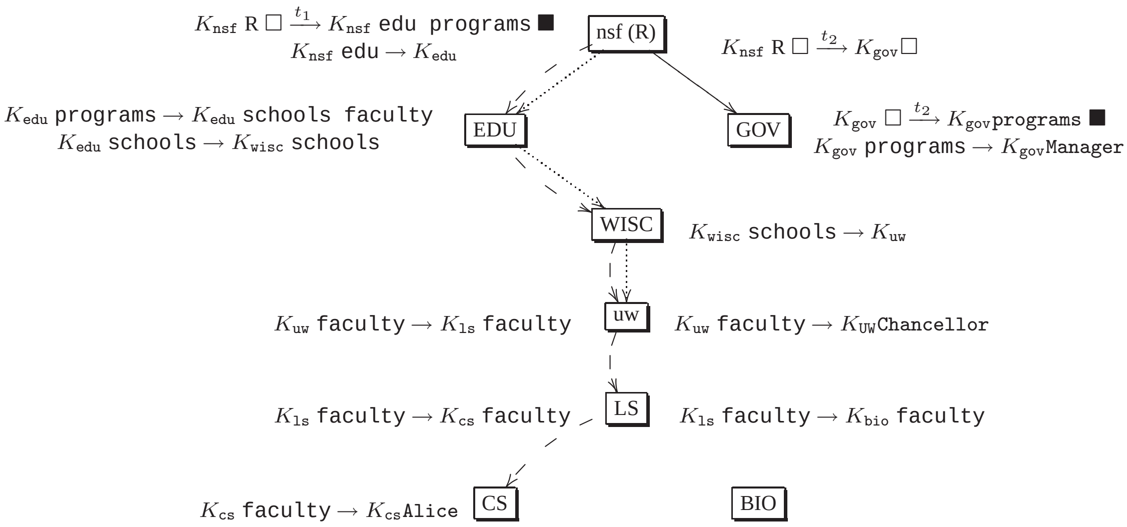
$K_{\texttt{nsf}}$ R □ $\xrightarrow{t_1}$ $K_{\texttt{nsf}}$ edu programs ■
$K_{\texttt{nsf}}$ edu → $K_{\texttt{edu}}$
nsf (R)
$K_{\texttt{nsf}}$ R □ $\xrightarrow{t_2}$ $K_{\texttt{gov}}$□
$K_{\texttt{edu}}$ programs → $K_{\texttt{edu}}$ schools faculty
$K_{\texttt{edu}}$ schools → $K_{\texttt{wisc}}$ schools
EDU
GOV
$K_{\texttt{gov}}$ □ $\xrightarrow{t_2}$ $K_{\texttt{gov}}$programs ■
$K_{\texttt{gov}}$ programs → $K_{\texttt{gov}}$Manager
WISC
$K_{\texttt{wisc}}$ schools → $K_{\texttt{uw}}$
$K_{\texttt{uw}}$ faculty → $K_{\texttt{ls}}$ faculty
uw
$K_{\texttt{uw}}$ faculty → $K_{\texttt{UW}}$Chancellor
$K_{\texttt{ls}}$ faculty → $K_{\texttt{cs}}$ faculty
LS
$K_{\texttt{ls}}$ faculty → $K_{\texttt{bio}}$ faculty
$K_{\texttt{cs}}$ faculty → $K_{\texttt{cs}}$Alice
CS
BIO

Figure 4: Test setup: `R` grants $t_1$: `(fundA apply)` to all NSF's `EDU` programs, and delegates $t_2$: `(fundB apply)` to all NSF's `GOV` programs. Each type of arrows represent one test scenario.

Table 1: Distributed Authorization Performance Results

| Scenario | # of sites | Request | Time (ms) |
|---|---|---|---|
| Manager@GOV | 2 | `(fundB apply)` | 581 |
| Chancellor@UW | 4 | `(fundA apply)` | 930 |
| Alice@CS | 6 | `(fundA apply)` | 1128 |

We populated the test environment with 1500 name certs and 30 auth certs, distributed over different sites. Each site runs on a separate machine on a local area network. All test machines have identical configurations: 800 MHz Pentium III with 256 MB RAM, running TAO Linux version 1.0.

Table 1 shows the results of the experiments. As expected, the number of sites involved in distributed authorization has direct impact on the performance of the system. In the most complex case (`Alice@CS`), where six Kerberos sites are involved, distributed authorization took almost twice as long as what it takes in the simplest case (`Manager@GOV`), where two sites are involved. However, as this is only a prototype, we expect to be able to improve the performance in the future by optimizing the code. Furthermore, our test setup is an extreme case where every Kerberos site has its own physical KDC. In practice, logical Kerberos sites can share one physical KDC, which would improve the real-world performance. For example, because `CS`, `BIO`, `LS` and `UW` are logical sites inside `UW`, it might be feasible and reasonable to have one physical KDC for all of them, and this would reduce the network overhead considerably.

## 6 Related Work

Leveraging the advantages of both Kerberos and Public-Key Infrastructure (PKI) has been explored before. *PKINIT* [24] is an IETF proposal that extends Kerberos by using public-key cryptography during the initial authentication between clients and the KDC. After a successful authentication, the KDC returns a standard

Kerberos TGT to the client, who can then use the TGT to access other Kerberos services. While PKINIT addresses how to use PKI within a Kerberos realm, *PKCROSS* [11] extends the idea of using PKI in Kerberos cross-realm authentication. KDCs in different realms exchange their public keys and use public-key cryptography for authentication and ticket exchange. *PKDA* [23] changes Kerberos by eliminating the KDC and the initial TGT exchange. With PKDA, a Kerberos client directly authenticates with the application server using public-key cryptography and receives a Service Granting Ticket (SGT) generated by the application server. Consequently, no TGT is needed in PKDA. However, Medvinsky et al. pointed out that PKDA can be implemented using PKINIT [18], albeit some changes are required. The aforementioned work differs from ours because their objective is to extend Kerberos to use public-key infrastructures for authentication purposes. Our work has a different goal, namely, *to use Kerberos to reduce the dependence of SPKI/SDSI on PKI*. Furthermore, their approaches require modifications to Kerberos infrastructure itself, while our approach does not.

*K-PKI* [5, 16] addresses the problem of accessing Kerberos services from PKI-based systems, such as web applications. K-PKI provides a special Kerberos server, KCA, which can generate short-term X.509 certificates for authenticated Kerberos clients. Later on, when a client tries to access Kerberos services through some web applications, she first authenticates with the web services using the generated certificate. The web services, in turn, can obtain necessary Kerberos credentials and access the Kerberos services on behalf of the client. While K-PKI provides a glue between Kerberos and PKI world, the complexity of the PKI systems is not reduced: all clients still need certificates. Our work, on the other hand, tries to reduce the reliance of trust-management systems on PKI. As a result, with our approach, clients no longer need to have public/private key pairs.

Another aspect of our work is to bring trust management, such as SPKI/SDSI, to Kerberos-based infrastructures. Although there has been some previous work on extending Kerberos's authentication framework with authorization services, that work generally assumes a centralized authority and does not address cross-realm authorization. Of these, Neuman's work on *restricted proxy* [19] is the closest to ours. Restricted proxy is a model for building various authorization services such as authorization servers, capabilities, and access control. However, SPKI/SDSI is a superset of restricted proxy, and it offers other features, such as distributed trust management. DCE's *Privilege Service (PS)* [9], ECMA's *SESAME* [7], and Microsoft's Kerberos extension [3] provide authorization capability through the use of an optional field (called *authorization data*) provided by Kerberos. For each authenticated Kerberos client, DCE's Privilege Service generates a ticket, called a *Privilege Attribute Certificate*, or *PAC*, which contains membership information for the corresponding principal. The client then presents her Kerberos ticket, together with the PAC, to Kerberos services that the client wants to access. The SESAME system from ECMA also defines and uses PAC to assert a principal's access rights. Microsoft's KDC extension stores authorization information, such as security identifiers and group membership information, inside the TGTs issued by the KDC. This authorization data is used by application servers to check users' access privileges. These works have the common drawback that, unlike SPKI/SDSI, they rely on a centralized authority for granting access privileges, and the authorization authority must know about every user. In contrast, our approach uses SPKI/SDSI, which does not require a central authority, and authorization decisions are made in a decentralized manner.

SPKI/SDSI [8], based on public-key-infrastructure, was designed to address the *centralized authority* issue of conventional PKI-based systems. SPKI/SDSI provides a novel framework for managing trust (in the form of certificates) using an entirely decentralized approach. In SPKI/SDSI, no central authority is needed because each principal can issue her own certificates. Much of the previous work on SPKI/SDSI focuses on theoretic aspects of SPKI/SDSI. Clarke et al. [6] proposed the original certificate-chain-discovery algorithm for answering authorization queries in SPKI/SDSI. Jha and Reps made an improvement to Clarke et al.'s algorithm by applying pushdown-system theory to the certificate-chain-discovery problem [13, 14]. Both algorithms require certificates to be centralized. Li et al. [17] presented a system, called $RT_0$, in which certificate-chain discovery can be done in a distributed manner. Despite this work, SPKI/SDSI has not been adopted in the real world, primarily due to the difficulty of key-management issues in PKI-based systems. Our work addresses this problem by reducing SPKI/SDSI's reliance on PKI, and making use of Kerberos, essentially unchanged. By relying on Kerberos, a system that is proven—and, more importantly, widely used—our approach can make SPKI/SDSI easier to be adopted in the real world.

## References

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

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

[3] John Brezak. Utilizing the Windows 2000 Authorization Data in Kerberos Tickets for Access Control to Resources, February 2002. *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnkerb/html/MSDN_PAC.asp* (circa May 2005).

[4] Didier Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106(1):61–86, 1992.

[5] CITI: Projects: Kerberos Leveraged PKI. *http://www.citi.umich.edu/projects/kerb_pki/* (circa May 2005).

[6] Dwaine Clarke, Jean-Emile Elien, Carl M. Ellison, Matt Fredette, Alexander Morcos, and Ronald L. Rivest. Certficate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(1/2):285–322, 2001.

[7] European Computer Manufacturers Association (ECMA). Secure European System for Applications in a Multi-vendor Environment (SESAME). *https://www.cosic.esat.kuleuven.ac.be/sesame/html/sesame_documents.html* (circa May 2005).

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

[9] The Open Group. DCE 1.1: Authentication and Security Services. *http://www.opengroup.org/onlinepubs/9668899/* (circa May 2005).

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

[11] Matthew Hur, Brian Tung, Tatyana Ryutov, Clifford Neuman, Ari Medvinsky, Gene Tsudik, and Bill Sommerfeld. Public Key Cryptography for Cross-Realm Authentication in Kerberos, November 2001. Internet-Draft, draft-ieft-cat-kerberos-pk-cross-08.txt.

[12] S. Jha and T. W. Reps. Model checking SPKI/SDSI. *Journal of Computer Security*, 12(3-4):317–353, 2004.

[13] Somesh Jha and Thomas Reps. Analysis of SPKI/SDSI certificates using model checking. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW)*, pages 129–146. IEEE Computer Society, June 2002.

[14] Somesh Jha and Thomas Reps. Model checking SPKI/SDSI. *Journal of Computer Security*, 12(3–4):317–353, 2004.

[15] T. Jim. SD3: A trust management system with certified evaluation. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2001.

[16] Olga Kornievskaia, Peter Honeyman, Bill Doster, and Kevin Coffman. Kerberized credential translation: A solution to web access control. In *10th USENIX Security Symposium*, pages 235–250, 2001.

[17] Ninghui Li, William H. Winsborough, and John C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, February 2003.

[18] Ari Medvinsky and Matthew Hur. Public key utilizing tickets for application servers (pktapp), January 1997. Internet-Draft.

[19] B. Clifford Neuman. Proxy-Based Authorization and Accounting for Distributed Systems. In *International Conference on Distributed Computing Systems*, pages 283–291, 1993.

[20] C. Neuman and T. Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications Magazine*, pages 33–38, September 1994.

[21] OpenAFS. *http://www.openafs.org* (circa May 2005).

[22] Stefan Schwoon, Somesh Jha, Thomas Reps, and Stuart Stubblebine. On generalized authorization problems. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 202–218. IEEE Computer Society, June 2003.

[23] M. Sirbu and J. Chuang. Distributed Authentication in Kerberos Using Public Key Cryptography, February 1997.

[24] Brian Tung, Clifford Neuman, Matthew Hur, Ari Medivinsky, Sasha Medvinsky, John Wray, and Jonathan Trostle. Public key cryptography for initial authentication in kerberos, 2004. Internet-Draft, draft-ieft-cat-kerberos-pk-init-17.txt.

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