Reducing the Dependence of SPKI/SDSI on PKI

Hao Wang[1⋆], Somesh Jha[1⋆], Thomas Reps[1⋆], Stefan Schwoon[2], and Stuart Stubblebine[3]

[1] University of Wisconsin, Madison, U.S.A., {hbwang, jha, reps}@cs.wisc.edu
[2] Universität Stuttgart, Germany, schwoosn@fmi.uni-stuttgart.de
[3] Stubblebine Research Labs, stuart@stubblebine.com

Abstract. Trust-management systems address the authorization problem in distributed systems. They offer several advantages over other approaches, such as support for delegation and making authorization decisions in a decentralized manner. Nonetheless, trust-management systems such as KeyNote and SPKI/SDSI have seen limited deployment in the real world. One reason for this is that both systems require a public-key infrastructure (PKI) for authentication, and PKI has proven difficult to deploy, because each user is required to manage his/her own private/public key pair. The key insight of our work is that issuance of certificates in trust-management systems, a task that usually requires public-key cryptography, can be achieved using secret-key cryptography as well. We demonstrate this concept by showing how SPKI/SDSI can be modified to use Kerberos, a secret-key based authentication system, to issue SPKI/SDSI certificates. The resulting trust-management system retains all the capabilities of SPKI/SDSI, but is much easier to use because a public key is only required for each SPKI/SDSI server, but no longer for every user. Moreover, because Kerberos is already well established, our approach makes SPKI/SDSI-based trust management systems easier to deploy in the real world.

1 Introduction

Authorization is a central problem in distributed environments where resources are shared among many users across different administrative domains. Trust-management systems [3] are designed to address the authorization problem in distributed environments; they answer the question "Is principal A allowed to perform operation O on a shared resource R?". Existing trust-management systems, such as KeyNote [2] and SPKI/SDSI[1] [9], rely heavily on public-key infrastructure (PKI). They use PKI to produce digitally-signed *certificates*, which authorize a principal to perform an operation on a shared resource.

However, PKI-based systems have proved difficult to deploy in practice because of several reasons [17]. Some issues (e.g., naming) have been addressed by trust-management systems, such as KeyNote and SPKI/SDSI. However, each user is still required to possess a public-private key pair, and it is cumbersome to securely transport and retrieve private keys. Complexity of PKI is another issue that makes PKI-based systems difficult to deploy. Implementing PKI-based

---

⋆ Supported by NSF under grants CCF-0524051 and CCR-9986308, and by ONR under grants N00014-01-1-{0796,0708}.

[1] Strictly speaking, SPKI/SDSI would not be considered to be a trust-management system according to the definition given by Blaze *et al.* [3]—if the processing of the certificates is not standardized (i.e., is application specific). In the context of this paper, we assume that certificate processing in SPKI/SDSI is standardized, and hence consider SPKI/SDSI to be a trust-management system.

solutions requires in-depth knowledge of PKI and much modification to existing systems.

Despite the issues mentioned above, trust-management systems are still desirable for authorization in distributed environments because they offer several advantages over traditional centralized authorization systems [2]. For example, because the trust-management system SPKI/SDSI has no conceptual requirement for a central authority and provides the ability to make authorization decisions in a truly distributed fashion [14], it is very scalable—an important requirement in distributed systems. SPKI/SDSI is also simple to use as it supports delegation, which simplifies access control, and provides locally defined name spaces, which allows each user to define his/her own security policies.

We introduce a technique to reduce the dependence of trust-management systems on PKI so that they become easier to deploy in the real-world. We observe that the main use of PKI in trust-management systems is to digitally sign each certificate with the private key of the principal who issues the certificate. The key behind our work is that *the signing process can be achieved using secret-key-based systems as well*. Although the notion of using secret-key cryptography in place of public-key cryptography as the building block of security operations has been studied previously [16, 8] and has been used in distributed military and banking systems, to the best of our knowledge, our work is the first to apply this technique in the context of trust-management systems, specifically SPKI/SDSI.

By utilizing existing secret-key-based systems, which are already widely deployed, we can reduce the dependence of trust-management systems on PKI because end users no longer need to have public-private key pairs. In our approach, each site in a distributed environment has a dedicated trust-management server, whose sole purpose is to issue digitally-signed certificates, and this server possesses a public-private key pair. Users at a site authenticate themselves to this server using a secret key, and the server issues digitally-signed certificates on their behalves. Thus, in our solution *just one server per site needs to have a public-private key pair*, as opposed to traditional trust-management systems, where *each principal must possess a public-private key pair*.

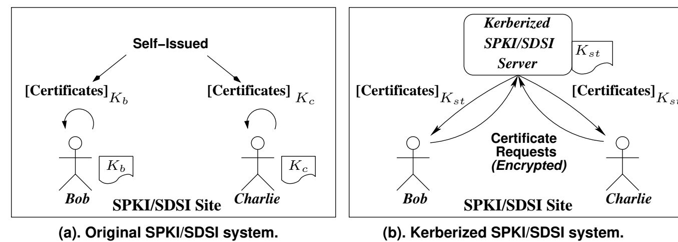

Fig. 1. Reducing SPKI/SDSI's dependence on PKI using Kerberos.

In this paper, we focus on the trust-management system SPKI/SDSI and show how to reduce its dependence on PKI by using Kerberos [19], a widely-deployed secret-key-based authentication system. In our approach, we allow authenticated Kerberos users to issue SPKI/SDSI certificates. The Kerberized

SPKI/SDSI server[2] (K-SPKI/SDSI) accepts certificate requests from authenticated Kerberos users, and generates corresponding SPKI/SDSI certificates on their behalves. In the original SPKI/SDSI system, shown in Figure 1(a), each principal has the ability to issue *name certificates* and *auth certificates*, signed using his/her public-private key. In contrast, with our solution, shown in Figure 1(b), each user no longer needs to have a public-private key pair. Instead, a site has a dedicated Kerberized SPKI/SDSI server—with its own public-private key—that is responsible for signing certificates. To issue a SPKI/SDSI certificate, a user first authenticates with the local Kerberos server and obtains a secure communication channel with the K-SPKI/SDSI server. The user can then issue the same certificates, in the form of *certificate requests*, but without the public-private key pair. The certificate requests are sent by the user, through the secure channel, to the K-SPKI/SDSI server, which creates and signs the certificates. The signed certificates can be either stored at the K-SPKI/SDSI server or sent back to the user, depending on the configuration of the system. In the case where the newly issued certificates are sent back to the user, the new system operates identically to the original SPKI/SDSI system because the certificates are stored locally, and authorization decisions can still be made locally. If the certificates are kept on the server, the server would act as a repository for the certificates issued by users in its domain. Such repositories could be used to organize certificate-chain discovery either centrally or in a distributed manner [14], leaving the burden of certificate management completely to the server.

Our technique offers several tangible benefits. Because end-users of the system authenticate themselves with a dedicated trust-management server using secret keys, it rids trust-management systems of the requirement that each principal must possess a public-private key pair. Furthermore, because we use secret keys to authenticate users to the dedicated server, and secret-key cryptography is widely deployed, we believe that the solution we present will make it easier to deploy PKI-based trust-management systems. In addition, only a small change is required at the end-user level to deploy our solution: each Kerberos application can now pass an optional parameter to the Kerberos library function `kuserok` to indicate that it wants to use our K-SPKI/SDSI server to perform an authorization check. Finally, because the dedicated trust-management server still uses a public-private key pair, our solution retains all the advantages of trust-management systems, such as delegation and distributed authorization.

The contributions of this paper are as follows:

- We show how to make SPKI/SDSI easier to deploy in the real world by reducing its dependence on PKI through leveraging Kerberos, a secret-key-based system that is already widely deployed.
- Our approach synthesizes the benefits of both secret-key-based authentication systems, such as Kerberos, and PKI-based trust-management systems, such as SPKI/SDSI. We utilize Kerberos' proven authentication framework while retaining SPKI/SDSI's elegant distributed authorization fea-

[2] Here, *Kerberize* means that we modify the SPKI/SDSI server to use Kerberos library.

tures, such as *delegation*, *authorization proofs*, *local name spaces*, and *distributed certificate-chain discovery*.
- We have created a prototype that implements the technique; the paper provides a preliminary report about our implementation and its performance.

Background on SPKI/SDSI is given in Section 2; readers with a knowledge of SPKI/SDSI may choose to skip this section. The method for combining SPKI/SDSI and Kerberos is described in Section 3. Section 4 discusses deployment and performance issues of our prototype. Section 5 discusses related work.

2 Background on SPKI/SDSI

SPKI/SDSI [9] is a novel public-key infrastructure designed to address the authorization problem in distributed systems. In SPKI/SDSI, a *principal* can be an individual, process, host, or any other entity. All *principals* are represented by their public keys, i.e., a principal *is* its public key. Let $\mathcal{K}$ denote 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 Σ. The set of identifiers is denoted by $\mathcal{A}$. Identifiers will be written in typewriter font, e.g., `A` and `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 `A`, where $K \in \mathcal{K}$ and $\mathtt{A} \in \mathcal{A}$. For example, K `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 `A`. An *extended name* is of the form $K\ \sigma$, where $K \in \mathcal{K}$ and σ is a sequence of identifiers of length greater than one. For example, K `UW CS 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 is a signed four-tuple $(K, \mathtt{A}, S, V)$. The issuer K is a public key and the certificate is signed by K. `A` is an identifier. The subject S is a term. Intuitively, S gives additional meaning for the local name K `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 (with or without delegation privileges) a specific authorization from an issuer to a subject. Specifically, an auth cert 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.

2.2 Certificates as Rewrite Rules

A *labeled rewrite rule* is a triple $L \xrightarrow{T} R$, where L and R are terms and T is an authorization specification. $\hat{T}$ is the authorization specification such that for all other authorization specifications t, $\hat{T} \cap t = t$, and $\hat{T} \cup t = \hat{T}$.[3] Sometimes we will write $\xrightarrow{\hat{T}}$ simply as $\longrightarrow$, i.e., a rewrite rule of the form $L \longrightarrow R$ has an implicit label of $\hat{T}$. We will treat certs as labeled rewrite rules:

- A name cert $(K, \mathtt{A}, S, V)$ will be written as a labeled rewrite rule $K\ \mathtt{A} \longrightarrow S$.
- 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 $K \square \xrightarrow{T} S \blacksquare$.

Note that in authorization problems, we only consider valid certificates, so, as a pre-processing step, we first check the validity specification V for each certificate in use. For the rest of the paper, we assume that only valid certificates are considered for authorization proofs.

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 at 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 $L \xrightarrow{T} R$ that corresponds to an auth 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,

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

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:

$$c_1: \quad K_A\ \mathtt{friends} \xrightarrow{T} K_A\ \mathtt{Bob}\ \mathtt{myFriends}$$
$$c_2: \quad K_A\ \mathtt{Bob} \xrightarrow{T'} K_B$$

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

A *certificate chain* ch is a sequence of certificates $[c_1, c_2, \cdots, c_k]$. The label of a certificate chain $ch = [c_1, \cdots, c_k]$, denoted by $L(ch)$, is the label obtained from $c_k \circ c_{k-1} \cdots \circ c_1$.

3 Kerberizing SPKI/SDSI

In this section, we explain how we can reduce the dependence of SPKI/SDSI on PKI by utilizing a secret-key-based authentication system, namely *Kerberos*. We first introduce an example that will be used throughout this section. Next, we use

[3] The issue of intersection and union of authorization specifications is discussed in [9, 11].

[4] In general, the composition operator $\circ$ is not associative. 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.

this example to illustrate how the original SPKI/SDSI system works. Finally, in Section 3.2, we describe how the reliance of SPKI/SDSI on PKI can be reduced by using Kerberos. We assume that the reader is familiar with Kerberos (for a detailed description of Kerberos see [19]).

Example. Suppose that there are two sites, `Bio` and `CS`, which correspond to the biology and the computer science departments, respectively. Two professors, *Alice* from `CS` and *Bob* from `Bio`, are collaborating on a project. *Bob* wants to delegate to *Alice* full access rights to a shared resource R. In addition, *Alice* plans to delegate access rights to resource R to her students, who are also involved in the project, without allowing them to delegate these rights further.

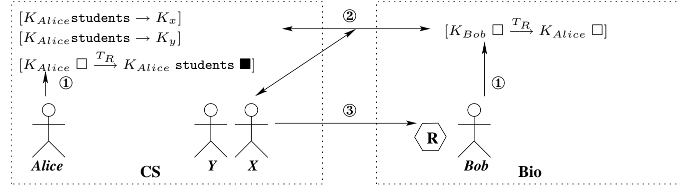

Fig. 2. Distributed authorization using SPKI/SDSI.

3.1 Authorization in SPKI/SDSI

In this section, we describe how SPKI/SDSI authorization works in a distributed environment, using the example given above. There are three components to a SPKI/SDSI authorization scenario, denoted by the circled numbers in Figure 2.

Certificate issuance (Figure 2 $\stackrel{①}{\rightarrow}$). First, each user issues auth and name certs. In our example, *Bob* delegates access rights T_R to resource R to *Alice* by issuing the following auth cert, signed with his private key:

$$\boxed{K_{Bob} \,\square \xrightarrow{T_R} K_{Alice} \,\square}$$

At `CS`, *Alice* grants two of her students, X and Y, access to R by issuing the following two name certs and one auth cert, all signed with *Alice*'s private key:

$$\boxed{\begin{array}{l} K_{Alice}\ \texttt{students} \longrightarrow K_X \\ K_{Alice}\ \texttt{students} \longrightarrow K_Y \\ K_{Alice} \,\square \xrightarrow{T_R} K_{Alice}\ \texttt{students}\ \blacksquare \end{array}}$$

The two name certs state that X and Y are students of *Alice*; the auth cert states that all of her students (i.e., X and Y) can access resource R with authorization specification T_R, but they cannot delegate the access right.

Now assume that student X wants to access R at site *Bio* according to authorization specification T_R. He needs to perform the following two steps:

Certificate-chain discovery (Figure 2 $\stackrel{②}{\rightarrow}$). To request access to resource R, a user U first performs certificate-chain discovery to obtain a proof that he can access resource R. This can be achieved by executing a distributed certificate-chain-discovery algorithm [14], and, if the algorithm finds that U is authorized, it returns a proof in the form of a finite set of certificate chains $\{ch_1, \cdots, ch_m\}$. In our example, student X initiates the distributed certificate-chain discovery,

which will involve both *Alice* and *Bob*. The distributed certificate-chain discovery returns the singleton set of chains $\{ch_1\}$, where $ch_1 = [c_1, c_2, c_3]$ and c_i are the following certificates:

$$c_3 = K_{Bob} \,\square \xrightarrow{T_R} K_{Alice} \,\square$$
$$c_2 = K_{Alice} \,\square \xrightarrow{T_R} K_{Alice} \texttt{ students} \,\blacksquare$$
$$c_1 = K_{Alice} \texttt{ students} \longrightarrow K_X$$

Requesting a resource (Figure 2 $\xrightarrow{③}$*).* After user U obtains a set of certificate chains $SCH = \{ch_1, \cdots, ch_m\}$ from the previous step, he presents SCH to the owner of the resource R to which T_R refers. The owner authorizes K_U iff $T_R \subseteq \bigcup_{i=1}^{m} L(ch_i)$ (this step is usually called *compliance checking*).

In our example, after making the certificate-chain-discovery request, "Does $K_{Bob} \,\square$ resolve to $K_X\square$ or $K_X\blacksquare$ with authorization specification T_R?", student X presents $\{ch_1\}$ to K_{Bob}. K_{Bob} checks that $T_R \subseteq L(ch_1)$, which is true, and hence grants X access to resource R.

3.2 Authorization in Kerberized SPKI/SDSI

Notice that, to use SPKI/SDSI, *every user* needs to have a public-private key pair. In this section, we describe how to reduce SPKI/SDSI's dependence on PKI by using the distributed authentication system *Kerberos* in a SPKI/SDSI implementation. The key insight behind our work is that *the certificate issuance process in SPKI/SDSI can also be achieved using secret-key-based systems, such as Kerberos*. In SPKI/SDSI, each certificate is signed by its issuer using the private key, and the signature serves as the proof for the authenticity of the certificate. In a secret-key-based system such as Kerberos, the authentication process also produces the evidence for who the user is, and this evidence can be employed by the user to issue certificates. In Kerberos, an authenticated user obtains a token called *Ticket Granting Ticket (TGT)*, which contains digital evidence about the user. This token can be used to obtain a secure communication channel with various Kerberos services. Therefore, in our approach, we use a *Kerberized* SPKI/SDSI server for each site so that an authenticated Kerberos user can securely issue certificate requests through the Kerberized SPKI/SDSI server. In essence, our approach relaxes SPKI/SDSI's binding requirement where each user is identified by its public key. Instead, each principal will be a Kerberos principal in a Kerberos realm. Consequently, with our approach, a SPKI/SDSI user no longer needs to have a public-private key pair, and we only require one public-private key pair *per site*—namely, for the SPKI/SDSI server. Our new system is called *K-SPKI/SDSI*, short for *Kerberized SPKI/SDSI*.

In our system, each SPKI/SDSI site runs a Kerberized SPKI/SDSI server (K-SPKI/SDSI), which shares a public-private key pair with the Kerberos *Key Distribution Center (KDC)* at its site. The public-private key of site *st* is denoted by K_{st}. We now describe the three components of our authorization scenario in this new setting. Figure 3 illustrates the high-level idea behind our approach.

Certificate issuance (Figure 3 $\xrightarrow{①}$*).* 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

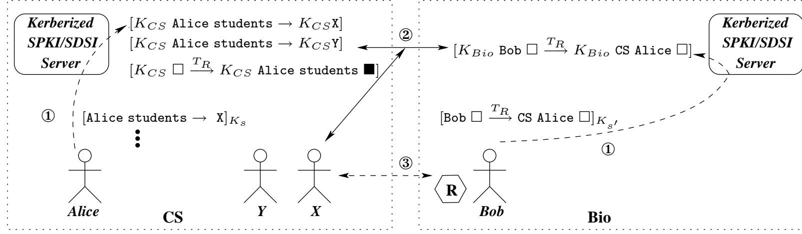

Fig. 3. Reducing SPKI/SDSI's dependence on PKI using Kerberos. Dashed lines represent secure Kerberos communication channels.

KDC. Using the TGT, the user requests a Service Granting Ticket (SGT) for accessing the Kerberized SPKI/SDSI (K-SPKI/SDSI) server. Throughout the rest of the section, we assume that the user has obtained an SGT for the K-SPKI/SDSI server at its site. Using the SGT, the user issues requests for generating SPKI/SDSI name certs or auth certs. The session key K_s provided in the SGT is used to protect both the integrity and confidentiality of the requests sent over the communication channel.

To issue an auth cert, a user at site *st* sends a cert request $E_{K_s}[U, S, D, T, V]$, encrypted with the session key K_s from the SGT, to the K-SPKI/SDSI server. Here U is the name of the user, S is the subject, D is the delegation bit, T is the authorization specification, and V is validity information. Upon receiving this encrypted auth-cert request, the K-SPKI/SDSI server ascertains its validity, and if the auth cert is valid, it creates a new K-SPKI/SDSI auth cert of the form $[K_{st}\ U, K_{st}\ S, D, T, V]$, signs it with its private key. The newly issued certificates can be stored in the K-SPKI/SDSI server so that authorization can be done more efficiently. However, to fully emulate SPKI/SDSI, the certs could also be sent back to the users who have requested them. In this scenario, authorization would be carried out exactly as in the original SPKI/SDSI. Notice that in the new auth cert the public key K_{st} of site *st* is added before both U and S. In our example, *Bob* sends the following auth-cert request, encrypted with the appropriate session key (obtained from his SGT), to the K-SPKI/SDSI server S_{Bio}:

$$\texttt{Bob}\ \square \xrightarrow{T_R} \texttt{CS Alice}\ \square$$

The auth cert states that *Bob* delegates full access rights to resource R to *Alice* from CS. The K-SPKI/SDSI server S_{Bio} verifies the encrypted auth certs shown above, then creates and signs the following K-SPKI/SDSI auth cert:

$$K_{\texttt{Bio}}\ \texttt{Bob}\ \square \xrightarrow{T_R} K_{\texttt{Bio}}\ \texttt{CS Alice}\ \square$$

To issue a name cert, a user at site *st* sends an encrypted name-cert request $E_{K_s}[U, \texttt{A}, S, V]$ to the K-SPKI/SDSI server. Here U, S, and V are the same as in an auth cert, while $\texttt{A}$ is an identifier. The validation step is exactly the same as that for issuing auth certs. After a request is validated, the K-SPKI/SDSI server creates a new name cert of the form $[K_{st}\ U, \texttt{A}, K_{st}\ S, V]$, signs it with its private key. Similar to auth certs, the name certs can be either stored in the K-SPKI/SDSI server or sent back to the users who have requested them. As before, we will write the name cert as $U\ \texttt{A} \longrightarrow S$. In our example, *Alice* sends

two name certs and one auth cert (Figure 3.2 (a)), encrypted with the session key K_s, to the K-SPKI/SDSI server at her site. The K-SPKI/SDSI server verifies the encrypted name certs, creates the corresponding K-SPKI/SDSI name certs, and signs them (Figure 3.2 (b)). Notice that the left-hand sides of K-SPKI/SDSI certificates have three symbols: the left-hand side of an extended auth cert is of the form K_α `U` $\square$ or K_α `U` $\blacksquare$, where K_α is the public key of site α and U is a user; the left-hand side of an extended name cert is of the form K_α `U A`, where both `U` and `A` are identifiers. In SPKI/SDSI the left-hand sides of auth and name certs have just two symbols. However, the translation from user certificate requests to the actual certificate can be done automatically because this is just a special case of left-prefix rewriting, and the primitives generalize to arbitrary left-prefix rewriting systems [5], which covers the case of K-SPKI/SDSI certs with three left-hand-side symbols.

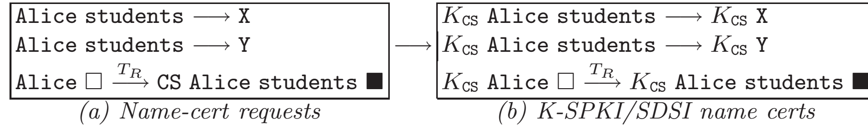

Fig. 4. Issuing SPKI/SDSI certificates using K-SPKI/SDSI.

Besides user-issued certificates, each SPKI/SDSI site also needs to exchange its public key with other SPKI/SDSI sites, represented as name certificates. For example, the site `Bio` would issue the following name certificate:

$$K_{\texttt{Bio}}\ \texttt{CS} \longrightarrow K_{\texttt{CS}}$$

Certificate Chain Discovery (Figure 3 $\xrightarrow{②}$*).* Suppose that user U at site st_1 wishes to access resource R at site st_2 with access rights given by T. U first initiates a certificate-chain discovery, e.g., using the algorithms from [20] or, in the case where certificates are stored at the K-SPKI/SDSI servers, using the distributed algorithm from [14]. If the search is successful and returns a set of certificate chains SCH, U needs its site st_1 to prepare a proof of authorization that U can present to the owner of R. This is because, while SCH proves that user U at site st_1 has access to R, only st_1 can assure the owner of R (who possibly resides at a different site st_2) that the requesting user is indeed U. Thus, U sends SCH to st_1, and st_1 sends back the following Kerberos tokens:

$$\begin{aligned} Token_U &= E_{K_s}(K_1)\, Ticket_U \\ Ticket_U &= E_{K_{st_2}}(K_2)\ E_{K_2}[st_1, (R, st_2, U, st_1, T, SCH, K_1, TS_1, Lifetime_1)_{K_{st_1}}] \end{aligned}$$

where K_s is the session key for U and st_1, K_1 and K_2 are fresh secret keys generated by st_1, and $(\cdot)_{K_{st_1}}$ denotes data signed by st_1. Intuitively, $Token_U$ makes the key K_1 known to U, and $Ticket_U$ says that SCH is a proof of authorization for access to R at site st_2 (with access type T) by user U at site st_1. By signing the message, site st_1 confirms that anybody in possession of key K_1 is indeed U. Notice that R, st_2, U, and st_1 are implicitly contained in SCH and can be omitted in practice. In our example, assume that student X receives a token with the set of certificate chains SCH $= \{ch_1\}$, where ch_1 is the certificate chain $[c_1, c_2, c_3, c_4]$: Notice that $(c_4 \circ c_3 \circ c_2 \circ c_1)(K_{Bio}\texttt{Bob}\,\square) \in \{K_{CS}\ \texttt{X}\,\square, K_{CS}\ \texttt{X}\,\blacksquare\}$ and $T_R \subseteq L([c_1, c_2, c_3, c_4])$.

$c_1 = K_{Bio}$ Bob $\square \xrightarrow{T_R} K_{Bio}$ CS Alice $\square$ $\quad c_3 = K_{CS}$ Alice$\square \longrightarrow K_{CS}$ Alice students $\blacksquare$
$c_2 = K_{Bio}$ CS $\longrightarrow K_{CS}$ $\quad c_4 = K_{CS}$ Alice students $\longrightarrow K_{CS}$ X

Requesting a resource (Figure 3 $\overset{③}{\rightarrow}$). Upon receiving $Token_U$, user U decrypts $E_{K_s}(K_1)$ and retrieves the session key K_1. He then constructs the following Kerberos authenticator:[5]

$$Authenticator_U \; = \; E_{K_1}[\, U, st_1, TS_2, Lifetime_2 \,]$$

User U sends the message $[Ticket_U \;\; Authenticator_U]$ to the owner of resource R (at site st_2), who requests its local K-SPKI/SDSI server st_2 to verify the message. The server performs the following steps:

- Decrypts the message $E_{K_{st_2}}(K_2)$ with its private key and retrieves K_2.
- Decrypts the part of $Ticket_U$ encrypted with K_2.
- Obtains the public key K_{st_1} and verifies the signature of st_1.
- Ascertains freshness and validity of the token using the time-stamp TS_1 and the lifetime $Lifetime_1$; checks that SCH indeed proves the desired access.
- Similarly, the K-SPKI/SDSI server ascertains the validity of the authenticator, thus making sure that the sender is indeed user U at st_1. 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 R indicating that U should be granted access, and that communication between R and U should be protected using key K_1.

3.3 Analysis

Correctness of the Protocol. The key idea behind our work is to rely on Kerberos to provide a secure channel for users to submit SPKI/SDSI name certs and auth certs, which are signed and stored at each site. In contrast, in the original SPKI/SDSI system, each user can issue and sign her own certs. We note that there is no conceptual difference between these two approaches; only the underlying security mechanisms used are different (one uses secret-key cryptography and the other uses public-key cryptography). For example, in the original SPKI/SDSI approach, a user U issues a name cert this way: $(K, \mathtt{A}, S, V)_{K'}$. Here the subscript K' denotes that the name cert is signed by the user using the private key K'. In comparison, the corresponding step in our approach is implemented by issuing the certificate request $E_K[U, \mathtt{A}, S, V]$, where E_K denotes that the name-cert request is encrypted by the session key shared between the user and the K-SPKI/SDSI server. The request is first validated, then translated by the K-SPKI/SDSI server into actual certificates, signed with K-SPKI/SDSI server's private key. Thus, in both cases, the possession of a secret (K' in SPKI/SDSI, and K in Kerberos) provides the digital links between the certs issued and the user who has issued them.

[5] The actual content of the authenticator is irrelevant in our example.

Trade Offs. Although previous work has shown that secret keys can be used in place of public keys to implement the same security objectives, such as building a secure broadcast-communication channel [8, 16], there are pros and cons with each approach. A secret-key-based system is simpler to set up and use. However, secret-key-based systems often require both communication parties to be online to function properly. For example, sending a message between two Kerberos users usually requires both the sender and receiver to be active at the same time so that they can exchange a secret encryption key.[6] On the other hand, public-key-based systems can operate in offline mode. For example, to send a message using PKI, the sender can simply encrypt the message using the recipient's public key, without contacting the recipient first. However, key management is a major issue that has hindered wider acceptance of PKI-based systems because it is much more cumbersome to maintain public-private key pairs.

Our approach eliminates the public-private key pairs for individual users. Instead, each K-SPKI/SDSI server has a dedicated public-private key pair that is used for signing SPKI/SDSI certificates, whereas users use secret keys in their communication with the server (e.g., for requesting certificates). We chose this hybrid approach for the following reasons. First, the use of PKI in the authorization mechanism of SPKI/SDSI lends itself to offline checking of certificate chains. Indeed, when a user requesting access to a resource presents a set of certificate chains to the owner of that resource, the architecture of SPKI/SDSI ensures that the owner can check the validity of these chains without contacting the owners of the keys involved in the chains (in fact, without even verifying their identities). While it is worth noting that by adopting the ideas of Lampson *et al.* [16] or Davis and Swick [8] it is possible to emulate SPKI/SDSI using secret keys only, such a scheme would not allow offline checking of certificates. Secondly, the communication between users and servers usually happens online, which motivates the use of secret keys in this context. Finally, if the certificates issued by users are stored in the SPKI/SDSI servers, our approach can be very well combined with the distributed certificate-chain algorithm presented in [14].

Threat Analysis. Our message exchange for requesting a resource is very similar to the exchange of messages between the client and KDC in Kerberos. In essence, the ticket $Ticket_U$ states that "anyone who uses K_1 is U". Since in $Token_U$ 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), only U could have known K_1 (assuming that authentication in Kerberos is correct). It is possible for an adversary to replay the message $[Ticket_U\ Authenticator_U]$ to the resource R and masquerade as U. However, this attack fails if R and U communicate using K_1, which is unknown to the attacker.

3.4 Extension to Other PKI-based Trust-Management Systems

Different trust-management systems have different logics to express security policies. Most of the components (auth and name certs in SPKI/SDSI) of these security policies are signed by principals using their private keys. Recall that our protocol essentially allows a server to sign statements on behalf of an authenti-

[6] Unless the two sides have previously agreed upon a shared secret key.

```
KeyNote-Version: 2
Local-Constants: Alice="DSA:4401ff92"
Authorizer: "RSA:abc123"
Licensees: Alice
Conditions: (app_domain == "RFC822-EMAIL") && (address ~= ".*@labs\\.com\$");
Signature: "RSA-SHA1:213354f9"
```

Fig. 5. An example of a KeyNote credential. Line 2 represents *Alice*'s public key.

```
KeyNote-Version: 2
Local-Constants: Alice="Kerberos:alice@LABS.COM"
Authorizer: "RSA:abc123"
Licensees: Alice
Conditions: (app_domain == "RFC822-EMAIL") && (address ~= ".*@labs\\.com\$");
Signature: "RSA-SHA1:213354f9"
```

Fig. 6. An example of the same KeyNote credential, but without requiring *Alice* to have a public key.

cated user. Although we have explained our protocol for SPKI/SDSI, it is clear that it can be used for other trust-management systems. We now demonstrate how our protocol can be extended for the trust-management system KeyNote [2]. Figure 5 shows an example of a KeyNote credential that grants some rights (`RFC822-EMAIL`) to *Alice*, who has the public key `DSA:4401ff92` (Line 2). We can achieve the same goal using our technique, as shown in Figure 6. In our approach, the credential states that if *Alice* is an authenticated Kerberos user with the Kerberos identity `alice@LABS.COM`, then she can have the rights specified in the credential. It must be noted that the two credentials, although they appear similar, have very different operational semantics. In the original KeyNote example, *Alice* can further delegate the rights she has received by issuing new credentials directly—without any compliance checking. However, in the Kerberized scenario, because *Alice* no longer has a public-private key pair, she can only delegate her rights by first authenticating herself through Kerberos, and then issue a delegation request through a dedicated Kerberized KeyNote server. In both cases, the *Authorizer* (Line 3) represents the public-private key for the Kerberized KeyNote server.

4 Implementation and Evaluation

We have built a prototype system to evaluate our approach. The implementation uses MIT's *Kerberos* distribution (version 1.3.1 [19]) and the *Distributed SPKI/SDSI* library, which is based on a model checker for weighted pushdown systems [20]. The test environment contains 1500 name certs and 30 auth certs, distributed over different sites. Each site runs on a dedicated 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.

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. 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 addition, sites need to exchange their public keys. However, we believe that this is a reasonable requirement because the exchange is done only once. Alternatively, public keys could be obtained on demand using existing solutions for public-key exchange.
2. *Install the K-SPKI/SDSI server:* Each Kerberos site must have its own K-SPKI/SDSI server. Because each 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. *Enhance Kerberos clients:* Kerberos clients that want to take advantage of our distributed authorization features can be updated easily by using a new library call to access the K-SPKI/SDSI server.

Performance: We have tested our implementation in a model where all certificates are stored at the K-SPKI/SDSI servers, which then uses the distributed algorithm from [14] for certificate-chain discovery (for results, see Section 4.2). In these experiments, the performance of distributed authorization is highly dependent on how K-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 of opportunity for optimizations that would improve the performance. Notice, however, that this issue is slightly orthogonal to the issue of integrating SPKI/SDSI with Kerberos, since certificate-chain discovery could still be done locally.

4.1 Ease of Deployment

The objective of this work is to make SPKI/SDSI, and potentially other PKI-based trust-management systems, less reliant on PKI and easier to deploy in the real world. We achieve this by two means. SPKI/SDSI's reliance on PKI is reduced by using the 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 substantial changes that would present an impediment to adoption. Deploying our system in environments where Kerberos is installed only requires a few small changes.

Second, in terms of implementation, we tried to minimize the changes to Kerberos, because such changes usually result in additional complications for deployment. We achieved this goal by implementing the K-SPKI/SDSI server as an independent unit, instead of changing the KDC. As a result, our implemen-

tation requires no changes to the KDC, and only one minor modification to the Kerberos library.[7]

Our approach also has some drawbacks. First, by using a separate server, clients must be modified to use the provided features—although 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 [4, 10, 22, 15]. 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 a collaboration is an administrative task that only needs to be done once for each collaborating site.

4.2 Performance

We also evaluated the performance of our system in a simulated distributed environment using the algorithm from [14]. We only considered the performance for distributed authorization because issuing certificates is an infrequent administrative task. The simulated test environment consisted of eight Kerberos sites, as shown in Figure 7. Each node in the graph represents a Kerberos site; nodes with a symbol `R` represent a 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, resolving an authorization request may involve multiple sites, depending on how the K-SPKI/SDSI certificates are distributed. For instance, in Figure 7 when `Manager` from the site `GOV` attempts to access resource `R` from `NSF`, only these two sites are involved in distributed authorization, as indicated 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, indicated by the three types of arrows in Figure 7.

Table 1 shows the results of the experiments. As expected, the number of sites involved in distributed authorization has a direct impact on the performance of the system. In the most complex case (`Alice@CS`), where six Kerberos sites were involved, resolving an authorization request took almost twice as long as the time required in the simplest case (`Manager@GOV`), where only two sites were involved. However, as this is only a prototype, we expect to improve the performance in the future by optimizing the code. Furthermore, our test setup is an extreme

[7] We changed the function `kuserok`, which, when 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.

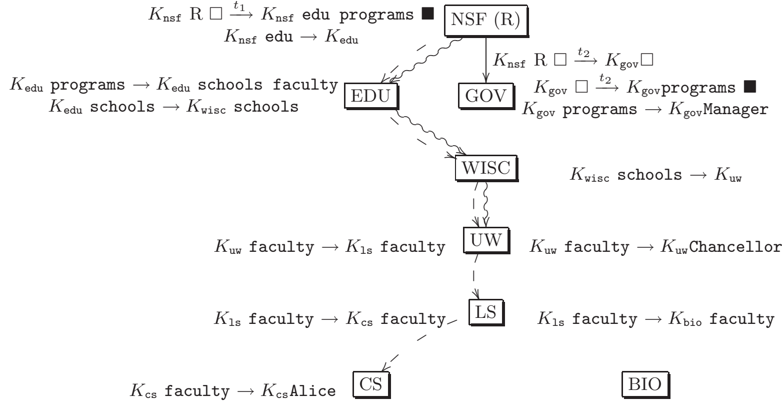

Fig. 7. Test setup with 1500 name certs and 30 auth certs (only a few are shown due to space constraints).

case where every Kerberos site has its own physical KDC. In practice, different logical Kerberos sites could share a single physical KDC, which would improve the performance by reducing the communication overhead.

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

5 Related Work

The key idea behind our approach is that we can use secret-key cryptography to implement the same SPKI/SDSI operations (e.g., issuing certificates) with the same level of security as with public-key cryptography. The notion of using secret keys in place of public-private key pairs as the building block of security operations was first proposed by Lampson *et al.* [16], who showed that, by using a *relay* (an agent that everyone trusts, e.g. the Kerberos KDC), one can build public-key-style secure communication channels. This idea has been extended by Davis and Swick to build other public-key-style security protocols using secret keys [8]. Our work also uses this idea, but applies it in the context of PKI-based trust-management systems, specifically SPKI/SDSI.

Leveraging the advantages of both Kerberos and Public-Key Infrastructure (PKI) has been explored before. *PKINIT* [23], *PKCROSS* [12], and *PKDA* [21] all extend Kerberos by using public-key cryptography for authentication purposes. Our work has a different goal: it is targeted toward authorization rather than authentication; in particular, the goal is *to use Kerberos to reduce the dependence of SPKI/SDSI on PKI.* Furthermore, the approaches of [23, 12, 21] require modifications to the Kerberos infrastructure itself, while our approach does not.

K-PKI [6, 15] addresses the problem of accessing Kerberos services from PKI-based systems, such as web applications. K-PKI provides a special Kerberos server, KCA, that 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, (s)he 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 the PKI world, the complexity of the PKI systems is not reduced: all clients are required to manage public-private key pairs. Our work, on the other hand, tries to reduce the reliance of trust-management systems on PKI so that individual users 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 previous work on extending Kerberos' 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* [18] 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 offers other features, such as distributed trust management. DCE's *Privilege Service (PS)* [22], ECMA's *SESAME* [10], and Microsoft's Kerberos extension [4] provide authorization capability through the use of an optional field (called *authorization data*) provided by Kerberos. For each authenticated Kerberos principal, authorization information (such as group membership, security identifiers) about the principal is stored in the field. This authorization data is used by application servers to check users' access privileges. These systems have the common drawback that, unlike SPKI/SDSI, they rely on a centralized authority for granting access privileges. In contrast, our approach uses SPKI/SDSI, which does not require a central authority, and authorization decisions can be made in a truly decentralized manner [14].

SPKI/SDSI [9], 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 a 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 theoretical aspects of SPKI/SDSI [1, 7, 13, 20]. Despite such 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—by making use of Kerberos, essentially unchanged. By relying on Kerberos, a well-accepted and widely used system, our approach should make it possible for SPKI/SDSI to be adopted in the real world more easily.

References

1. L. Bauer, S. Garriss, and M. K. Reiter. Distributed proving in access-control systems. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*,

pages 81–95, May 2005.
2. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote trust-management system version 2. RFC 2704, Sept. 1999.
3. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The role of trust management in distributed systems security. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, pages 185–210, 1999.
4. J. Brezak. Utilizing the Windows 2000 authorization data in Kerberos tickets for access control to resources. http://msdn.microsoft.com/library/default.asp?myurl=/library/enus/dnkerb/html/MSDN_PAC.asp.
5. D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106(1):61–86, 1992.
6. CITI: Projects. Kerberos leveraged PKI. http://www.citi.umich.edu/projects/kerb_pki/.
7. D. Clarke, J.-E. Elien, C. M. Ellison, M. Fredette, A. Morcos, and R. L. Rivest. Certficate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(1/2):285–322, 2001.
8. D. Davis and R. Swick. Network security via private-key certificates. In *Proceedings of the 3rd USENIX Security Symposium*, September 1992.
9. C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylönen. *RFC 2693: SPKI Certificate Theory*. The Internet Society, September 1999.
10. 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.
11. J. Howell and D. Kotz. A formal semantics for SPKI. Technical Report 2000-363, Department of Computer Science, Dartmouth College, Hanover, NH, Mar. 2000.
12. M. Hur, B. Tung, T. Ryutov, C. Neuman, A. Medvinsky, G. Tsudik, and B. Sommerfeld. Public key cryptography for cross-realm authentication in Kerberos, Nov. 2001. Internet-Draft, draft-ieft-cat-kerberos-pk-cross-08.txt.
13. S. Jha and T. Reps. Model checking SPKI/SDSI. *Journal of Computer Security*, 12(3–4):317–353, 2004.
14. S. Jha, S. Schwoon, H. Wang, and T. Reps. Weighted pushdown systems and trust-management systems. In *TACAS*, 2006.
15. O. Kornievskaia, P. Honeyman, B. Doster, and K. Coffman. Kerberized credential translation: A solution to web access control. In *10th USENIX Security Symposium*, pages 235–250, 2001.
16. B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, 1992.
17. J. Linn and M. Branchaud. An examination of assorted PKI issues and proposed alternatives. In *Proceedings of the 3rd Annual PKI R&D Workshop*, April 2004.
18. B. C. Neuman. Proxy-based authorization and accounting for distributed systems. In *ICDCS*, pages 283–291, 1993.
19. B. C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33–38, September 1994.
20. S. Schwoon, S. Jha, T. Reps, and S. Stubblebine. On generalized authorization problems. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 202–218. IEEE Computer Society, June 2003.
21. M. Sirbu and J. Chuang. Distributed authentication in Kerberos using public key cryptography, Feb. 1997.
22. The Open Group. DCE 1.1: Authentication and security services. http://www.opengroup.org/onlinepubs/9668899/.
23. B. Tung, C. Neuman, M. Hur, A. Medivinsky, S. Medvinsky, J. Wray, and J. Trostle. Public key cryptography for initial authentication in Kerberos, 2004. Internet-Draft, draft-ieft-cat-kerberos-pk-init-17.txt.