Information Technology Journal 6 (8): 1208-1216, 2007
ISSN 1812-5638
© 2007 Asian Network for Scientific Information

A Comparative Study of SPKI/SDSI and K-SPKI/SDSI Systems

[1]V. Vasudevan, [1]N. Sivaraman and [1]S. Senthil Kumar,
[1]R. Muthuraj, [2]J. Indumathi and [2]G.V. Uma
[1]Department of Information Technology, Madras Institute of Technology,
[2]Department of Computer Science and Engineering, College of Engineering,
Anna University, Chennai 600 025, Tamilnadu, India

Abstract: SPKI/SDSI is a popular trust-management system which defines a formal language for expressing authorization and access-control policies and relies on an algorithm to determine when a specific request can be granted. It also provides support for delegation of authority. K-SPKI/SDSI is a trust management system in which SPKI/SDSI is used in conjunction with Kerberos. This study focuses on differences between the two systems. SPKI/SDSI has seen limited deployment because of the fact that it is PKI-based, i.e., every principal is required to have a public-private key pair. K-SPKI/SDSI overcomes this disadvantage by using Kerberos for local authentication. We also compare the performance of the two systems based on various parameters such as Time taken to authenticate, Number of key pairs required and Number of certificates generated.

Key words: Trust management systems, SPKI/SDSI, kerberos, certificate chain discovery, authentication, authorization, distributed systems

INTRODUCTION

Systems with shared resources use access-control mechanisms for protection. In a centralized system, authorization is based on the assumption that all of the parties are known or their identity can be established using an authentication system. In a distributed system, that assumption is not valid. Trust-management systems 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 (Blaze *et al*., 1999). 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 popular trust-management system is SPKI/SDSI (Ellison *et al*., 1999).

SPKI/SDSI is a trust management system which uses certificates to provide authentication and authorization. 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 main reason being the need for every principal to have a public-private key pair.

K-SPKI/SDSI is a trust management system derived from SPKI/SDSI. The major addition is the Kerberos, an existing widely deployed authentication system (Neuman and Ts'o, 1994). Specifically, using SPKI/SDSI in conjunction with Kerberos reduces the reliance on PKI by requiring only one public-private key pair per site. This 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 K-SPKI/SDSI, there are two levels of certificates. Each level resembles 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. In this system, the authentication of the user taken care by Kerberos and hence there is no requirement of separate public-private key pair. Thus only the resource owners need to have a public private key pair. K-SPKI/SDSI server accepts certificates from authenticated Kerberos users and generates

Corresponding Author: R. Muthuraj, Department of Information Technology, Madras Institute of Technology, Anna University, Chennai B 600 044, India

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).

In this study, we used a distributed certificate-chain discovery algorithm that generalizes the non-distributed algorithm of Jha and Reps (2004).

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. 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 Σ. The set of identifiers is denoted by 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 ε K and A ε 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 σ, where K ? K and ó is a sequence of identifiers of length greater than one. For example, K UW CS faculty is an extended name.

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, 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 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:

- A name cert (K, A, S, V) will be written as a labeled rewrite rule $KA \xrightarrow{T} S$, where T is the authorization specification such that for all other authorization specifications t, T ∩ t = t and T U t = T. Sometimes we will write $\xrightarrow{T}$ as simply $\longrightarrow$, i.e., a rewrite rule of the form $L \xrightarrow{T} R$ has an implicit label of T.
- An auth cert (K, S, D, T, V) will be written as $K\ w \xrightarrow{T} S\ w$ if the delegation bit D is turned on, otherwise, it will be written as $K\ w \xrightarrow{T} S\ w'$.

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 KUA, in which elements of K appear only in the beginning. For uniformity, we also refer to strings of the form S w and S w' 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,

$$(KA_{Bob} \longrightarrow KB)\ (KA_{Bob}\ \text{my Friends}) = KB\ \text{myFriends}$$

Consider two rules $c_1 = (L_1 \longrightarrow R_1)$ 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: KA friends $\xrightarrow{T}$ KA Bob myFriends
c_2: KA Bob $\xrightarrow{T'}$ KB

The composition $c_2 \circ c_1$ is KA friends $\xrightarrow{T \cap T'}$ KB myFriends. Two rules c_1 and c_2 are called compatible if their composition $c_2 \circ c_1$ is well defined.

The authorization problem in SPKI/SDSI: Assume that we are given a set of certs 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 C is of a sequence of certificates $[c_1, c_2, ..., c_k]$ in C. The label of a certificate chain ch = $[c_1, ..., c_k)$ (denoted by L (ch)) is the label obtained from $c_k \circ c_{k-1} ... \circ c_1$ (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 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

$\{ch_1, \ldots, ch_m\}$ of certificate chains such that for all $1 \le i \le m$

$$\text{Compose }(ch_i)(K_{owner[T]}\ w) \in \{Kw, Kw'\} \text{ and}$$

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

Clarke *et al*. (2001) presented an algorithm for certificate-chain discovery in SPKI/SDSI with O $(n^2_K|C|)$ time complexity, where n_K is the number of keys and |C| is the sum of the lengths of the right-hand sides of all rules in 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:

c_1: (K, K_A, 0, ((dir/etc) read), $[t_1, t_2]$)
c_2: (K, K_A, 0, ((dir/etc) write), $[t_1, t_2]$)

Suppose that Alice makes the request

(KA, ((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.* (2001) 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. Stefan *et al.* (2003) 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 research uses the WPDS-based algorithm for certificate-chain-discovery introduced by Stefan *et al.* (2003).

SPKI/SDSI AND KERBEROS

We describe the authorization scenario in SPKI/SDSI. We also 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 in this part.

Example 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 Sst_1 and Sst_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 Scs the following name certs, which are signed by Alice:

$$\text{Kalice students} \longrightarrow KX$$
$$\text{KAlice students} \longrightarrow KY$$
$$\text{KAlice students} \longrightarrow KZ$$

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:

$$K_{Bob}\ w \xrightarrow{T_v} K_{Bob} \text{ students } w$$
$$K_{Bob}\ w \xrightarrow{T_v} K_{alice} \text{ students } w'$$

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 $\{ch_1, \ldots, ch_m\}$ to U. In 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).

$$c_1 = K_{Alice}\text{ students} \longrightarrow KX$$
$$c_2 = K_{Bob}\text{ w} \xrightarrow{T_V} K_{Alice}\text{ students w'}$$

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 = $\{ch_1, \ldots, ch_m\}$. 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 earlier.

In Example 1, user X wants access to server V according to the authorization specification TV. 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}) w $\varepsilon\{K_X \text{ w}, K_X \text{ w'}\}$ and TV $\subseteq|$ L (ch_1). X presents $\{ch_I\}$ to server V. V checks that TV $\subseteq|$ L (ch_1), which is true and hence V grants U access.

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 detailed descriptions on Kerberos (Neuman and Ts'o, 1994).

The following assumptions are made:

- Each site is a Kerberos realm. The KDC at site st is denoted by KDCst.
- 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 all three components of our authorization scenario are described 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 Ks 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:

$$EK_s\,(U, A, S, V)$$

Where:

U	=	Name of the user
A	=	Identifier
S	=	Subject
V	=	Validity specification

As before, we will write the name cert EKs (U, A, S, V) as U A $\longrightarrow$ S. Upon receiving the encrypted name cert EK_s (U, 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, 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 Ks to the K-SPKI/SDSI server at its site.

$$\text{Alice students} \longrightarrow X$$
$$\text{Alice students} \longrightarrow Y$$
$$\text{Alice students} \longrightarrow Z$$

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

$$\text{KCS Alice students} \longrightarrow \text{KCS X}$$
$$\text{KCS Alice students} \longrightarrow \text{KCS Y}$$
$$\text{KCS Alice students} \longrightarrow \text{KCS Z}$$

A user U at site st sends an auth cert EK_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 EK_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 [Kst U, Kst 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 KSPKI/SDSI server S_{bio}.

$$\text{Bob w} \xrightarrow{T_v} \text{Bob students w'}$$
$$\text{Bob w} \xrightarrow{T_v} \text{CS Alice students w'}$$

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 TV), 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.

$$K_{Bio}\text{ Bob w} \xrightarrow{T_v} K_{Bio}\text{ Bob students w'}$$
$$K_{Bio}\text{ Bob} \xrightarrow{T_v} K_{Bio}\text{ CS Alice students w'}$$

The K-SPKI/SDSI servers also add name certs corresponding to the K-SPKI/SDSI servers of other sites. In our example, SCS signs and adds the name cert KCS Bio $\longrightarrow K_{Bio}$, which states that the public key of the site Bio is KBio. Similarly, SBio signs and adds the name cert

$$K_{Bio}\text{ CS} \longrightarrow K_{CS}.$$

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_αUw or K_αUw', where K_α 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_αU 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 (Caucal, 1992). 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 Ks provided by the SGT.

$$EK_s\,(st_2, V, T)$$

The K-SPKI/SDSI server Sst_1 at site st_1 initiates a distributed certificate-chain discovery request CCDrequest (K_{st1} U, T) on behalf of U. This process involves K-SPKI/SDSI servers, both local and remote, that contain related ESPKI/SDSI certificates. If the request CCDrequest (K_{st1} U, T) is successful and returns a set of certificate chains SCH, user U receives the following token from S_{st1}.

$$\text{Token}_U = EK_s\,(K_1)\,\text{Ticket}_U$$
$$\text{Ticket}_U = EK_{st2}\,(K_2)\,EK_2\,(st_2, K_1, V, T, SCH, TS_1, \text{Lifetime1})$$

In example 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.

$$c_1 = K_{Bio}\text{ Bob w} \xrightarrow{T_v} K_{Bio}\text{ CS Alice students w'}$$
$$c_2 = K_{Bio}\text{ CS} \longrightarrow K_{CS}$$
$$c_3 = K_{CS}\text{ Alice students} \longrightarrow K_{CS}\text{ X}$$

Notice that compose ((c_1, c_2, c_3)) (K_{Bio}Bobw) ε{K_{CS} X w', K_{CS} X } and T_V L (c_1, c_2, c_3).

Upon receiving Token_U, user U decrypts EKs (K_1) and retrieves the keyK_1 (recall thatKs is the session key in the TGT for the K-SPKI/SDSI server at site st_1). User U constructs the following authenticator:

$$\text{Authenticator}_U = EK_1[ID_U||AD_U||TS_2||\text{Lifetime}_2]$$

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

$$\text{Ticket}_U\ \text{Authenticator}_U$$

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 EK_{st2} (K_2) with its private key and retrieves the session key K_2.
- Decrypts the message EK_2 [st_2, K_1, V, T, 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 EK_1 [$ID_U||AD_U||TS_2||\text{Lifetime}_2$]. 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.

COMPARISON IN PERFORMANCE OF SPKI/SDSI AND K-SPKI/SDSI

Thus the major advantage of K-SPKI/SDSI over SPKI/SDSI is that it does not require a separate private-public key pair for each user. In order to compare these two systems, we consider SDSI, which is an implementation of SPKI/SDSI. We evaluate the performance of K-SPKI/SDSI and SDSI in a simulated distributed environment using the algorithm of Jha and Reps (2004). We did not consider the task of issuing certificates as its effect is negligible. The simulated test environment consisted of several sites and with different access controls. 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. This would clearly show the performance of the system for a given access control.

We have tested our implementation in a model where all certificates are stored at the K-SPKI/SDSI servers, which then use the distributed algorithm (Jha and Reps, 2004) for certificate-chain discovery. 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. We analyze the performance of the two systems based on the parameters such as Time to authenticate, Key Pairs required and Number of certificates generated. Also it is assumed that there are 10 users per site to facilitate the comparison between SDSI and K-SPKI/SDSI.

Time to authenticate: Time to authenticate is the time taken for a request made by a user to access a remote resource to be satisfied. This measure indicates how quick the system is, in responding to user requests.

Table 1 presents 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. The time taken to authenticate increases as the number of sites in the system increases.

As we can see time taken for authentication in case of K-SPKI/SDSI is much less than SDSI. This is under the assumption that each site contains 10 users. Also this test performed for the same access control policies. Since in a real world scenario, the number of users per site would be higher than what we have considered, we can conclude that K-SPKI/SDSI requires less time to authenticate than SDSI. This is presented in Fig. 1.

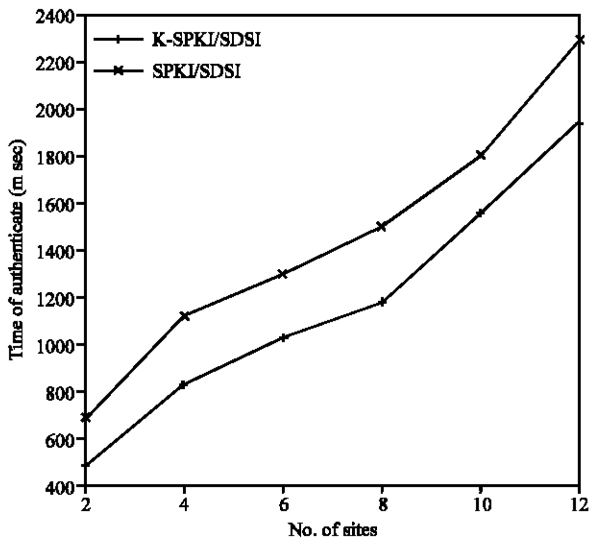

Fig. 1: Plot between number of sites and time to authenticate for K-SPKI/SDSI and SDSI assuming 10 users per site

Table 1: Time to authenticate for K-SPKI/SDSI and SDSI

No. of sites	Time to authenticate (m sec)	
	K-SPKI/SDSI	SDSI
2	488	681
4	840	1120
6	1043	1330
8	1180	1500

Table 2: Number of key pairs required for K-SPKI/SDSI and SDSI

No. of sites	No. of key pairs required	
	K-SPKI/SDSI	SDSI
2	2	20
4	4	40
6	6	60
8	8	80

Number of key pairs required: As we know, both K-SPKI/SDSI and SDSI are PKI based and they require key pairs for communication. This experiment shows the number of key pairs to be generated to perform the required communication. The number of key pairs required for SDSI is much higher than that of K-SPKI/SDSI since SDSI requires each user to have a separate private public key pair (Table 2).

The number of key pairs required for SDSI increases with the user and hence the more the number of users per site, the more the difference in required number of key pairs for these two systems (Fig. 2).

Number of certificates generated: The number of certificates generated depends on the access control policy adopted by the administrator. We consider a system where 90% of the users have access to all the

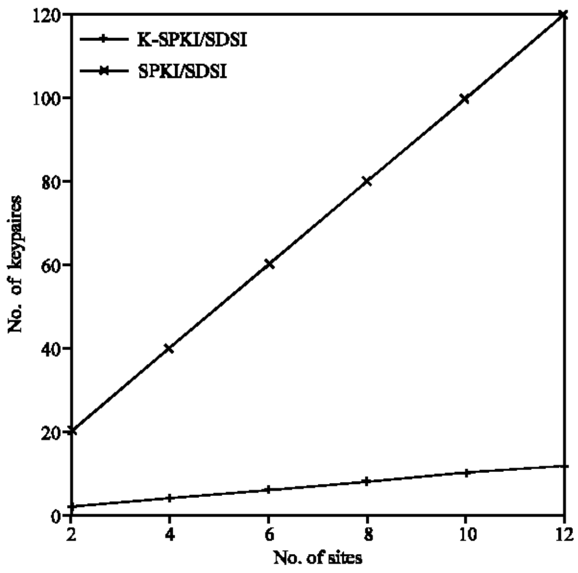

Fig. 2: Plot between No. of sites and number of key pairs for K-SPKI/SDSI and SDSI assuming 10 users per site

Table 3: No. of certificates generated for K-SPKI/SDSI and SDSI

No. of sites	No. of certificates generated	
	K-SPKI/SDSI	SDSI
2	50	330
4	110	1300
6	160	3100
8	230	6200

sites. Under such an access control policy, we analyze the number of certificates generated for each system.

The number of certificates generated depends on the number of sites. As the number of sites increases, the number of certificates generated also increases (Table 3). In case of K-SPKI/SDSI

$$C = \Sigma\,(R_i.S-1) + U,\ 1 \le i \le U$$

In case of SDSI,

$$C = \Sigma\,((U-1).\,Ri), 1 \le i \le U$$

Where:

C = No. of certificates
R_i = No. of resources at site i
U = Total number of users in the system

This is done with the assumption that there are 10 users per site. Since in case of SDSI, certificates are generated for each user, the number of certificates generated exponentially increases with the number of users. But in case of K-SPKI/SDSI its linear (Fig. 3).

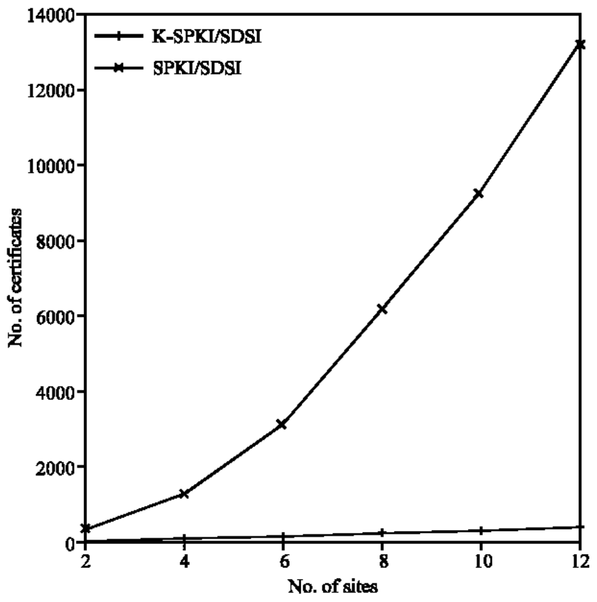

Fig. 3: Plot between number of sites and number of certificates generated for K-SPKI/SDSI and SDSI assuming 10 users per site

Table 4: No. of certificates generated for K-SPKI/SDSI and SDSI with constant number of sites

No. of sites	No. of certificates generated	
	K-SPKI/SDSI	SDSI
2	9300	9300
4	9500	18600
6	10200	27900
8	10700	37200

Number of certificates generated with constant number of sites: Assuming the same access control policy and constant number of sites, the number of certificates generated for SDSI is drastically high (Table 4).

This experiment is done with 100 sites and varying the number of user per site. The number of certificates generated for K-SPKI/SDSI varies linearly with the Number of users where as its exponential in SDSI. We can see that as the number of users per site increases, the difference in performance between the two systems varies drastically (Fig. 4).

Thus these experiments clearly show that K-SPKI/SDSI is much better in comparison with SDSI. Moreover since it is based on Kerberos, a widely deployed authentication system, it still preserves the security of SDSI

However, our implementation of K-SPKI/SDSI is only a prototype and we expect to improve the performance in the future by optimizing the code.

Threat analysis

SPKI/SDSI: The message exchanges are encrypted using the keys provided for each user. A certificate states the

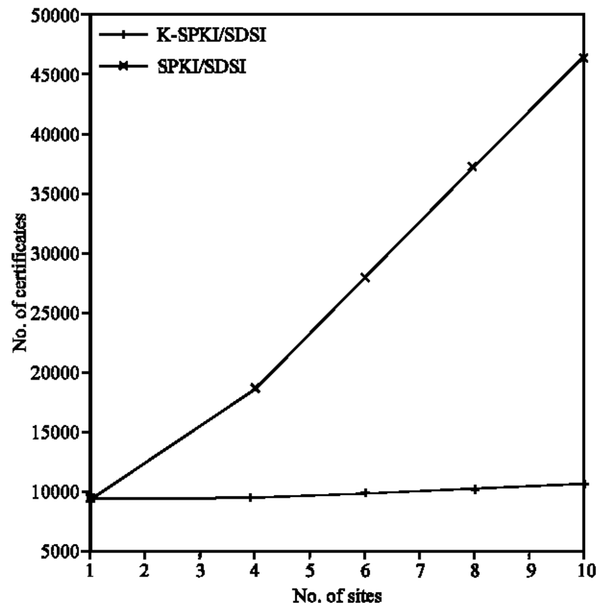

Fig. 4: Plot between number of users per site and number of certificates generated for K-SPKI/SDSI and SDSI assuming 100 sites

relationship between the two users of the access provided to one user for using his resource. Since these certificates can be generated only by the owner of the key (Remember that a certificate is valid only when it is signed by a key), it ensures that access to any resource is made with the consent of its owner. Since the entire system is based on PKI, the risk of attack on the system is minimal.

K-SPKI/SDSI: 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.

RELATED WORK

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 and his colleques (Doster *et al.*, 2001). This idea has been extended by Davis and Swick to build other public-key-style security protocols using secret keys (Ellison *et al.*, 1999). Leveraging the advantages of both Kerberos and Public-Key Infrastructure (PKI) has been explored before. PKINIT (Chuang and Sirbu, 1997), PKCROSS (Jha and Reps, 2004) and PKDA all extend Kerberos by using public-key cryptography for authentication purposes.

K-PKI addresses the problem of accessing Kerberos services from PKI-based systems, such as web applications (Jim, 2001). 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, 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.

One aspect of K-SPKI/SDSI 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 is the closest (Medvinsky and Hur, 1997). 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) (Stefan *et al.*, 2003), ECMA's SESAME (Howell and Kotz, 2000) and Microsoft's Kerberos extension (Caucal, 1992) provide authorization capability through the use of an optional field (called authorization data) provided by Kerberos.

SPKI/SDSI, 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. By making use of Kerberos to reduce SPKI/SDSI's reliance on PKI, K-SPKI/SDSI can be easily adopted to a wide number of environments. Our work essentially compares the two systems based on the parameters listed above and we conclude that K-SPKI/SDSI outperforms SPKI/SDSI both in time and the amount of processing involved.

REFERENCES

Blaze, M., Feigenbaum, J. Ioannidis and A.D. Keromytis, 1999. The role of trust management in distributed systems security. Secure Internet Programming: Security Issues for Mobile and Distributed Objects, LNCS 1603, pp: 185-210.

Caucal, D., 1992. On the regular structure of prefix rewriting. Theor. Comput. Sci., 106: 61-86.

Chuang, J. and M. Sirbu, 1997. Distributed Authentication in Kerberos Using Public Key Cryptography. Internet Draft, Symposium on Network and Distributed System Security.

Clarke, D., J.E. Elien, C.M. Ellison, M. Fredette, A. Morcos and R.L. Rivest, 2001. Certificate chain discovery in SPKI/SDSI. J. Comput. Secur., 9: 285-322.

Doster, B., O. Kornievskaia, P. Honeyman and K. Coffman, 2001. Kerberized credential translation: A solution to web access control. In: 10th USENIX Sec. Sym., pp: 235-250.

Ellison, C.M., B. Frantz, B. Lampson, R. Rivest, B. Thomas and T. Yl¨onen, 1999. SPKI Certificate Theory. The Internet Society, RFC 2693.

Howell, J. and D. Kotz, 2000. A formal semantics for SPKI. Technical Report 2000-363, Dartmouth College, Hanover, NH.

Jha, S. and T.W. Reps, 2004. Model checking SPKI/SDSI. J. Comput. Sec., 12: 317-353.

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

Medvinsky, A. and M. Hur, 1997. Public key utilizing tickets for application servers (PKTAPP). Internet-Draft.

Neuman, C. and T. Ts'o, 1994. Kerberos: An authentication service for computer networks. IEEE Commun. Mag., pp: 33-38.

Stefan, S., S. Jha, T. Reps and S. Stubblebine, 2003. On generalized authorization problems. In: Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW), IEEE Comput. Soc., pp: 202-218.