Extensible Pre-Authentication in Kerberos *

Phillip L. Hellewell, Timothy W. van der Horst, and Kent E. Seamons
Internet Security Research Lab
Brigham Young University
Provo, UT, USA
{sshock, timv, seamons}@cs.byu.edu

Abstract

Kerberos is a well-established authentication system. As new authentication methods arise, incorporating them into Kerberos is desirable. However, extending Kerberos poses challenges due to a lack of source code availability for some implementations and a lengthy standardization process.

This paper presents Extensible Pre-Authentication in Kerberos (EPAK), a Kerberos extension that enables many authentication methods to be loosely coupled with Kerberos, without further modification to Kerberos. To demonstrate the utility of the framework, two authentication methods for open systems are presented that have been implemented as Kerberos extensions using EPAK. These extensions illustrate the flexibility EPAK brings to Kerberos while maintaining backwards compatibility.

1 Introduction

Kerberos [7] is a distributed, identity-based authentication system that allows a user to authenticate once and then connect to application servers within the Kerberos realm without authenticating again for a period of time. Kerberos is time-tested and widely used. Version 5 was standardized over a decade ago, and is used in business, government, military, and educational institutions.

Adopting new authentication methods to replace Kerberos is prohibitive because access control systems and applications are often built up around the Kerberos infrastructure. For example, Microsoft Active Directory is a well-established, enterprise-level authorization system built around Kerberos. Extending Kerberos provides an attractive solution that allows systems like Active Directory to remain intact. However, adding extensions poses challenges due to a lack of source code availability for some implementations and a lengthy standardization process.

*This research was supported by funding from the National Science Foundation under grant no. CCR-0325951, prime cooperative agreement no. IIS-0331707, and The Regents of the University of California.

One trend that has sparked the development of new authentication methods is the increasing need for organizations to provide services to those outside their local security domain. It can be expensive to provision each outside user into a local security domain. *Open systems* allow the authentication of users who are outside the local security domain and do not have a direct pre-existing relationship with the authentication server [12, 1]. Kerberos was not originally designed to support open systems.

This paper presents Extensible Pre-Authentication in Kerberos (EPAK), a Kerberos extension that enables many authentication methods to be loosely coupled with Kerberos, without further modification to Kerberos. Two authentication methods for open systems have been integrated into Kerberos using EPAK to demonstrate the power and flexibility of the framework design.

As a motivating example, suppose Company A creates a collaborative file-sharing service to be fully accessible to the employees of Company B, but provide read-only access to employees at Company C. Rather than manage accounts for each partner's employees, Company A would like to group them into local users $employeeB$ and $employeeC$. Company A would also like to leverage its existing security infrastructure to manage users.

Suppose employees from Company B and C could be authenticated to Company A's domain merely by proving ownership of their email address. Company A could grant and remove access to outsiders simply by mapping an email address to a local user name without having to establish a shared secret. For scalability, the mapping could allow wildcards (e.g., `*@companyB.com`) to map a group of outsiders to a single local name.

The remainder of the paper is organized as follows. Section 2 gives a background of Kerberos. Then Sections 3 to 5 describe the design and implementation of EPAK and two EPAK-based protocols that enable Kerberos to operate as an open system. Section 6 contains a threat analysis of EPAK and Sections 7 and 8 give related work, conclusions, and future work.

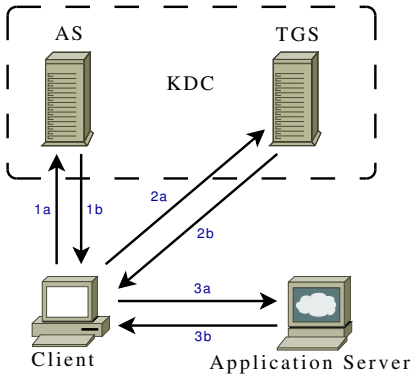

Figure 1. The three-phase Kerberos protocol

2 Kerberos

The Kerberos server consists of an Authentication Server (AS) and a Ticket-Granting Server (TGS). The AS and TGS are responsible for creating and issuing tickets to the clients upon request. The AS and TGS usually run on the same computer, and are collectively known as the Key Distribution Center (KDC).

The Kerberos authentication process consists of three phases (see Figure 1). In the first phase, the client makes a request for a ticket-granting ticket (TGT) from the AS. The AS responds with a TGT, and an encrypted session key needed for the next phase. The session key can be decrypted only by a client that possesses the user's password, which is never communicated over the network. In the second phase, the client presents the TGT to the TGS, which responds with a service-granting ticket (SGT) and a second session key encrypted with the first session key. In the final phase, the SGT is presented to the application server, which then grants access to the service.

Kerberos is stateless [2], which increases scalability. Session keys are not maintained by Kerberos servers, but are included in the TGT and SGT so that the client can communicate securely with the TGS and application servers.

Single sign-on (SSO) is an important feature of Kerberos. With SSO, a user's password must only be entered once per session. The TGT and session key obtained in phase 1 are saved, so each time the user wants to gain access to a service, only phases 2 and 3 are performed. The tickets contain start/end times indicating a valid time period when they can be used. SSO provides convenience, efficiency, and added security.

A Kerberos server (KDC) maintains several secret keys. A single key, K_{tgs}, is used to encrypt the TGT returned in step 1b (see Figure 1). Several keys, K_{c_x}, one for each client, are used to encrypt the session key, also returned in step 1b. Finally, several keys, K_{v_x}, one for each server, are used to encrypt the SGT returned in step 2b.

A credential cache on a client machine stores tickets obtained by a user, such as the TGT and SGTs. Each credential includes a client principal, server principal, encrypted ticket (opaque to the user) and a session key that matches the session key hidden inside the ticket. The credential cache must be secured to prevent impersonation. Heimdal Kerberos secures credentials by storing them in a temporary file, `/tmp/krb5cc_$UID`, which has read/write permissions only for the user who obtained the credential. Other implementations, e.g., Microsoft's, store credentials in memory for greater security.

Users and servers have names called principals [10]. Server principals are composed of a primary name, instance, and realm (name/instance@REALM), while client principals do not have an instance (name@REALM).

Pre-authentication, introduced in Kerberos version 5, allows a client to prove its authenticity before being issued a TGT. A pre-authentication data (`padata`) field in the AS request proves the client's authenticity, such as a timestamp encrypted with the user's password-based key or the user's private key in PKINIT [13]. Pre-authentication prevents an attacker from impersonating a user by obtaining an AS reply and performing an offline dictionary attack against the encrypted data.

Kerberos provides a mechanism whereby authorization information can be embedded into a Kerberos ticket in an *authorization-data* field [7], but not all implementations support this field. Since this data is server specific, a lack of interoperability may arise between Kerberos authentication servers and application servers that do not understand the same authorization data. The Windows 2000 implementation of Kerberos suffers from this incompatibility [5]. Our goal is to pursue a design that will not obligate changes to authorization mechanisms.

Kerberos provides a mechanism for cross-realm authentication that enables an authenticated user in one realm to obtain services in another realm, but it does not scale well because each realm must establish a shared key with every other realm it trusts. Public key extensions to Kerberos improve scalability by eliminating the need to establish shared secrets [2]. In practice, however, a user in one realm is usually provisioned explicitly in another realm.

2.1 Limitations

Conventional Kerberos does not operate as an open system because every user must be known *a priori*. A shared secret between the AS and the user (a password-derived key) must be maintained by the AS, and each user has a 1-to-1 mapping with a principal name.

Most Kerberos extensions are not designed to make Kerberos operate as an open system. PKINIT [13] and other public-key extensions (see Chapter 7) extend credential

management to third parties (trusted CAs), but the third parties usually cooperate directly with the Kerberos administrator in creating certificates with principal names that exist in the database.

Our goal is to extend Kerberos to be an open authentication system, but modifying Kerberos for each new authentication type is burdensome. New authentication types are subject to approval by a standards committee. Once defined, extensions are often rigid and cannot be updated without being re-approved and assigned new pre-auth type numbers. PKINIT has undergone this process.

One might wish to incorporate a proprietary extension into Kerberos without involving the standardization process, but this can be difficult or even impossible when the source code is not available (e.g., Microsoft's implementation). Even when the source code is available, continual resources must be expended to maintain a patch against the latest version of the source code.

3 EPAK Design

Extensible Pre-Authentication in Kerberos (EPAK) is designed to extend Kerberos to support a variety of authentication methods. If large security providers such as Microsoft were to adopt EPAK, many businesses would benefit by having the ability to plug in different authentication protocols, including those that would enable Kerberos to operate as an open system.

Similar to previous Kerberos extensions, EPAK extends just the initial authentication phase to allow the security infrastructure built up around Kerberos to remain unchanged. Unlike existing Kerberos extensions, EPAK enables the integration of many authentication methods without further modification to Kerberos implementations.

The design goals for EPAK are to:

- Allow extensible integration of authentication systems
- Enable attribute-based authentication in Kerberos
- Preserve the existing security properties of Kerberos
- Improve efficiency and usability
- Provide scalable account provisioning for outsiders
- Maintain backwards compatibility with Kerberos

3.1 Architecture

EPAK naturally extends Kerberos by adding a single phase similar to the existing phases (see Figure 2). The EPAK framework enables phase 1 of Kerberos to succeed after a Pre-Authentication Client (PAC) authenticates to a Pre-Authentication Server (PAS) using the desired authentication mechanism. The PAS determines which users can authenticate to which principals. If authentication succeeds, the PAS returns an authentication-granting ticket (AGT) to be used as `padata` in the AS request, and a randomly-generated session key for decrypting the AS reply.

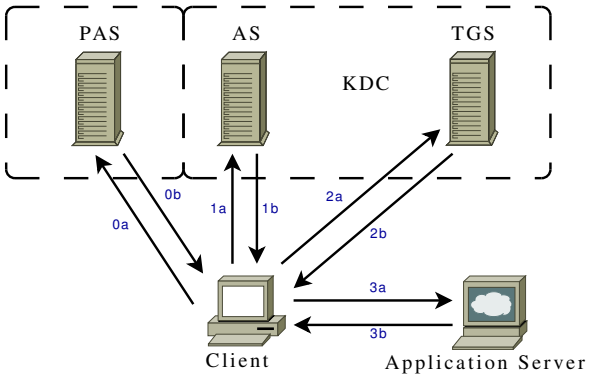

Figure 2. EPAK adds phase 0 to Kerberos

PAS Realms K_{epak} is a randomly-generated key known only to the PAS and AS. By encrypting the AGT with K_{epak} the PAS ensures that only the AS can decrypt it. To provide load balancing and fault tolerance, the PAS may be distributed among multiple machines.

A Kerberos administrator can also choose to outsource pre-authentication by allowing trusted parties to host their own PAS. In this setup, each PAS has its own shared key with the AS, similar to cross-realm authentication. Each party controlling a PAS is known as a *PAS realm*.

To prevent name conflicts and maintain an arms-length trust relationship with each PAS realm, the Kerberos administrator specifies a policy for each PAS realm, indicating all principals the PAS realm is permitted to authenticate. The AS determines in phase 1 which PAS issued the AGT and enforces the corresponding policy.

Outsourcing the PAS offloads principal management in addition to computational work and network traffic. It also allows heterogeneous PAS's supporting different authentication mechanisms within the same Kerberos realm. This increases flexibility, but similar to cross-realm authentication, the tight relationship and shared keys limit scalability.

Principal Mapping As mentioned earlier, the PAS is responsible for mapping users to Kerberos principals and must implement a strategy for determining which users are allowed to authenticate as which principals.

A straightforward strategy is a *1-1* mapping from users to principals. For example, if users are identified by an email address, a formula can be used to convert email addresses into corresponding principals, e.g., john@gmail.com can authenticate as john_gmail_com@REALM. The PAS may incorporate a mapping policy for valid users, or simply rely on the AS to reject principals that do not exist.

Although this approach is more open than traditional Kerberos, which requires a shared secret (user password) to be maintained by the Kerberos server, it remains a closed system since the AS maintains a tight relationship with the

PAS in provisioning a principal for each valid user.

A rule-based approach like a *1-1* mapping is not dynamic enough to allow Kerberos to scale to a large number of outsiders because users are still provisioned individually. A more scalable alternative is to map a group of users to a Kerberos principal without requiring that each user be provisioned in the local Kerberos realm in advance. Two such strategies are described below.

The first strategy, an *m-1* mapping, provides a coarse-grained approach to map users to a single principal. For instance, all users at partner companies can be mapped to a guest principal, e.g., guest@REALM. This dynamic arrangement provides increased scalability because the local Kerberos administrator manages only a single principal and is shielded from all changes to the user population at partner companies. However, mapping users to a single principal is not flexible because all outsiders are treated uniformly.

The second strategy, an *m-n* mapping, is a fine-grained approach that provides a balance of flexibility and scalability. An attribute-based policy can specify which groups of users can authenticate as which principals. Principals can be defined to represent large groups (e.g., companyC@REALM, partner@REALM).

Combining this *m-n* mapping technique with multiple PAS realms produces an even finer-grained, adaptable solution for user management. Consider the scenario presented in Section 1. Company A avoids having to manage accounts for each employee of Company B by grouping them all together (e.g., `*@companyB.com`). Although dynamic and scalable, this configuration may be too coarse-grained for Company A's needs. To enable a more fine-grained setup, Company A could entrust Company B to run a PAS that authenticates users to specific principals, such as `developerB`, `customerB`, and `salesB`. This does not preclude Company A from continuing to use a coarse-grained approach with Company C.

3.2 Protocol

The EPAK protocol makes use of four messages, defined in Table 1, where the AGT is referred to as the $epakticket$. The EPAK protocol is divided into two authentication phases: pre-authentication and AS authentication (see Figure 3). In EPAK, the PAC and the AC are separate programs that communicate through the client's credential cache to allow phase 0 to be customized by EPAK-based protocols without further modification to phase 1.

Pre-Authentication Phase During pre-authentication, a valid client obtains an EPAK-REP from the PAS. The pre-authentication protocol is shown in Figure 3, phase 0:

a) The PAC sends an EPAK-REQ to the PAS to indicate the principal requesting authentication. Additional messages may be exchanged in order for the client to complete the authentication

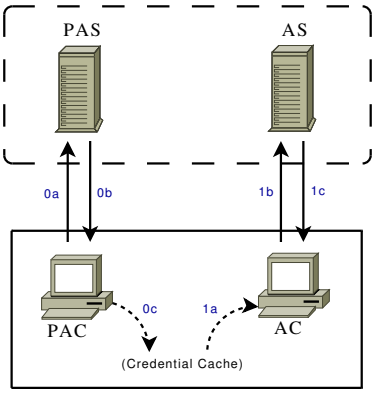

Figure 3. Pre-authentication and AS authentication

b) The PAS responds with an EPAK-REP

c) The $epakticket$ and session key $K_{c,as}$ of the EPAK-REP are stored in the client's credential cache under the server name `epakt/REALM@`$pasrealm$

The $epakdata$ identifies the client, and specifies requested ticket start/end times. The times are then restricted by the PAS in the EPAK-REP to enforce the maximum lifetime, similar to other Kerberos tickets.

The EPAK-REP must be communicated securely to protect the session key $K_{c,as}$ from eavesdroppers, and to prevent replay. TLS or another suitable mechanism may be used to transmit the EPAK-REP securely.

The PAS verifies the EPAK version number and then performs any other steps the particular authentication algorithm might require. The PAS must only return an EPAK-REP if the user proves authenticity and is allowed to authenticate to the desired principal.

AS Authentication Phase The protocol for AS authentication with EPAK pre-authentication data is shown in Figure 3, phase 1:

a) The encrypted $epakticket$, $pasrealm$, and session key $K_{c,as}$ are retrieved from the client's credential cache

b) The client generates $epakauth$ and sends an AS request with PA-EPAK-AS-REQ as the `padata`

c) The server responds with an AS response with PA-EPAK-AS-REP as the `padata`. The session key $K_{c,tgs}$ is encrypted with the session key $K_{c,as}$

The $epakauth$ included in the PA-EPAK-AS-REQ shows that the client has recent knowledge of the session key in the $epakticket$. It serves the same purpose as the Authenticator used in phase 2 and 3 [7].

If authentication fails, the PA-EPAK-AS-REP contains an error result value and the encrypted part of the AS reply

EPAK Messages

EPAK-REQ	$epakvno \parallel epakdata$
EPAK-REP	$epakvno \parallel epakdata \parallel pasrealm \parallel K_{c,as} \parallel E_{K_{epak}}[epakticket]$
PA-EPAK-AS-REQ	$epakvno \parallel pasrealm \parallel E_{K_{epak}}[epakticket] \parallel E_{K_{c,as}}[epakauth]$
PA-EPAK-AS-REP	$epakvno \parallel result$

EPAK Message Elements

$epakvno$	EPAK version = 1
$epakdata$	$cname \parallel crealm \parallel starttime \parallel endtime$
$epakticket$	$K_{c,as} \parallel epakdata$
$epakauth$	$cname \parallel crealm \parallel cksum \parallel cusec \parallel ctime$
$K_{c,as}$	Random session key generated by PAS
K_{epak}	PAS's key for encrypting $epakticket$
$pasrealm$	PAS's realm
$cname$	Client name (principal name)
$crealm$	Client realm (principal realm)
$starttime$	Starting time of $epakticket$
$endtime$	Expiration time of $epakticket$
$cksum$	Checksum of AS request (excl. `padata`)
$cusec$	Timestamp [7] (microseconds)
$ctime$	Timestamp [7]
$result$	Authentication error/success code

Table 1. EPAK Message Definitions

is set to unusable random data. Alternatively, a Kerberos error message may be returned.

The $pasrealm$ indicates the PAS that issued the $epakticket$, and is used to identify the appropriate EPAK key needed to decrypt the ticket. It is also used to identify the correct policy specifying principal names when multiple PAS realms are involved.

The AS must enforce the following PA-EPAK-AS-REQ verification rules before generating a successful AS reply.

1. $epakvno$ must be a valid version number
2. $epakauth$ must be valid as in RFC 4120
3. $epakticket$ realm must match the realm of the AS
4. $epakticket\ starttime \leq now$
5. $epakticket\ endtime > now$
6. $epakticket$ principal must exist in Kerberos database
7. $epakticket$ and AS request principals must match
8. $epakticket$ principal must appear in policy

Rule 6 maintains harmony with current Kerberos implementations, and its absence would necessitate dynamic creation of principals, or modifications to later phases to handle unknown principals. Such changes would have far-reaching effects into the systems built around Kerberos.

The lifetime of the TGT is limited so as not to extend beyond the lifetime of the $epakticket$.

3.3 Advantages and Disadvantages

EPAK benefits both Kerberos and the systems it integrates with Kerberos.

Incorporating attribute-based authentication methods into Kerberos enables an open system that allows services built on Kerberos to scale to larger communities. Also, the need for shared keys between clients and the AS is eliminated, along with the risk to the client of a compromised central repository.

Complex authentication systems with slow performance may benefit from Kerberos' SSO capabilities. EPAK's low integration barrier may accelerate the adoption of newer or lesser-known systems.

EPAK supports backward compatibility. A Kerberos server with EPAK still supports normal Kerberos password-based authentication. A Kerberos server without EPAK support fails gracefully with a "pre-auth type not supported" error whenever it receives an EPAK authentication request.

One drawback to EPAK is that it requires at least one extra round of communication. The PAC must communicate with the PAS to obtain the $epakticket$. Other extensions, such as PKINIT [13], provide pre-authentication data in the AS request without an additional phase.

4 Open Systems in EPAK

To demonstrate the generality and flexibility of EPAK, two authentication systems were integrated into Kerberos: Simple Authentication for the Web and trust negotiation.

4.1 SAW

Simple Authentication for the Web (SAW) [11] leverages email (or other personal messages, e.g., text and instant messages) to authenticate users. SAW significantly improves upon the basic technique employed by the "Forgot your password?" link common to many web sites.

In SAW, users must demonstrate their ability to retrieve two short-lived, single-use *Authentication Tokens* (see Figure 4). If a user-supplied email address is authorized, a random secret, $AuthToken_{complete}$, is generated and split into two shares as follows:

$$AuthToken_{complete} \oplus AuthToken_{email} = AuthToken_{user}$$

where $AuthToken_{email}$ is a randomly generated value. $AuthToken_{user}$ is returned directly to the user over the secure link used to initiate the authentication process (e.g., HTTPS) while $AuthToken_{email}$ is emailed. If the user returns both tokens then the authentication is successful.

Since $AuthToken_{user}$ is returned over a secure link, passively observing $AuthToken_{email}$ is worthless.

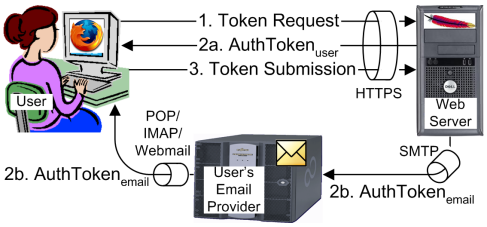

Figure 4. The SAW protocol

Vulnerability to Active Impersonation By submitting a victim's email address to a site an attacker obtains a valid $AuthToken_{user}$. Consequently, by observing the victim's incoming email messages, the attacker acquires the corresponding $AuthToken_{email}$ and is able to authenticate as the victim. This is called an active impersonation attack.

SAW's threat analysis argues that SAW provides an acceptable level of risk, even in light of this attack, because sites that employ email-based password resets (EBPR) are also susceptible to a similar attack in which an attacker requests a password reset for the victim and then observes the resulting email message sent by the site. The prolific adoption of EBPR indicates that these risks are manageable.

One-Round SAW Step 3 of SAW is eliminated in *one-round* SAW by setting $AuthToken_{complete}$, normally a random value, to the item requested by the user. Since only authentic users can reconstruct $AuthToken_{complete}$, only those users will be able to obtain the item. As the token splitting used by SAW creates two shares of equal size to the secret it splits, it is advised for a large item to encrypt the item, split the encryption key, and then deliver the encrypted item with one of the encryption key shares.

Group-Based SAW SAW is often used in closed systems, i.e., an ACL specifies all authorized email addresses. This works well for sites (e.g., forums or photo-sharing) willing to provision accounts for each user.

Unfortunately, this one-to-one specification of users to permissions is insufficient for open systems. For example, this approach requires Business A, from the scenario described in Section 1, to maintain an ACL containing some or all of the employee emails of its affiliate, Company B.

For more flexibility, SAW can be modified to use ACLs that contain wildcards or regular expressions. This is known as *group-based* SAW. With this enhancement, Business A can specify that anyone with a Company B email address (e.g., `*@companyB.com`) is allowed access.

4.1.1 SAWK Naïve Approach

A naïve approach to integrating SAW into Kerberos is to send an email address in the AS request, inside `padata`, of type PA-SAW-AS-REQ. The AS replies with the $AuthToken_{user}$ and the session key $K_{c,tgs}$ encrypted with $AuthToken_{complete}$, and emails $AuthToken_{email}$.

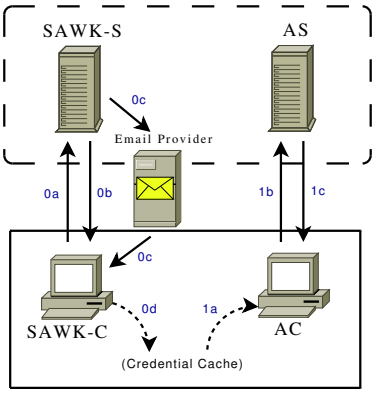

Figure 5. The SAWK protocol

After retrieving $AuthToken_{email}$, the user reconstructs $AuthToken_{complete}$ and unlocks the session key.

As with most Kerberos extensions, the adoption of SAW with this naïve approach would be impeded until it was approved and integrated into popular Kerberos implementations. Before integration, a patch file would have to be maintained, and Kerberos would have to be built manually to enable this functionality.

This approach also provides no mechanism for securing $AuthToken_{user}$, making it susceptible to eavesdropping.

4.1.2 SAWK Protocol

Simple Authentication for the Web in Kerberos (SAWK) is an EPAK-based protocol that enables flexible, email-based authentication in Kerberos, and avoids the limitations of the naïve approach.

Pre-Authentication Phase The protocol for SAWK pre-authentication is shown in Figure 5, phase 0:

a) The SAWK-C sends an EPAK-REQ and email address to the SAWK-S

b) If the address is allowed to authenticate as the principal specified in the EPAK-REQ, the SAWK-S responds with $AuthToken_{user}$ and an EPAK-REP encrypted with the random $AuthToken_{complete}$

c) $AuthToken_{email}$ is emailed to the specified address and is used to reproduce $AuthToken_{complete}$ and decrypt the EPAK-REP

d) The *epakticket* and session key $K_{c,as}$ of the EPAK-REP are stored in the client's credential cache

The communication between SAWK-C and SAWK-S (steps 0a and 0b) is secured (e.g., TLS) to thwart eavesdropping and impersonations of the SAWK-S.

SAWK uses group-based SAW for a flexible mapping of email addresses to principals. The addresses are specified as regular expressions, which provide an *m-n* mapping with a high level of scalability.

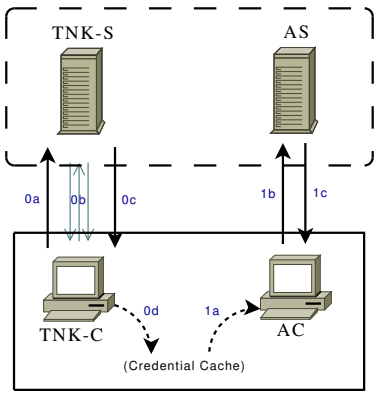

Figure 6. The TNK protocol

AS Authentication Phase The protocol for AS authentication following SAWK pre-authentication is shown in Figure 5, phase 1. This phase is identical to phase 1 of EPAK. As previously mentioned, EPAK-based authentication protocols can be integrated into Kerberos without further modification to the Kerberos client and server programs.

4.2 Trust Negotiation

Trust negotiation [12, 1] is a protocol for establishing trust between strangers with no preexisting relationship. Automated trust negotiation works by exchanging digital credentials until enough trust has been established to gain access to a service or resource. If each party has the required credentials, and their policies allow them to be shown to each other, then trust negotiation will succeed.

An access control policy defines what credentials must be supplied before access to a resource is granted. Policies can also be used to protect sensitive credentials. For instance, a credit card credential won't be disclosed unless the other party has a Better Business Bureau credential.

4.2.1 TNK Protocol

Trust Negotiation in Kerberos (TNK) is an EPAK-based protocol that uses trust negotiation to authenticate clients.

Pre-Authentication Phase The protocol for TNK pre-authentication is shown in Figure 6, phase 0:

a) The TNK-C sends an EPAK-REQ to the TNK-S
b) Trust negotiation is performed until the policy has been satisfied, or trust negotiation fails
c) If the policy is satisfied, an EPAK-REP is returned
d) The $epakticket$ and session key $K_{c,as}$ of the EPAK-REP are stored in the client's credential cache

The principal name in the EPAK-REQ serves as the role the user must satisfy before the EPAK-REP is disclosed. By its very nature, trust negotiation provides a scalable, attribute-based mapping of users to principals.

The communication between the TNK-C and TNK-S is performed over a secure TLS connection to protect potentially sensitive credentials (step 0b), provide server authentication, and to prevent an eavesdropper from viewing the session key $K_{c,as}$ in the EPAK-REP (step 0c).

AS Authentication Phase The protocol for AS authentication after TNK pre-authentication is shown in Figure 6, phase 1. This phase is identical to phase 1 of EPAK.

4.2.2 TNK vs PKINIT

PKINIT [13] is a Kerberos extension that uses public-key cryptography for initial authentication. Similar to TNK, only phase 1 of the protocol changes. But unlike TNK, PKINIT authentication does not require additional rounds.

In PKINIT, the user sends a certificate to the AS. After verifying its validity (signed by a trusted CA and not revoked or expired), the AS responds with the TGT and session key. The session key is encrypted with the user's public key extracted from the certificate, instead of a password-derived key. PKINIT also allows a key generated through a Diffie-Hellman key exchange to be used for this encryption.

PKINIT relies on trusted CAs to issue certificates for users. The principal names are usually specified directly in the certificates, creating a one-to-one mapping between certificates and principals. This limits PKINIT's ability to operate as an open system, since the CAs must work directly with the Kerberos administrator in managing principals.

PKINIT can function as an open system if the AS is modified to use a different binding mechanism from certificate properties to Kerberos principals [13]. One way to achieve this is to map large groups to principal names. For example, the subject name of the certificate maps to a principal name via regular expression mapping, similar to the SAWK mapping policy. Other certificate properties could also be involved in the mapping to provide an even more flexible, attribute-based solution, similar to TNK.

5 EPAK Implementation

The EPAK implementation is shown in Figure 7. EPAK is implemented as a patch to Heimdal Kerberos [3], and this section is geared towards those familiar with Kerberos implementations. Additional details are available in [4].

Changes to the client include modifying `kinit` and adding helper programs `genpatrequest` and `savepat`. Changes to the server include modifying `kdc` and adding the `genpatreply` helper program.

genpatrequest A PAC utility for generating an EPAK-REQ. The principal name is specified (if different from the user's name), as well as the ticket lifetime and start time.

genpatreply A PAS utility for generating an EPAK-REP from a valid EPAK-REQ. It must be run by a privileged user with access to the EPAK key (K_{epak}) stored

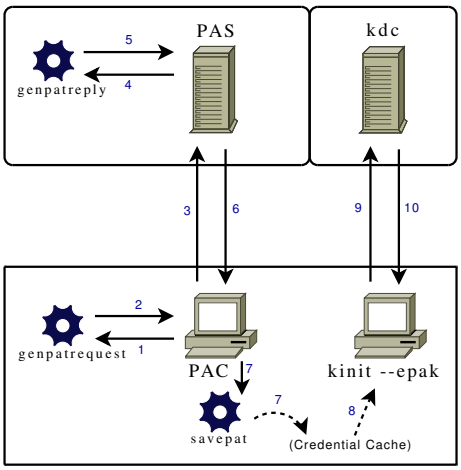

Figure 7. EPAK Implementation

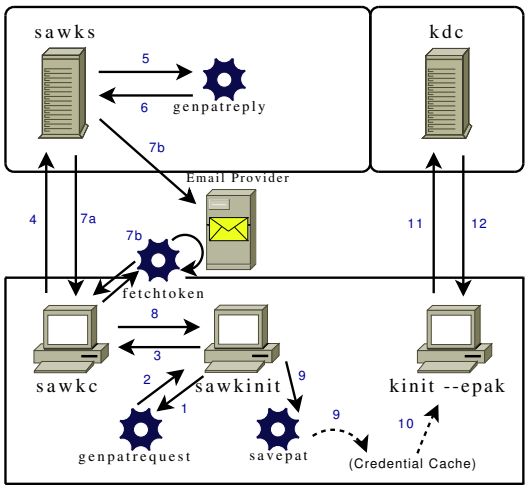

Figure 8. SAWK Implementation

in the `krb5.keytab` file. K_{epak} is used to encrypt the *epakticket* in the EPAK-REP. The verification rules specified in Section 3.2 are enforced. The existence of the client principal is not enforced because:

1. The PAS may not have access to the Kerberos database, especially if the PAS is running on a different machine.
2. The principal name will be verified later, by the kdc when it receives the AS request.

savepat A PAC utility for saving an EPAK-REP to the client credential cache. The *epakticket*, session key $K_{c,as}$, and *pasrealm* from the EPAK-REP are formatted into a `krb5_creds` which is then stored into the credential cache with `krb5_cc_store_cred()`. The credential can be viewed using `klist`.

kinit A Kerberos program extended to support a new option, `--epak` for performing EPAK instead of password-based authentication. The `epakt/REALM` service credential, which holds an *epakticket* and session key $K_{c,as}$, is read from the credential cache. If it does not exist or is expired, `kinit` aborts with an error. An *epakauth* is created and encrypted with $K_{c,as}$, and is sent along with the *epakticket* in a `padata` of type PA-EPAK-AS-REQ to the AS. If the AS reply includes a PA-EPAK-AS-REP indicating success, `kinit` uses $K_{c,as}$ to decrypt part of the reply.

kdc A Kerberos program that performs the function of the AS. It supports EPAK by recognizing and responding appropriately to PA-EPAK-AS-REQ `padata`. K_{epak} is used to decrypt the *epakticket*, and the rules specified in Section 3.2 are enforced, including verification of *epakauth*, principal name, and ticket times. The AS reply includes the TGT and session key $K_{c,tgs}$ like normal, but $K_{c,tgs}$ is encrypted with $K_{c,as}$ instead of a K_c. A PA-EPAK-AS-REP is also included in the reply.

A new option, `epak_ticket_lifetime`, added to krb5.conf indicates the maximum lifetime of an EPAK ticket. The default value is eight hours.

5.1 SAWK Implementation

The SAWK implementation is shown in Figure 8.

sawk A script that runs `sawkinit` followed by `kinit --epak`, to perform pre-authentication and AS authentication in one command.

sawkinit A script that runs `genpatrequest`, `sawkc`, and `savepat`. The `sawkinit.conf` configuration file specifies the location of the these programs and the hostname and port of the SAWK-S machine. The ticket times, if specified, are forwarded to `genpatrequest`, and the credential cache name is forwarded to `savepat`.

sawkc and sawks Java programs that perform one-round, group-based SAW authentication. They communicate over TLS to protect the $AuthToken_{user}$ returned in step 7a. `sawksc` sends an email address, specified in `sawkc.properties`, and EPAK-REQ in step 4. `sawks` checks if the email address is valid and maps it to a principal name according to the `sawks.policy` file, which uses regular expressions to group email addresses. In step 7a, `sawks` responds with three items: $AuthToken_{user}$, an EPAK-REP encrypted with $AuthToken_{complete}$, and a transaction ID that helps identify the email of step 7b. To prevent leaking valid/invalid addresses, an authentication failure returns a random value in place of the EPAK-REP. In step 7b `fetchtoken` retrieves $AuthToken_{email}$, which is XOR-ed with $AuthToken_{user}$ to produce $AuthToken_{complete}$ for decrypting EPAK-REP.

fetchtoken A helper C program that polls the email provider to obtain the $AuthToken_{email}$. The email to

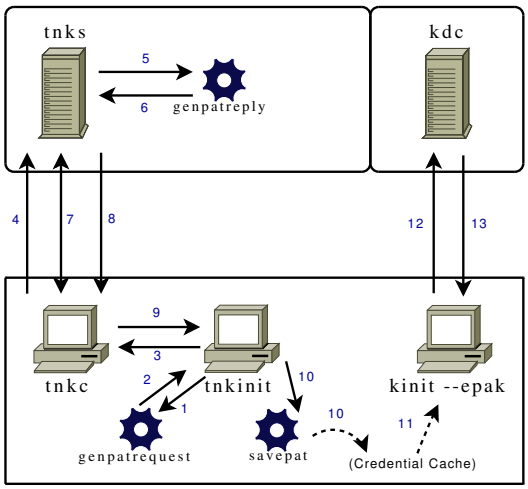

Figure 9. TNK Implementation

retrieve is identified by a transaction ID and the SAWK-S hostname (to help prevent phishing attacks). The email subject line contains the transaction ID, hostname, and $AuthToken_{email}$ to facilitate quick retrieval. The $AuthToken_{email}$ is saved to a specified token file to be read by `sawkc`. Account properties for email retrieval are specified in the configuration file `fetchtoken.conf` and include username, email protocol, mail server, and timeout. Valid email protocols include POP3 and IMAP, optionally over TLS.

5.2 TNK Implementation

The TNK implementation is shown in Figure 9.

tnk A script that runs `tnkinit` followed by `kinit --epak`, to perform pre-authentication and AS authentication in one command.

tnkinit A script that launches the `genpatrequest`, `tnkc`, and `savepat` programs. A configuration file, `tnkinit.conf`, specifies the location of the three programs and the hostname and port of the machine running the TNK-S. The ticket times, if specified, are forwarded to `genpatrequest`, and the credential cache name is forwarded to `savepat`.

tnkc and tnks Java programs that perform trust negotiation to obtain the resource "Authenticated as principal X". Policy files dictate what credentials must be released to obtain this resource. The `tnkc` and `tnks` communicate over TLS to protect potentially sensitive credentials. The EPAK-REQ is sent in step 4, trust negotiation is performed in step 7, and the EPAK-REP is returned in step 8 if trust negotiation succeeds.

6 Threat Analysis

Kerberos has several security properties that make it resistant to attacks (see Section 2): mutual authentication, message integrity, and confidentiality. EPAK aims to retain these properties and reduce opportunities for new flaws by extending phase 1 with the standard `padata` mechanism, and leaving phases 2 and 3 unchanged.

Phase 0 (pre-authentication) adopts the proven Kerberos approach for tickets and session keys. An $epakticket$ is confidential and integrity protected. Only the PAS and AS can access it, and any modifications can be detected. The $epakauth$, created by the client, is encrypted with the session key $K_{c,as}$ so that it can be viewed only by the AS. It is short-lived and provides assurance that the client presenting the ticket is the one who was issued the ticket.

If EPAK-REQ is not communicated securely, an eavesdropper can replay it. However, an attacker must either prove his authenticity before he obtains an EPAK-REP or the PAS encrypts the EPAK-REP in a manner that only an authentic client can obtain the decryption key.

The PAS must safeguard the session key returned in EPAK-REP in order to prevent an eavesdropper from gaining access. It needs to be encrypted, and the implementations of SAWK and TNK encrypt the entire EPAK-REP using TLS.

TLS communication between the PAC and PAS provides server authentication and DNS spoofing protection. TNK supports additional server authentication during negotiation.

Access to a client's credential cache enables impersonation. Kerberos tickets can be used multiple times and therefore require persistent storage. Both system administrators and those with physical access to a client's machine can impersonate a user holding a valid ticket. An $epakticket$ is non-renewable, so it presents less risk than other tickets. This risk is also mitigated by implementations that store credentials in memory instead of on disk.

The `padata` of phase 1 (PA-EPAK-AS-REQ) may be replayed by an attacker, but is ineffective for two reasons. First, the $epakauth$ is short-lived, providing a small window of vulnerability. Second, even within that window the AS reply is useless to an attacker who does not possess the session key $K_{c,as}$.

An $epakticket$ for one principal cannot be used to authenticate to a different principal; an expired $epakticket$ will also be rejected. A client should not be allowed to obtain a ticket with an arbitrary lifetime. The PAS restricts the ticket lifetime by enforcing the rules specified in Section 3.2. Likewise, the AS only accepts valid tickets meeting the conditions delineated in Section 3.2.

Each EPAK authentication method must undergo a thorough security analysis. SAWK and TNK inherit the security risks of SAW and trust negotiation, respectively, as well as the security of TLS. The design and use of the EPAK-REQ and EPAK-REP messages in phase 0 must also be analyzed.

7 Related Work

Public key based Kerberos for Distributed Authentication (PKDA [9]) relieves the load on a Kerberos KDC server by off-loading authentication to the application servers. Clients have no contact with the KDC using this protocol. PKDA is essentially a replacement for SSL, but the authors themselves admit that SSL is a "formidable" solution.

PKINIT [13] is a Kerberos extension that moves Kerberos beyond password-based authentication to public-key cryptography, which provides greater scalability. EPAK builds on ideas from PKINIT and other public-key extensions to enhance Kerberos in similar ways.

Role-based Access Control (RBAC) [8] maps user identities to roles within an organization. Users authenticate to known subjects, and then subjects are assigned a role(s). All access control policies are specified in terms of roles. This indirection provides scalability. As users enter and leave the system, the role assignment rules change, but all access control policies remain the same. EPAK leverages this same idea to map users to Kerberos principals.

GSSAPI [6] is a generic API for client/server authentication. Since most Kerberos distributions include a GSS-API implementation, applications that support GSSAPI also support Kerberos. Extending Kerberos with EPAK allows these applications to support many other authentication systems. Alternatively, an authentication system could just implement the GSSAPI interface, but that would not afford it the benefits of Kerberos (like SSO), and it could not be used with Kerberized services that do not support GSSAPI.

8 Conclusions and Future Work

EPAK is a powerful framework for incorporating diverse authentication mechanisms into Kerberos. EPAK cleanly separates pre-authentication and AS authentication so that a variety of authentication systems can be loosely coupled with Kerberos, while maintaining backwards compatibility. Two concrete examples, SAWK and TNK, have been designed and implemented to demonstrate the extensibility of EPAK.

SAWK and TNK provide grouping techniques that increase Kerberos' ability to scale to larger numbers of people since services can be provided to outsiders without managing individual user accounts. Large Kerberos deployments (e.g., Microsoft Active Directory) could adopt EPAK to provide businesses with a deployment path to more open systems.

New authentication systems can now be integrated more easily into Kerberos to increase their usability and performance without changing existing Kerberos-based security frameworks. Complex authentication systems with slow performance can leverage Kerberos' SSO capability.

EPAK extensibility makes it possible for individuals and organizations to independently extend Kerberos without the overhead of the standards process. However, the thorough peer review that the standards process provides must remain, but can occur in a more decentralized manner.

In the future, additional authentication protocols can be incorporated into Kerberos using EPAK to test its extensibility and guide further enhancements. EPAK standardization is a worthy goal. The types PA-EPAK-AS-REQ and PA-EPAK-AS-REP need to be assigned reserved values to avoid conflicts. An EPAK RFC is a natural next step for it to become an IETF standard.

The software for EPAK, SAWK, and TNK is available at http://isrl.cs.byu.edu.

References

[1] E. Bertino, E. Ferrari, and A. C. Squicciarini. Trust-X: A Peer-to-Peer Framework for Trust Establishment. *IEEE Transactions on Knowledge and Data Engineering*, 2004.

[2] A. Harbitter and D. A. Menascé. Performance of Public Key-Enabled Kerberos Authentication in Large Networks. In *IEEE Conference on Security and Privacy*, Oakland, CA, May 2001.

[3] Heimdal Kerberos 5 Impl. http://www.pdc.kth.se/heimdal/.

[4] P. Hellewell. Extensible Pre-Authentication in Kerberos. Master's thesis, Computer Science Department, Brigham Young University, Aug 2007.

[5] D. Kearns. Kerberos and Windows 2000. *Network World Fusion*, Mar 2000.

[6] J. Linn. RFC: 2743: Generic Security Service Application Program Interface Version 2, Jan 2000.

[7] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. RFC 4120: The Kerberos Network Authentication Service (V5), Jul 2005.

[8] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, Feb 1996.

[9] M. A. Sirbu and J. C.-I. Chuang. Distributed Authentication in Kerberos Using Public Key Cryptography. In *Network and Distributed System Security*, Feb 1997.

[10] J. G. Steiner, B. C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *USENIX Technical Conference*, 1988.

[11] T. W. van der Horst and K. E. Seamons. Simple Authentication for the Web. In *3rd International Conference on Security and Privacy in Communication Networks*, Sep 2007.

[12] W. H. Winsborough, K. E. Seamons, and V. E. Jones. Automated Trust Negotiation. In *Information Survivability Conference and Exposition*, Jan 2000.

[13] L. Zhu and B. Tung. RFC: 4556: Public Key Cryptography for Initial Authentication in Kerberos (PKINIT), Jun 2006.