Distributed Authentication in Kerberos Using Public Key Cryptography

Marvin A. Sirbu
sirbu@cmu.edu

John Chung-I Chuang
chuang+@cmu.edu

Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

In this work we describe a method for fully distributed authentication using public key cryptography within the Kerberos ticket framework. By distributing most of the authentication workload away from the trusted intermediary and to the communicating parties, significant enhancements to security and scalability can be achieved as compared to Kerberos V5. Privacy of Kerberos clients is also enhanced. A working implementation of this extended protocol has been developed, and a migration plan is proposed for a transition from traditional to public key based Kerberos.

1. Motivation

The scalability of network security infrastructures is becoming a growing concern as the explosive growth of the Internet continues unabated. The number of new users and applications requiring authentication will continue to increase at a rapid rate for the foreseeable future. The proliferation of web-based commerce, for example, will add to the network tens of millions of daily transactions between large numbers of geographically distributed merchants and consumers. Authentication schemes are needed which can scale to easily handle millions of principals within a single realm of trust, such as all the customers of a major bank.

As Neuman et al. have noted [11] the traditional Kerberos presents an attractive security target in the form of the KDC which maintains a shared symmetric key with every principal in the realm. In the event of a KDC compromise, all the symmetric keys will be divulged to the attacker and will have to be revoked. Recovering from such a compromise requires the re-establishment of new shared keys with all principals in the realm. Such a recovery is very costly in terms of time, effort and financial resources.

In this paper, we attempt to address both of these concerns through the integration of public key cryptography with traditional Kerberos authentication.

1.1. The role of trusted intermediaries in authentication

Public key based systems rely on Certificate Authorities as trusted intermediaries when authenticating clients and servers. Key Distribution Centers in Kerberos are another form of trusted intermediary. Scalability of the trusted intermediary is a challenge in either system. If the challenge is not met, users can experience significant delays in authentication, or be forced to accept an increased risk of fraudulent credentials.

In the Kerberos scheme, the KDC issues to clients a relatively short lived credential (a Ticket Granting Ticket-TGT) which must then be presented to a centralized Ticket Granting Service (TGS) to obtain a session ticket for a particular server. The KDC is burdened by the need to constantly renew these short-lived TGT's, and the TGS must be involved every time a client wishes to establish contact with a new server. Typically, the short lifetime of a TGT provides the only protection against revoked credentials; a TGS replica will not be informed when the KDC revokes a user's privileges.

In a public key scheme, a Certificate Authority (CA) issues a relatively long lived credential -- a public key certificate. When both clients and servers have such certificates they can authenticate to each other without further reference to a CA. However, precisely because these certificates are long-lived, some method is required to inform servers of revoked certificates. This can be done by requiring servers to check a certificate's current validity with the CA on each use of a certificate, or by distributing Certificate Revocation Lists (CRL's) to all servers periodically.

Thus, the factors controlling the burden on centralized services are different in the two cases. In Kerberos, the burden on the KDC/TGS is determined by the number of times clients want to authenticate to servers. In a Public Key scheme, it is determined by the frequency with which clients or servers must be in touch with a CA to learn of revoked certificates.

Partitioning of the principals into realms served by different servers is one method of improving scalability. This must be coupled with a means for cross-realm authentication. In the Kerberos scheme, each realm has its own well-defined administrative boundary, and more importantly, a manageable number of principals. A Kerberos KDC and/or TGS may be replicated in each realm to handle all authentication requests within the realm, though one instance of the KDC must be declared the master for updates of shared secrets. Cross-realm authentication is made possible through the bi-lateral establishment of inter-realm keys by the realm administrators. This is accomplished by registering the Ticket Granting Server of one realm as a principal in the other. Even though Kerberos realms can, in theory, be organized hierarchically, bi-lateral agreements must be established a priori and few Kerberos realms are cross registered in practice. Nevertheless an organization, such as a bank with millions of customers, can handle the issue of scalability by dividing its customer base into separate realms and setting up cross-realm authentication among them.

Similarly, a chain of certificates allows a principal to verify a certificate issued by a different CA than the principal's own.

In practice companies such as Verisign have already demonstrated the ability to handle from a single CA the burden of hundreds of thousands of certificates. By incorporating public key cryptography into the Kerberos framework, it should be possible to establish a Kerberos realm with millions of principals, with no need for cross-realm authentication.

There have been numerous recent proposals to incorporate public key cryptography into Kerberos [3] [11]. These proposals focus on various aspects of Kerberos, such as security and portability. The centralized KDC remains in all of the proposals. The present work, extended and generalized from the NetBill security and transaction protocol [2], seeks to address both the scalability and security concerns by bypassing the centralized KDC altogether. This proposed extension to Kerberos shall be referred to as "Public key based Kerberos for Distributed Authentication" or "PKDA" in the remainder of this paper.

Neuman et al. propose to solve the security problem by calling for the use of public key cryptography in the initial authentication between the clients and the KDC. By registering only the public keys with the KDC, the clients will not have to re-generate a new shared secret in the event of a KDC compromise. Only the application servers, which will continue to use conventional cryptography, will have to re-establish new symmetric keys. Limiting the use of public key cryptography to the initial authentication is justified on performance grounds. Once key-exchange is achieved during initial authentication, all subsequent message exchanges, including those to secure server tickets from a TGS, can be accomplished using the computationally more efficient symmetric key method.

The proposed PKDA extension requires the use of public key operations each time a service ticket is required. However, these operations are distributed among the clients and servers, rather than concentrating them at the KDC. Additionally, it offers a more complete solution to the security problem. Fully distributed authentication between the Kerberos clients and servers using public key cryptography means that neither the clients nor the servers will need to maintain symmetric keys with the KDC. In fact, there is no longer a centralized KDC to be compromised. Only the CA remains as the trusted intermediary.

Finally, while Kerberos V5 and its predecessors provide the means to protect the content of messages, they provide no protection against traffic analysis, as the identities of the parties to a communication are typically sent in the clear during session establishment. With a simple modification to the message format, it is possible to preserve the privacy of the client identity at very little cost.

1.2. PKDA versus SSL

Version 3.0 of the Secure Sockets Layer protocol [5] provides public key based services for mutual authentication, and key exchange for privacy. It has been rapidly adopted by many firms for use in conjunction with HTTP, as well as for other services such as telnet.

Like the PKDA proposal, SSL 3.0 allows a client and server who are each in possession of a public key certificate signed by a trusted CA to mutually authenticate and establish a shared symmetric session key. Moreover, SSL specifications cover the use of multiple public key algorithms (DSA, RSA) and multiple session key algorithms (DES, RC4) as determined by the parties. While Kerberos provides a means of specifying multiple encryption methods, most implementations support only DES.

We see four fundamental advantages to PKDA vis-à-vis SSL 3.0:

1. SSL 3.0 is a transport layer protocol which can only be used in conjunction with TCP-based client-server communications. Kerberos, on the other hand, sits at the application layer and can be used with either UDP or TCP. However, this transport layer independence comes at a price: it is more work to integrate Kerberos with each application.
2. In order to avoid the computational burden of public key operations at the beginning of every TCP connection, SSL 3.0 permits the reuse of session keys. To make use of this facility, however, requires the server to keep a cache of all recently issued session keys and an associated serial number. Kerberos (and PKDA) relieves servers from maintaining such state information, by conveying the session key in an encrypted ticket readable only by the server. Moreover, agreeing to use cached session keys in SSL requires the exchange of several packets at the beginning of each TCP connection.

No such exchange is required to re-use a previously issued Kerberos ticket.

3. Kerberos tickets can be used to encrypt messages which transit one or more servers on the way to their ultimate destination. This is not possible with SSL 3.0.
4. The Kerberos proxy mechanism provides a convenient way of conveying rights limitations along with authentication. Restricted rights could only be implemented in SSL via certificate extensions which have not been specified. Rights delegation in PKDA will be discussed in Section 2.5.

Despite these advantages for PKDA, the widespread support for SSL in the marketplace makes it a formidable alternative.

2. The PKDA Protocol

In the RFC 1510 protocol specification for Kerberos V5 [7], a normally executed (error-free) authentication procedure begins with the exchange of the following five messages:

1. C --> AS:	AS_REQ
2. AS --> C:	AS_REP
3. C --> TGS:	TGS_REQ
4. TGS --> C:	TGS_REP
5. C --> S:	AP_REQ

where

AS_REQ:	Authentication Service Request
AS_REP:	Authentication Service Response
TGS_REQ:	Ticket Granting Service Request
TGS_REP:	Ticket Granting Service Response
AP_REQ:	Application Service Request

The client first obtains a ticket granting ticket (TGT) from the Authentication Service (AS) of the KDC, with whom a shared symmetric key has previously been established. Using this TGT, the client communicates with the Ticket Granting Server (TGS) to secure a shared session key between itself and the server with which it wishes to communicate. The client can then proceed to request the desired service from the application server in step 5. This sequence of message exchanges is preserved in the proposed extension by Neuman et al. [11]. The extension departs from conventional Kerberos only in the first two steps, where the messages are encrypted and signed using the public key pairs of the client and the KDC. The client will continue to receive and use conventional TGT's and service tickets.

In the proposed PKDA protocol, the Kerberos KDC is bypassed altogether. Communicating directly with the application server, the client will be able to request the server's certificate and establish a session ticket without necessarily contacting a centralized intermediary. A PKDA-enabled server will be able to service certificate requests as well as issue session tickets in the capacity of the TGS, albeit only for sessions with itself:

1. C --> S:	SCERT_REQ
2. S --> C:	SCERT_REP
3. C --> S:	PKTGS_REQ
4. S --> C:	PKTGS_REP
5. C --> S:	AP_REQ

where

SCERT_REQ:	Request for Server's Certificate
SCERT_REP:	Provision of Server's Certificate
PKTGS_REQ:	Public key based TGS Request
PKTGS_REP:	Public key based TGS Response
AP_REQ:	Application Service Request

It is evident from the above that the PKDA protocol can be executed in a completely distributed fashion. Yet the protocol is not computationally more demanding than other public key based proposals such as SSL. The first two steps require no cryptographic operations, since the information being transferred is all public information and the integrity of the certificate is verified in subsequent steps. In the case where the client has certificate caching capabilities, these two steps can even be bypassed for repeated authentication. Again, public key cryptography is limited to initial authentication only, which occurs in step 3 in this protocol. All subsequent steps employ the faster symmetric cryptography. While the client generates and sends a public key based TGS request in step 3, it receives a conventional service ticket in step 4 and normal operations proceed from step 5 onwards.

The PKDA protocol extension to Kerberos V5 is built upon existing public key infrastructure standards such as PKCS [12] and X.509 [6]. Detailed protocol specifications in ASN.1 format can be found in [13]. A sample specification of the protocol, based upon the Interface Specification Language [1], can also be found in [8].

2.1. Notation

The following notation will be used to denote the parties, operations, and key variables involved in the message exchanges described in the rest of the paper:

C	Client
S	Server
G	Rights Grantee (Proxy)
CA	Certificate Authority
K_{rand}	random one-time symmetric key
$K_{c,s}$	symmetric key shared by C and S
P_s	public key of S
P_c^{-1}	private key of C
$\{M\}K$	message encrypted using key K

$\{M\}P_s$	message encrypted using S's public key
$\{M\}P_c^{-1}$	message signed using C's private key
Ts#	time-stamps
T_{auth}	initial authentication time
$T_{c,s}$	ticket for session between S and C

2.2. Obtaining the server's public key certificate

The client initiates the authentication exchange by requesting from the server the public key certificate of the application service. This is necessary since the construction of the subsequent ticket request message (PKTGS_REQ) requires the encryption of data using the server's public key. The SCERT_REQ message simply consists of the identity (principal name and realm) of the server:

SCERT_REQ: S

This and all subsequent PKDA authentication messages are directed to the application service's assigned port.

In response to the request, the server returns its certificate or certificate-chain, which can be transmitted via an unprotected channel:

SCERT_REP: s-cert

If the client has certificate caching capabilities, the above two steps may be bypassed for subsequent authentication attempts with a server.

The client is responsible for verifying that the certificate has been signed by a trusted CA or chain of CA's, and that it has not been subsequently revoked. This can be accomplished either by checking against a regularly-updated copy of the Certificate Revocation List (CRL), or by performing a query to the Certificate Authority (CA). Alternatively, the client may choose to secure the server's certificate directly from the CA. In either case, the client must have established secure communications with the CA, for example using PKDA with the CA's well known public key. The operational details of the CA/CRL infrastructure are beyond the scope of this paper.

2.3. Client/server authentication using public key cryptography

2.3.1. Public key based TGS request. Once the client has obtained and verified the server's public key certificate, it can proceed to generate the service ticket request. The message contains similar information to that in a conventional ticket request, but it is sent to the server directly, rather than to the KDC as in traditional Kerberos. This message is digitally signed with the client's private key, and encrypted with the server's public key. Therefore, the server and only the server can determine and authenticate the identity of the client. Conversely, the client is assured of the identity of the server because only the server with the matching private key can decrypt the PKTGS_REQ and construct a valid response.

The critical fields of the PKTGS_REQ message are as follows:

PKTGS_REQ: S, $\{T_{auth}, K_{rand}, \text{auth-data}\}P_s$

where

$$\text{auth-data} = C, \text{c-cert}, \{K_{rand}, S, P_s, T_{auth}\}P_c^{-1}$$

The server's identity, S, is the only field transmitted in the clear. All other fields are encrypted for either security or privacy reasons. The integrity of the field S is guaranteed by its inclusion in the authorization field, which is digitally signed. The server identity must be in the clear so that the listener process receiving the PKTGS_REQ message knows for which principal the message is intended in the event that multiple principals are served from the same server port.

The remainder of the message is encrypted using the public key of the server, P_s. (In practice this means encrypting the message with a symmetric key which in turn is encrypted using P_s.) There are three distinct elements in this encrypted portion, namely the authentication time T_{auth}, the random key K_{rand}, and the digitally signed authorization field 'auth-data'.

The field T_{auth} indicates the time of the initial authentication request, which is the time at which the current request message is being generated. Since the client generates this timestamp, the server will have to verify the time elapsed between this timestamp and when it receives this message. By refusing to service a ticket request that occurred 'too far' in the past, i.e., beyond the acceptable clock skew, the server can prevent replay attacks. This field should not be left in the clear since any observable lag would indicate to a potential attacker that the client may have a clock synchronization problem which can be exploited.

The field K_{rand} is a random one-time key generated by the client. In traditional Kerberos, this random key is generated by the KDC for the client to use in communicating with the TGS in the TGT_REQ/ TGT_REP exchange. The client now generates this random key. This key is not the actual session key, but is rather used by the server to encrypt the response, which contains the service ticket and the session key itself.

The generation of this random key does impose a burden on the client to have an appropriate random number generator. However, the client does have access to the user's private key, which can be used along with other available sources of entropy to seed a quality pseudo-random number generator. It should be noted, in comparison, that traditional Kerberos requires the generation by the client of a nonce, but the nonce can be based either on a random number generator or on timestamps. The inclusion of this one-time random key in the message eliminates the need for a separate nonce.

The third and final element, 'auth-data', is in essence an authorization field digitally signed with the client's private key. This field contains the information necessary to authenticate the client's identity and to check the integrity of the message.

The construction of 'auth-data' deserves special attention. We choose to represent the 'auth-data' field using the 'SignedData' construct as specified in the PKCS Cryptographic Message Syntax Standard (PKCS #7) [12]. The 'SignedData' container includes not only a placeholder for the content to be signed, but also placeholders for the client's identity C, and the client's certificate 'c-cert' (among other supporting fields needed for public key operations). There is therefore no need to explicitly duplicate these fields elsewhere in the request message.

The client's identity and certificate fields, while part of the 'auth-data' construct, are not themselves 'signed' per se. The fields actually subject to the signature process are the random key K_{rand}, the server's identity S, the server's public key P_s, and the timestamp T_{auth}. The random key K_{rand} is signed for authenticity, while the server identity is included to prevent replay attacks first addressed by Denning-Sacco [4]. The server's public key (or any other identifier which uniquely ties the server's certificate to the key used in encrypting this PKTGS_REQ message) will serve to avert "man-in-the-middle" attacks. An alternate candidate for this field may be a combination of 'Certificate.issuer' and the issuer specific 'Certificate.serialNumber' fields in a X.509 certificate. The server, S, by comparing this unique identifier with the same field found in its own certificate copy, will detect any attempt by a previous server S^* to reuse the client's signed auth-data in an attempt to obtain a ticket in C's name for S. Finally, the timestamp T_{auth} is included to prevent replay attacks.

Note that the client's identity and its certificate can only be found within the encrypted portion of the message. The same is true for the PKTGS_REP message to be described in the next section. This is in contrast to the conventional TGS_REQ and TGS_REP messages, where the client's identity is transmitted in cleartext. The impact of this change on client privacy will be revisited in Section 2.3.2.

Upon receipt of the PKTGS_REQ message, the server decrypts the message using its private key. It then retrieves the client's public key, found in the client's certificate, c-cert. Using this key, the client's signature (and authenticity of the request) can be verified. The server may also want to check for any revoked client certificates.

2.3.2. Response to a PKTGS request. The server, in its ticket granting service capacity, responds with a PKTGS_REP message very similar to the TGS_REP message of traditional Kerberos:

$$\text{PKTGS\_REP:} \quad T_{c,s}, \{C, S, K_{c,s}, T_{auth}\}K_{rand}$$

This message, like the TGS_REP, consists of the service ticket $T_{c,s}$ and an encrypted part. The ticket is just a conventional ticket, identical to that issued by the traditional TGS:

$$T_{c,s} \quad = \quad S, \{K_{c,s}, C, T_{auth}\}K_s$$

Here, the ticket is encrypted using K_s, a symmetric key known only to the server. This prevents the client from modifying the ticket. In traditional Kerberos, the symmetric key is shared between the server and the TGS. Of course, the server and the TGS are the same entity in PKDA. This reinforces the fact that the server can only issue session tickets for clients to communicate with itself, and not with any other application server. This is in contrast to traditional Kerberos, where the centralized TGS can issue session tickets for any application service that has registered itself with the TGS.

Moving to the encrypted portion of the message, it can be noted that the client identity, C, is no longer transmitted in plaintext, as is the case in traditional Kerberos. This modification echoes that of the PKTGS_REQ change described in the previous section. The purpose of this pair of changes is to offer a greater degree of protection of client privacy. While this approach does not prevent a network observer from capturing IP address information, it does prevent the tracking of session requests between identifiable client and server pairs, as is the case with Kerberos V5. There are many ways to reduce the value of any captured IP address information. Corporate firewalls often perform IP address remapping, thus making IP addresses less useful to an observer outside the firewall. (Note, however, that in this case, if the optional 'caddr' field is set in the PKTGS_REQ message, it should be set to the IP address of the firewall, not of the client.) Dial-up IP service providers dynamically assign IP addresses to customers. Finally, the client may be on a multi-user host. In the event that multiple authentication requests originate from a single IP address, the matching of the PKTGS_REP to the correct PKTGS_REQ can still be unambiguously achieved by examining the originating port address. However, a network observer would not have direct access to the client's identity.

The session key $K_{c,s}$ is also included in the encrypted response. The client will use this session key to construct the authenticator exactly as in traditional Kerberos.

It is worth noting that the encryption key used in PKTGS_REP is the random key K_{rand}, extracted from the PKTGS_REQ message. This key serves the same function as the session key extracted from the TGT in traditional Kerberos. Additionally, since K_{rand} is a one-time key generated by the client, it can and does serve a second role as the nonce.

2.4. Using the service ticket

The service ticket received by the client is simply a conventional service ticket. Therefore, the client generates the application service request as before:

AP_REQ: $T_{c,s}$, {C, Ts1}$K_{c,s}$

All operations from this point on can proceed per normal Kerberos operations. Direct authentication between client and server is thus accomplished.

2.5. Rights delegation in PKDA

Kerberos V5 supports rights delegation via the use of proxiable and forwardable tickets. Neuman showed that it is straightforward to implement proxies in encryption-based authentication mechanisms based on either public key or conventional cryptography [9].

Under the proposed PKDA scheme, it is possible to achieve rights delegation even in the absence of a centralized KDC. As long as the destination server is PKDA-enabled and is configured to provide proxiable tickets, the same delegation mechanism used in Kerberos can be applied here.

If the client C wants G (the rights grantee) to act as its proxy in communicating with the destination server S, it will first establish a session with G and then initiate the following exchange:

1. C --> S:	SCERT_REQ
2. S --> C:	SCERT_REP
3. C --> S:	PKTGS_REQ (with PROXIABLE flag set)
4. S --> C:	PKTGS_REP (returns proxiable ticket)
5. C --> G:	KRB_CRED
6. G --> S:	AP_REQ

The client contacts the application server directly to request a proxiable ticket with the address of G included in the ticket. The full PKTGS_REQ message format, like the AS_REQ and TGS_REQ message formats, allows the client to set the PROXIABLE flag and specify the addresses from which the ticket is usable [13]. The PKDA-enabled server simply responds with a proxiable ticket. The client will then construct an authenticator which the grantee will present to the application server. The authenticator includes a proxy key in the 'subkey' field, and this key will be used as the actual session key between G and S. The client can also set the desired proxy restrictions in the 'authorization-data' field within the authenticator.

Using the KRB_CRED construct in traditional Kerberos, the client will forward the proxiable ticket, the authenticator and the proxy key to the grantee in step 5. Since C and G have already established a secure session with each other, the proxy key and other encrypted information in KRB_CRED will not be divulged to an eavesdropper. Furnished with this 3-tuple of ticket, authenticator, and proxy key, the grantee can then proceed to communicate with the application server on the client's behalf as per traditional Kerberos.

There may be a scenario where the client cannot communicate directly with the application server. For example S may be located behind a firewall and can only be accessed via the gateway G. In such a case G can simply serve as a relay for the messages exchanged in steps 1-4. Since the PKDA protocol has been designed to be immune to a "man-in-the-middle" attack, there is nothing that G can do to compromise the protocol (beyond denial of service).

3. Migration plan - obtaining service tickets from a PKDA-enabled TGS

If the server with whom the client wishes to communicate is not capable of handling service ticket requests using the PKDA protocol, the client will have to resort to sending the request to a centralized PKDA-enabled TGS. We should realistically expect that a hybrid of PKDA-enabled and non-PKDA-enabled application servers will be in co-existence during a system-wide transition to PKDA. Therefore the existence of these interim centralized TGS's as PKDA-servers of last resort is essential to preserving the functionality and integrity of the Kerberos Authentication scheme during the transition.

When the application server fails to respond to a certificate request (SCERT_REQ) with a matching SCERT_REP, the client can assume that the server is not PKDA-enabled. Then the client will have to communicate with a PKDA-enabled TGS to get a traditional TGT and the subsequent service ticket for the application server. This is accomplished by the following seven step exchange:

1. C --> TGS:	SCERT_REQ
2. TGS --> C:	SCERT_REP
3. C --> TGS:	PKTGS_REQ
4. TGS --> C:	PKTGS_REP
5. C --> TGS:	TGS_REQ
6. TGS --> C:	TGS_REP
7. C --> S:	AP_REQ

It is worth noting that the client will be communicating with the TGS for the first six steps of the exchange. Therefore, the performance bottleneck associated with a centralized KDC/TGS remains. However, the centralized database of symmetric keys will be much smaller in size, since the clients will now authenticate themselves to the TGS using public keys instead of symmetric keys. Shared symmetric keys between the KDC and the application servers are still required.

The essential features of this message exchange are as follows:

SCERT_REQ:	TGS
SCERT_REP:	tgs-cert
PKTGS_REQ:	TGS, $\{T_{auth}, K_{rand}, \text{auth-data}\}P_{tgs}$
PKTGS_REP:	$\{C, TGS, K_{c,tgs}, T_{auth}\}K_{rand}$, TGT
TGS_REQ:	C, S, Ts1, TGT, $\{\text{authenticator}\}K_{c,tgs}$
TGS_REP:	C, $\{K_{c,s}, S, Ts1\}K_{c,tgs}$, $T_{c,s}$
AP_REQ:	$T_{c,s}$, $\{C, Ts2\}K_{c,s}$

where

auth-data	=	C, c-cert, $\{K_{rand}, TGS, P_{tgs}, T_{auth}\}P_c^{-1}$
TGT	=	$T_{c,tgs}$
	=	TGS, $\{K_{c,tgs}, C, T_{auth}\}K_{tgs}$
$T_{c,s}$	=	S, $\{K_{c,s}, C, T_{auth}\}K_{s,tgs}$

The first four steps of the message exchange are identical to those of the PKDA protocol as described in Section 2, and so the discussion will not be repeated here. The only exception is that the server being contacted in this case is the TGS (which has to be PKDA-enabled.) Therefore, all instances of and references to 'S' are replaced by 'TGS'. In effect, the client is issuing a PKTGS_REQ to the TGS requesting a "service ticket" in the form of a traditional TGT. Then, steps 5-7 are really identical to steps 3-5 of the traditional Kerberos exchange, where the client uses the TGT to request an actual service ticket. Since this traditional TGT can be reused, only steps 5-7 need to be repeated for subsequent authentication with other non-PKDA-enabled servers.

It is obvious from this message format, however, that the protection of the client's identity in steps 3 and 4 of this exchange is betrayed by the traditional Kerberos exchange in steps 5 and 6. Therefore, the privacy enhancements of PKDA cannot be realized unless clients contact servers directly using the PKDA protocol.

An alternate approach to transition might have each server host running a localized PKDA-enabled TGS at a well-known port. This "localized TGS" would have certificates for each server principal running on that host and share a symmetric key with each service. With this setup, steps 1-4 of the PKDA protocol could be directed to this well-known port instead of the assigned port of the specific application service. The session ticket obtained from this "localized TGS" can then be used to contact an unmodified traditional Kerberized application service running at the host. We do not advocate this approach, however, as it is likely to further delay the complete migration to PKDA-enabled applications.

4. Progress

The fundamental correctness of the PKDA protocol has been verified using Convince, a formal verification tool based on belief logic. This work is reported by Lichota et al. [8]. A working implementation of PKDA has been developed for the NetBill electronic payment project at CMU [14]. NetBill consumers and merchants authenticate with each other, as well as with the NetBill server, in a fully distributed manner using the handshake described above. NetBill uses DCE RPC's between clients and servers and steps 1-4 of the PKDA protocol are additional RPC's that must be supported on any server's interface. A pre-processor to the standard DCE IDL compiler is used to automatically add these two RPC's to every interface. An Internet Draft [13] detailing the PKDA protocol, in its full specification, has been submitted to the IETF. This proposal is compatible with, and can be integrated with other current proposals (such as [11]) to bring public key cryptography into Kerberos.

5. Conclusion

An extension to the Kerberos authentication framework has been described. This extension employs public key cryptography to facilitate initial authentication directly between Kerberos servers and clients. By distributing the authentication workload from the centralized KDC's to the individual principals on the network, potential communications and/or processing bottlenecks at the KDC's can be avoided. This significantly enhances the scalability of the Kerberos framework in the rapidly growing Internet environment. By eliminating the need for a KDC and its centralized database of symmetric keys, the catastrophic consequences of a KDC compromise, including the subsequent key-recovery effort, can be avoided.

It has also been shown that the privacy of the client's identity in a Kerberos authentication can be protected. By moving the client identity fields from unencrypted to encrypted portions of the messages, potential eavesdroppers will be prevented from extracting such information.

The feasibility and robustness of the PKDA protocol has been successfully demonstrated via a formal verification process and the development of a working implementation. A migration plan has been proposed for a transition from traditional to public key based Kerberos.

Acknowledgments

The authors would like to acknowledge Ben Cox and J. Doug Tygar for their contributions to the design and implementation of PKDA, and Cliff Neuman and the anonymous referees for their helpful comments on this paper.

This work is sponsored by the Air Force Materiel Command, under Advanced Research Projects Agency Contract No. F19628-95-C-0018, "Electronic Commerce: The NetBill Project. " Additional support comes from the National Science Foundation under Cooperative Agreement No. IRI-9411299. The views and conclusions in this paper remain those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Advanced Research Projects Agency, the National Science Foundation, or the U.S. Government.

Kerberos is a trademark of the Massachusetts Institute of Technology; NetBill is a trademark of Carnegie Mellon University.

References

[1] S. Brackin. An Interface Specification Language for Automatically Analyzing Cryptographic Protocols, *Internet Society Symposium on Network and Distributed System Security*, February 1997.

[2] B. Cox, J.D. Tygar, M. Sirbu. NetBill Security and Transaction Protocol. In *Proceedings of the USENIX Workshop on Electronic Commerce*, July 1995.

[3] D. Davis. Kerberos Plus RSA for World Wide Web Security. In *Proceedings of the USENIX Workshop on Electronic Commerce*, July 1995.

[4] D.E. Denning, G.M. Sacco. Timestamps in Key Distribution Protocols. *Communication of the ACM*, 24(8):533-536, August 1981.

[5] A.O. Freier, P. Karlton, P.C. Kocher. Secure Socket Layer 3.0. Internet Draft, March 1996. (ftp://ietf.org/ internet-drafts/draft-freier-ssl-version3-01.txt)

[6] International Telegraph and Telephone Consultative Committee (CCITT). Recommendation X.509: The Directory Authentication Framework. 1988.

[7] J. Kohl, C. Neuman. The Kerberos Authentication Service (v5). Internet RFC 1510, September 1993.

[8] R. Lichota, G. Hammonds, S. Brackin. Verifying Cryptographic Protocols for Electronic Commerce. *2nd USENIX Workshop on Electronic Commerce*, November 1996.

[9] B.C. Neuman. Proxy-Based Authorization and Accounting for Distributed Systems. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, May 1993.

[10] B.C. Neuman, T. Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications*, 32(9):33-38, September 1994.

[11] B.C. Neuman, B. Tung, J. Wray, J. Trostle. Public Key Cryptography for Initial Authentication in Kerberos. Internet Draft, October 1996. (ftp://ietf.org/internet-drafts/draft-ietf-cat-kerberos-pk-init-02.txt)

[12] RSA Laboratories. PKCS #7: Cryptographic Message Syntax Standard. Version 1.5, November 1993.

[13] M. Sirbu, J.C. Chuang. Public key Based Ticket Granting Service in Kerberos. Internet Draft, May 1996. (ftp://ietf.org/internet-drafts/draft-sirbu-kerb-ext-00.txt)

[14] M. Sirbu, J.D. Tygar. NetBill: An Internet Commerce System Optimized for Network Delivered Services. *IEEE CompCon Conference*, March 1995.