Public-key Cryptography Extensions into Kerberos

Ian Downard
University of Missouri – Rolla
Department of Electrical and Computer Engineering
1870 Miner Circle
Rolla, MO 65409
Phone: 573-341-8422
Fax: 573-341-4532
itd@umr.edu

Key words: computer security, public-key cryptography, authentication

March 26, 2002

Abstract

How can and why should the Kerberos authentication standard (RFC1510) be extended to support public-key cryptography? These are the questions which we explore in this paper. We outline three proposed extensions to integrate public-key cryptography into Kerberos, and make some basic performance comparisons between them. We also comment on the security issues public-key enhancements introduce to the Kerberos trust model. A brief introduction to Kerberos and public-key cryptography is provided for those unfamiliar with these security systems.

Introduction

Integrating public-key cryptography (PKC) within Kerberos represents the leading edge of proposed enhancements to the current Kerberos standard. Several recent publications have contributed novel solutions to accomplish this task, and each has its own advantages and disadvantages. Before any proposal is fully adopted, issues like performance and security must be analyzed. Actually addressing those issues in depth is beyond the scope of this paper, but the motivation to study multiple unique solutions with those issues in mind is stimulating.

We begin with overviews of PKC and Kerberos, and then discuss what improvements PKC can offer to the traditional Kerberos standard. After summarizing three different protocols for public-key enhanced Kerberos, we explain the performance penalties associated with PKC and reference qualitative results from other research which compares the performance of two of our described PKC specifications. Finally, we summarize some of the security issues associated with bringing public-key support into the traditional Kerberos framework.

Public-Key Cryptography Primer

A private key (KR) and public key (KU) are related such that for any plaintext, x,

$$x = D_{KR}(E_{KU}(x)) = D_{KU}(E_{KR}(x))$$

where

$D_{KR}(Y)$ denotes decryption of ciphertext Y with KR,
$D_{KU}(Y)$ denotes decryption of ciphertext Y with KU,
$E_{KU}(X)$ denotes encryption of plaintext X with KU, and

$E_{KR}(X)$ denotes encryption of plaintext X with KR.

That is, data encrypted with KR can only be decrypted with KU, and visa versa. Public-key cryptosystems use these two related keys to perform some type of cryptographic function. Those functions generally fall into the following three categories, (where "Alice" denotes the source of a message intended for receiver, "Bob"):

1. **Encryption/Decryption:** A message encrypted with the Bob's public key can only be deciphered by Bob, with his (presumably secret) private key.
2. **Key Distribution:** Alice can communicate a secret key to Bob by encrypting it with Bob's public key and sending him the resulting ciphertext. Bob can then recover the secret key by decrypting Alice's message with his private key.
3. **Digital Signatures:** Alice encrypts a publicly readable message with her private key. Anyone who decrypts her ciphertext with her public key and obtains the original publicly available message has verified the authenticity of Alice's signature and the integrity of her message.

The security of PKC is entirely dependant on the secrecy of private keys, and the widespread availability of their corresponding public keys in unaltered form. The later issue can be addressed by using protocols like X.509 [9] which employ trusted Certificate Authorities to manage the distribution of trustworthy public keys.

Kerberos Introduction

Kerberos (RFC1510) is a security system developed at the Massachusetts Institute of Technology in 1988 for providing a secure means of communication between incorporated programs and for providing a secure means of authenticating users into a network of incorporated workstations and servers [1] [2]. Since secure authentication is the most important achievement of the Kerberos protocol, it is important to know exactly what kind of authentication Kerberos offers. Traditionally, authentication has focused only on verifying the identities of human users (as clients) on a network, while the authenticity of servers has been presumed valid. Kerberos extends the reach of guaranteed identity by also authenticating servers in the Kerberos network (a feature called *mutual authentication*). This assures clients that the services they are requesting will be processed by authorized servers, and not by other machines possibly masquerading as legitimate servers. Furthermore, Kerberos does not ever need to transmit passwords, even in encrypted form, in order to authenticate a client user.

After users have been authenticated, most Kerberos implementations also offer the following features:

- *remote logins* by way of the "kerberized" `rlogin`, `telnet`, and `ftp` utilities.
- *encrypted communications* with the DES or 3DES encryption algorithm
- *single sign-on capabilities*, so a user can access kerberized services without logging in more than once.

These features are available on the "realm" of computers that have been specifically selected by an administrator to be part of a kerberized network. (Realms are described in the next section.) A basic Kerberos protocol is summarized in Figure 1.

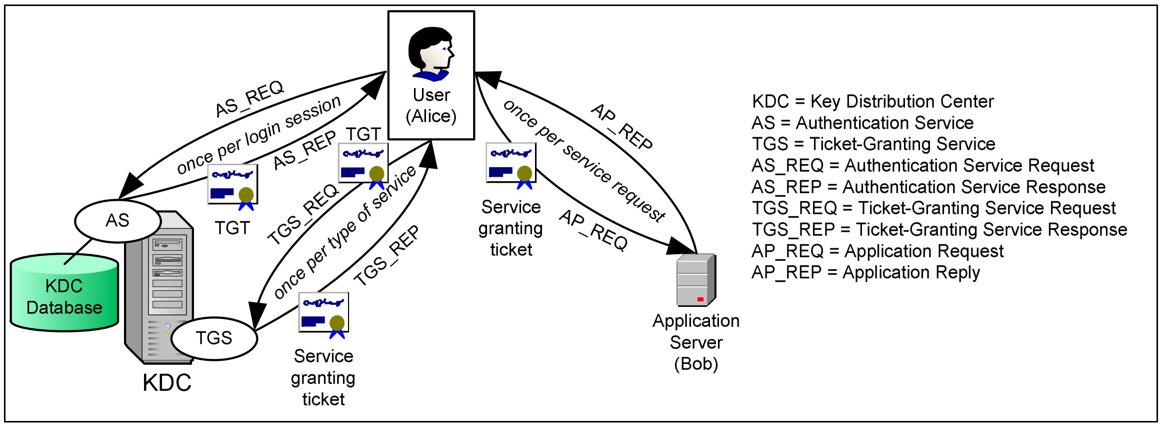

User (Alice)
AS_REQ
once per login session
AS_REP
TGT
AS
TGT
KDC Database
TGS_REQ
once per type of service
TGS_REP
TGS
KDC
Service granting ticket
Service granting ticket
AP_REP
once per service request
AP_REQ
Application Server (Bob)
KDC = Key Distribution Center
AS = Authentication Service
TGS = Ticket-Granting Service
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 Request
AP_REP = Application Reply

Figure 1 - Kerberos Overview

The Kerberos protocol illustrated in Figure 1 starts when user, Alice, logs into a client workstation and requests a service from application server, Bob. Alice obtains permission for that service with a service-granting ticket issued from the Ticket-Granting Service (TGS) running on the Kerberos server (which is called the KDC), but first Alice must prove her identity to and obtain a Ticket Granting Ticket (TGT) from the Authentication Service (AS) running on the Kerberos server. The TGT gives a user permission to request service tickets which grant access to application servers. The messages shown in Figure 1 are described in more detail below[1]:

AS_REQ:

- Alice requests a TGT from the AS
- message format (sent in the clear if preauthentication is not used):
  `UserID: Alice`

AS_REP:

- The AS verifies with the Key Distribution Center (KDC) database that Alice is authorized to request tickets, then (pending authorization) the AS generates and sends to Alice a session key (S_{Alice}) encrypted with Alice's secret key (K_{Alice}), and the TGT.
- message format:
  K_{Alice}`{use` S_{Alice} `with TGS},`
  `TGT:` K_{TGS}`{use` S_{Alice} `with Alice}`

  where

  K_{Alice}`{X}` is the encryption of `X` with Alice's secret key,
  S_{Alice} is the session key shared between the KDC and Alice, and
  K_{TGS}`{X}` is the encryption of `X` with the TGS's secret key.

TGS_REQ:

- After decrypting AS_REP and recovering S_{Alice}, Alice requests a service-granting ticket from the TGS, with her authenticator (which proves her identity) and her TGT.

[1] We follow the convention used by Microsoft in [5] for denoting message contents.

- message format:

  `“Alice is requesting Bob”,`
  `Authenticator:` S_{Alice}`{Alice, time`$_1$`},`
  `TGT:` K_{TGS}`{use` S_{Alice} `with Alice}`

TGS_REP:

- After verifying Alice’s authenticator, the KDC sends the service-granting ticket to Alice in a message encrypted with S_{Alice}.
- message format:

  S_{Alice}`{use` S_{AB} `with Bob},`
  `service ticket:` K_{Bob}`{use` S_{AB} `with Alice}`

  where

  S_{AB} is the session key shared between Alice and Bob, and
  K_{Bob} is Bob’s secret key.

AP_REQ:

- Once Alice recovers S_{AB} and the service ticket from TGS_REP, she encrypts her userID with S_{AB} to prove her identity to Bob, and sends that authenticator along with the service ticket to Bob. Optionally, she can set a flag to turn on mutual authentication so the server will subsequently prove it’s identity to her.
- message format:

  S_{AB}`{Alice, time`$_2$`},`
  `service ticket:` K_{Bob}`{use` S_{AB} `with Alice}`
  `mutual authentication flag: on/off`

AP_REP:

- If Alice requested mutual authentication, then Bob extracts `time`$_2$ from AP_REQ and returns it to the client encrypted using S_{AB}.
- message format:

  S_{AB}`{time`$_2$`}`

Subsequent messages carry information relating to the service Alice requested from Bob.

What are Kerberos Realms?

Kerberos *realms* represent a networked collection of client workstations, application servers, and a single master KDC which is delegated to the following responsibilities:

1. maintaining a database of matching user IDs and hashed passwords for registered Kerberos users,
2. maintaining shared secret keys with each registered application server,
3. maintaining shared secret keys with remote KDCs in other realms, and
4. propagating new or changed secret keys and database updates to slave KDCs[2].

[2] Slave KDCs are redundant copies of the master KDC, and increase the tolerance of a Kerberos environment to faults in the master KDC.

Typically, networks of client workstations and application servers under different administrative domains fall into different Kerberos realms. *Cross-realm* authentication is required when a user requests a service from an application server which belongs to a remote realm.

How can Public-key Cryptography improve Kerberos?

Scalability

PKC simplifies the distribution of keys in the Kerberos environment because secret-key cryptology (SKC) requires a separate secret key to be shared between every pair of users. So, for a Kerberos environment using secret-key based cryptography between `C` clients and `S` application servers, the total number of shared keys which must be maintained throughout the network is `C*S` session keys (shared between communicating clients and application servers) and `C+S` private keys (shared between each client or server and the KDC). Alternatively, PKC only requires users to share a single public key with each of their peers. So, the same network of `C` clients and `S` application servers using PKC enabled Kerberos must only maintain `C+S` shared (public) keys. The private keys associated with PKC do not add overhead to the key distribution process since they never need to be transmitted, and the shared public keys can be used for initial authentication as well as session encryption during service requests.

In general, a network of `n` users communicating with secret key cryptography will require a total of `n(n-1)/2` shared keys, while the same network of users using PKC will only require `n` shared (public) keys. Thus, the scalability of key distribution within the Kerberos framework can be improved with PKC.

Improved Security

Traditionally, Kerberos Authentication Servers reply to a user's TGT request with a ticket encrypted by that user's secret key. Therefore, the KDC must remember every user's secret key, and if an unauthorized individual gains even read-only access to the KDC's database, the secrecy of user keys is violated and the security of the KDC compromised. Since PKC enabled Kerberos uses public keys for encrypting TGTs, read-only access into a KDC's database would not compromise security at all. To violate the PKC security policy, an attacker must obtain write access to a public-key repository, such as a X.509 Certificate Authority directory. PKC improves Kerberos security because actively modifying public keys in the public-key infrastructure is much more difficult than passively reading secret-keys in traditional Kerberos databases.

Performance issues of PKC in Kerberos

Kerberos has traditionally been applied on a small enough scale that performance bottlenecks at KDCs do not occur, but recent developments like Microsoft's .NET Passport service are implementing Kerberos in large networks with many entities participating in the authentication process. Furthermore, the computational requirements of PKC are generally higher than those of SKC for the following reasons:

- The calculations involved during key generation and encryption and decryption routines are computationally expensive. For example, DES uses table lookups and XOR operations where as the public-key RSA algorithm uses exponentiation and multiplication for encryption and decryption. In fact, hardware implementations of RSA are about 1000 times slower than DES [8].
- PKCs generally require much larger keys than conventional SKC. Table 1 shows the recommended key lengths for various secret-key and public-key algorithms [8].

Table 1 – Recommended key lengths for various secret-key and public-key algorithms.

Secret-key algorithms	Key Length	Public-key algorithms	Key Length
DES	56 bits	RSA	1024 bits
Triple-DES	168 bits	ECC	1024 bits
Rijndael	256 bits	Tao Renji's Finite Automation algorithm	4152 bits

PKINIT and PKCROSS public-key extensions

A group of researchers from various companies in industry collaborated to design public-key extensions for Kerberos, and they recently published their specifications in the following Internet-Drafts:

1. Public-key Cryptography for Initial Authentication in Kerberos (PKINIT) [3]
2. Public-key Cryptography for Cross-Realm Authentication in Kerberos (PKCROSS) [4]

PKINIT

The Internet-draft specification for PKINIT has been adopted by Microsoft for PKC support in their Windows 2000 implementation of Kerberos [5]. This specification for public-key based initial authentication of a client to its local KDC is summarized in Figure 2.

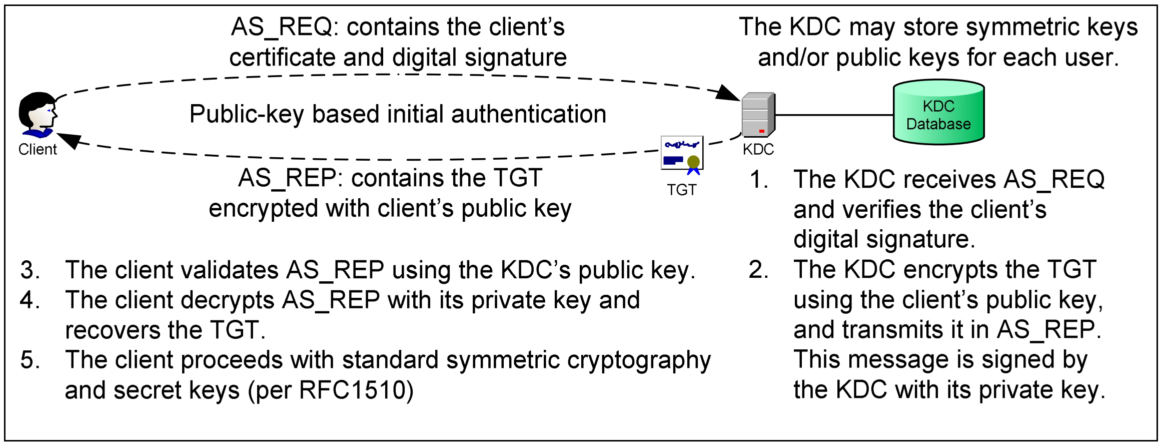

AS_REQ: contains the client's certificate and digital signature
The KDC may store symmetric keys and/or public keys for each user.
Public-key based initial authentication
Client
KDC
KDC Database
TGT
AS_REP: contains the TGT encrypted with client's public key
1. The KDC receives AS_REQ and verifies the client's digital signature.
2. The KDC encrypts the TGT using the client's public key, and transmits it in AS_REP. This message is signed by the KDC with its private key.
3. The client validates AS_REP using the KDC's public key.
4. The client decrypts AS_REP with its private key and recovers the TGT.
5. The client proceeds with standard symmetric cryptography and secret keys (per RFC1510)

Figure 2 - PKINIT Overview

PKCROSS

The primary benefits of PKCROSS involve "simplifying the administrative burden of maintaining cross-realm keys." [4]. In other words, it improves the scalability of Kerberos in large multi-realm networks where many application servers may be participating in the authentication process. This protocol is summarized in Figure 3.

The public-key extensions proposed in PKCROSS take place only between pairs of KDCs (i.e. KDC-to-KDC authentication), and are thus transparent to end-users requesting cross-realm tickets. This means the messages communicated between users and their local KDC during cross-realm authentication can almost exactly follow RFC 1510.

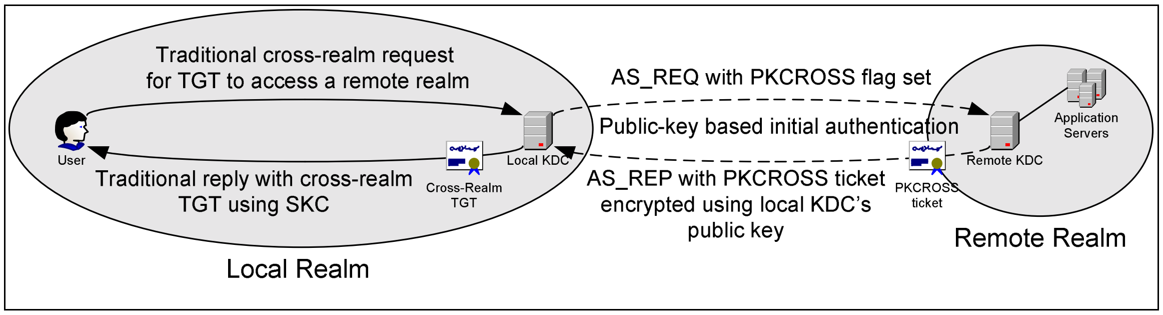

Traditional cross-realm request for TGT to access a remote realm
User
Local KDC
Cross-Realm TGT
Traditional reply with cross-realm TGT using SKC
Local Realm
AS_REQ with PKCROSS flag set
Public-key based initial authentication
AS_REP with PKCROSS ticket encrypted using local KDC's public key
PKCROSS ticket
Remote KDC
Application Servers
Remote Realm

Figure 3 - PKCROSS Overview

The PKCROSS ticket is used for local KDC to remote KDC authentication. The messages exchanged between the two KDCs closely follow the PKINIT specification, with the local KDC acting as the client. When the remote KDC issues a PKCROSS ticket to the local KDC, it trusts the local KDC to issue the remote realm TGT to its local client on behalf of the remote KDC.

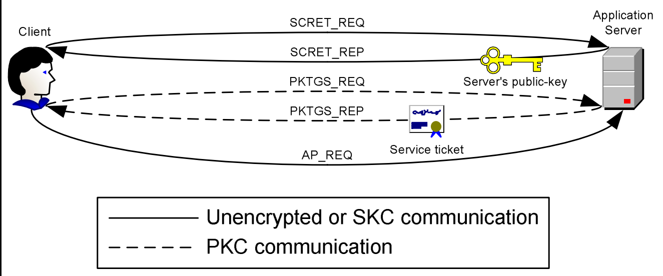
In summary, PKINIT facilitates public-key based authentication between local Kerberos clients and their local KDC, while PKCROSS extends the public-key capabilities to cross-realm KDC-to-KDC authentication. Thus, by using PKINIT and PKCROSS together, public-key based authentication can be integrated throughout the entire Kerberos environment.

Public-key based Kerberos for Distributed Authentication (PKDA)

A pair of researchers at Carnegie Mellon University embraced and extended the PKINIT and PKCROSS protocols to create a new protocol, called "Public-key based Kerberos for Distributed Authentication" (PKDA) [6]. It claims to offer enhanced privacy of Kerberos clients, as well as increased scalability and security within the Kerberos framework. Increased client-side privacy is achieved by simply "moving the client identity fields from unencrypted to encrypted portions" of the authentication messages. Increased scalability and security is attempted by moving authentication procedures away from centralized KDCs to between the individual clients and application servers on a network. Theoretically, this should reduce bottlenecks at KDCs during high volumes of authentication requests (increased scalability), and eliminate the single point of failure a KDC's centralized collection of keys imposes on the Kerberos environment (improved security). In practice, requiring lengthy public-key transactions between some clients and many application servers may not be anymore efficient than using public-key based authentication once with a KDC and faster secret-key cryptography for subsequent encryption with application servers.

The PKDA design differs from all other proposed public-key enhancements to Kerberos by completely removing the presence of a centralized KDC in the Kerberos framework. Using PKC for every service ticket request renders the symmetric key maintained in the KDC completely unnecessary, even for cross-realm authentication! In fact, since PKDA doesn't rely on the KDC for initial authentication, specialized cross realm authentication procedures are also unnecessary (as they can be identical to those procedures within a local realm).

The five messages illustrated in Figure 4 describe the PKDA exchanges which occur when an unauthenticated client requests a service from an application server.



SCRET_REQ: The client requests the application server's public-key.
SCRET_REP: The server replies with its public-key, signed by a trusted Certificate Authority
PKTGS_REQ: The client requests a session ticket to access the application server. This message is digitally signed with the client's private key, and encrypted with the server's public-key.
PKTGS_REP: The server returns a read-only service ticket to the client.
AP_REQ: The client requests a service from the application server, using the traditional secret-key procedures defined in RFC1510.

Figure 4 - PKDA Overview

PKDA accomplishes mutual client/server authentication in step 3 by digitally signing PKTGS_REQ with the client's private key and encrypting it with the server's public-key. The signature can only be verified with the client's public-key, and the PKTGS_REQ message can only be decrypted with the server's private key. The client is assured of the server's authenticity since only the server can decrypt the message to construct a valid response, and the server is assured of the client's authenticity by verifying the digital signature with the client's public-key.

The application server replies to the client in step 4 with a service ticket for the requested service. This ticket is encrypted with a key known only to the server, so the client can not modify it. Finally, the client continues in step 5 by accessing its requested service using the service ticket and the procedures defined in the standard secret-key based Kerberos specification.

PKCROSS and PKDA Performance Comparisons

Determining whether it is more efficient to authenticate to a central KDC than to individual application servers was the topic of a paper presented by Alan Harbitter and Daniel Menascé in the 2001 IEEE Symposium on Security and Privacy [7]. They modeled a multi-realm Kerberos environment in order to observe how the following parameters effected the response times of public-key based authentication transactions:

- Number of realms in the Kerberos environment
- Number of applications servers per realm
- Loads on applications servers and KDCs
- Network delay

These experiments demonstrated that even though PKDA requires fewer messages than PKCROSS during client authentication with remote application servers, PKCROSS achieves better cross-realm performance results than PKDA[3] for networks with two or more application servers in a remote realm. From these results, one can conclude that PKDA suffers from sending long public-key messages and performing expensive public-key calculations for authentication with multiple application servers. PKCROSS is more efficient because it only uses PKC once with the KDC and then uses more efficient SKC for subsequent communication with application servers. In other words, the cost of maintaining S secret keys outweighs the expense of performing S public key transactions between each client and S application servers. Thus, PKDA will not always achieve significantly higher scalability than other PKC proposals can.

Security issues of Public-Key Cryptography in Kerberos

The largest security issue associated with incorporating PKC in Kerberos involves trust management. PKC enhancements to Kerberos expand the scope of trusted entities to include everything in the public-key infrastructure (PKI), but assuming an already secured PKI may not always be valid. For example, weak public-key management opens a public-key cryptosystem to man-in-the-middle attacks where an attacker capable of modifying the traffic between two communicating users can achieve unauthorized decryption of their public-key encrypted messages. Fortunately, using the X.509 Certificate Authorities supported by PKINIT, PKCROSS, and PKDA can provide a trustworthy PKI without introducing a liability into the Kerberos trust model.

Another concern regards the implementation of public-key specifications and PKIs. The specifications are not simple, and deploying the enterprise-wide X.509

[3] Actually, their experiments used PKTAPP instead of PKDA, but PKTAPP is only a slight variation on the PKDA specification. PKTAPP was originally introduced as PKDA and follows its fundamental design approach of excluding KDCs from authentication procedures.

infrastructure can be particularly difficult. These complexities, while hard to overcome, increase the likelihood of unreliable implementations.

Conclusion

The three public-key based protocols we have summarized are PKINIT, PKCROSS, and PKDA. Each of these three protocols add public-key cryptography (PKC) support into different stages of the Kerberos framework, but they all attempt to improve Kerberos scalability and security by simplifying key management and utilizing trustworthy public-key infrastructures like X.509. PKINIT is the core specification to PKCROSS, and together they couple to define a public-key based authentication solution across multi-realm Kerberos networks. PKDA makes more fundamental changes to the Kerberos standard in an attempt to achieve greater improvements in scalability, security, and client privacy issues within the Kerberos framework. However, research by Harbitter and Menascé [7] concluded that the complexity of public-key computations and length of its messages makes PKDA generally less efficient than the simpler PKCROSS protocol for cross-realm (KDC-to-KDC) authentication. So, while PKDA may achieve better security and privacy characteristics than other PKC proposals, it offers no greater improvement in scalability.

PKC enhancements to the traditional Kerberos standard incorporate a public-key infrastructure (PKI) into the scope of underlying systems trusted by Kerberos. Consequently, any vulnerability in the PKI will inherently degrade the trustworthiness of Kerberos. Although no security flaws have been associated with the PKINIT, PKCROSS, or PKDA specifications, nothing can be said about their reliability until they have been implemented together with a PKI and applied for wide-spread public review. The difficulties associated with implementing these public-key protocols on top of an enterprise-wide PKI make this kind of qualitative analysis the critical "next step" in adding public-key support to Kerberos.

Bibliography

[1] Massachusetts Institute of Technology. *Kerberos: The Network Authentication Protocol.* http://web.mit.edu/kerberos/www/, 2000.

[2] Kohl, J., *The Kerberos Network Authentication Service (V5)*, C. Neuman, Editor. 1993: http://www.ietf.org/rfc/rfc1510.txt

[3] Tung, B., *et al.*, *Public Key Cryptography for Initial Authentication in Kerberos*, May 25, 2002 (expiration): http://www.ietf.org/internet-drafts/draft-ietf-cat-kerberos-pk-init-15.txt

[4] Tung, B., *et al.*, *Public Key Cryptography for Cross-Realm Authentication in Kerberos*, May 8, 2002 (expiration): http://www.ietf.org/internet-drafts/draft-ietf-cat-kerberos-pk-cross-08.txt

[5] Microsoft TechNet. *Windows 2000 Kerberos Authentication.* http://www.microsoft.com/technet/prodtechnol/windows2000serv/deploy/confeat/kerberos.asp, 1999.

[6] Sirbu, M., Chung-I Chuang, J. *Distributed Authentication in Kerberos Using Public Key Cryptography*. Symposium on Network and Distributed System Security, 1997. San Diego, California: IEEE Computer Society Press.

[7] Harbitter, A., Menascé, D. *Performance of Public-Key-Enabled Kerberos Authentication in Large Networks*. 2001 IEEE Symposium on Security and Privacy Proceedings, IEEE Computer Society Press.

[8] Schneier, B. Applied Cryptography. New York: Wiley, 1996.

[9] ITU-T (formerly CCITT) Information technology – Open Systems Interconnection – The Directory: Authentication Framework Recommendation X.509 ISO/IEC 9594-8

About the Author

Ian Downard is a graduate student at the University of Missouri – Rolla (UMR) working towards a master's degree in Computer Engineering. He received a BS in Computer Engineering at UMR in 2000. As a graduate student, Ian has taught a Computer Science course on algorithms and researched computer network security. Ian has held internships at the Naval Research Laboratory and is partially funded by a fellowship from the US Department of Education. You can reach him at http://www.umr.edu/~itd.