Comparative study of various PKINIT methods used in Advanced Kerberos

*Ms. Shital S. Thorat,** Prof. H. K. Sawant*** Mrs. Sarita S. Gaikwad ****Prof. G. T. Chavan

* MTech(IT)BVCOEP, ** Ap. Dept. Comp Dept.BVCOEP, ***ME(Comp CN), SCOE **** Ap. Comp. Dept. SCOE

Abstract - **Traditional authentication method is password, but it cannot resist dictionary and playback attack. Thus, applications, which send an unencrypted password over the network, are extremely vulnerable. Kerberos can be used as a solution to these network security problems. The Kerberos protocol with public key cryptography may help client to prove its identity to a server (and vice-versa) across an insecure network connection. This paper shows comparative study of various PKINIT methods used in Kerberos with their results.**

Keywords - *Authentication Server, Ticket-Granting Server, Ticket Granting Ticket, Ticket-Granting Service Request, Ticket-Granting Service Response, Authentication Service Request, Authentication Service Response, Application Request, Application Reply, Public key cryptography for initial authentication, Extensible Pre-Authentication in Kerberos,Key Distribution Center, Kerberos Authentication Server. Data Encryption Standard, Certificate Revocation List, Online Certificate Status Protocol Services.*

I. EVOLUTION OF KERBEROS

The name Kerberos comes from Greek mythology; 'Cerberos' was the three-headed dog that guarded the entrance to Hades. Kerberos is a network authentication protocol developed by MIT (Massachusetts Institute of Technology) as part of Project Athena, which started in 1983 when MIT decided to integrate network computers as part of its campus curriculum[1].

The goals of Athena were the integration of a SSO (Single Sign-on), networked file systems, a unified graphical environment, and a naming convention service. Kerberos has since evolved into a strategic security standard that provides secure authentication services to users, applications, and network devices, which eliminates the threats caused by passwords being stored or transmitted across the network. Additionally, Kerberos provides data integrity to ensure messages are not tampered with on the network and message privacy (encryption) to ensure messages are not visible to eavesdroppers on the network[3].

The Kerberos model is partly based on trusted third-party authentication protocol. Versions one through three never reached outside MIT, but version 4 was (and still is) quite popular, especially in the academic community. It is also used in commercial products like the AFS file system.

Following points are discussed here:

- Traditional authentication service exchange
- Working principle of Kerberos
- Authentication process
- Existing Methodology: Kerberos Products on HP-UX

A. *The Traditional Authentication Service Exchange*

Kerberos is designed to eliminate the need for users to demonstrate possession of private or secret information (the password). Instead, Kerberos disseminates this information. Kerberos Server lets entities authenticate themselves, without transmitting their passwords in clear text over the network.

The abstract structure of the messages in the traditional (non-PKINIT) AS exchange is given in Figure 1[3]. A client C generates a fresh nonce n1 and sends it, together with own name and the name T of TGS for whom client desires a TGT, to the KAS. The KAS responds by generating a fresh key AK for use between the client and the TGS. This key is sent back to the client, along with the nonce (n_1) from the request and other data, encrypted under a long-term key k_C shared between C and the KAS[3]; this long-term key is usually derived from the user's password. We write $\{m\}_k$ for the encryption of m with symmetric key k.

This is the only time that this long-term key is used in a standard Kerberos run because later exchanges use freshly generated keys. AK is also included in the TGT, sent alongside the message encrypted for the client. The TGT is encrypted under a long-term key k_T shared between the KAS and the TGS named in the request.

These encrypted messages are accompanied by the client's name and other data. Once the client has received this reply, she may undertake the Ticket-Granting exchange.

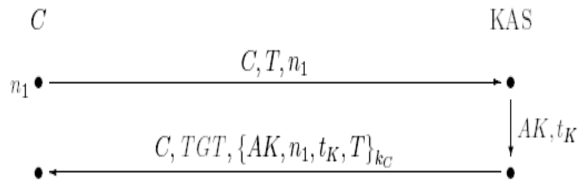


Figure 1. Message Flow in the Traditional AS xchange where TGT = $\{AK, C, t_T\}\ k_T$

B. *Working Principle of Kerberos*

Kerberos uses secret-key cryptography, which is used to identify the entities communicating over networks. Kerberos is based on the concept of a trusted third party that performs secure verification of users and services. In the Kerberos protocol, this trusted third party is called the key distribution center (KDC)[5].

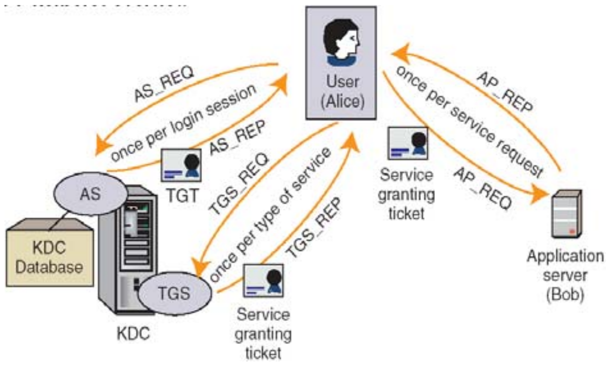


Figure 2. Kerberos Overview

Kerberos is used to verify that users and the network services they use are really who and what they claim to be. To accomplish this, a trusted Kerberos Server issues tickets to users. These tickets, which have a limited lifespan, are stored in a user's credential cache and can be used in place of the standard username-and-password authentication mechanism. The ticket can then be embedded in virtually any other network protocol, thereby letting the processes implementing that protocol to be sure about the identity of the principals involved.

The Kerberos credential scheme includes the Single Sign On (SSO)[2] concept. Secure authentication is based on previously established initial credentials, which eliminates the need to re-key a password on multiple occasions.

A Kerberos server consists of the following elements:

- **Realm** - a user-defined administrative boundary.
- **Key Distribution Center (KDC)** - the heart of the Kerberos realm. It provides Kerberos authentication services by issuing encrypted tickets that require secret keys to decode.
- **Principal** - a unique name for a user or service stored in a KDC.
- **Tickets** - records that help a client authenticate to a server

Kerberos *realms* represent a networked collection of client workstations, application servers, and a single master Key Distribution Center (KDC) with the following responsibilities[3]:

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 KDC's in other realms
4. Propagating new or changed secret keys and database updates to slave KDC's. (Slave KDC's are redundant copies of the master KDC. They 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 that resides in a remote realm.

The client process interacts with three types of principals (KDC server, Kerberos security principal, Validating server) during the three rounds of Kerberos 5 (with or without PKINIT). The client's goal is first to authenticate himself to various application servers (e.g., email, file, and print servers). This is done by first obtaining credentials, called the "ticket-granting ticket" (TGT), from a "Kerberos Authentication Server" (KAS) and then by presenting these credentials to a "Ticket-Granting Server" (TGS) in order to obtain a "service ticket" (ST), which is the credential that the client finally presents to the application servers in order to authenticate himself A TGT might be valid for a day, and may be used to obtain several ST's for many different application servers from the TGS, while a single ST might be valid for a few minutes and is used for a single application server. The KAS and the TGS are altogether known as the "Key Distribution Center" (KDC).

The client's interactions with the KAS, TGS, and application servers are called the Authentication Service (AS), Ticket-Granting (TG), and Client-Server (CS) exchanges, respectively. The focus of this work will be the AS exchange, as PKINIT does not alter the remaining parts of Kerberos[4].

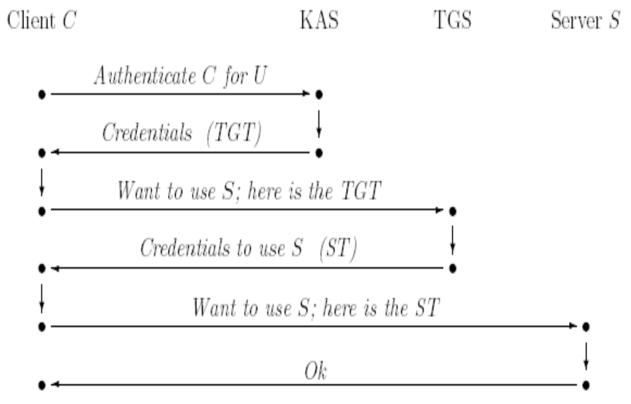

Client C
KAS
TGS
Server S
Authenticate C for U
Credentials (TGT)
Want to use S; here is the TGT
Credentials to use S (ST)
Want to use S; here is the ST
Ok

Figure 3. An Overview of Kerberos Authentication[3]

C. Authentication Process

The following steps describe how a client and a server authenticate each other using Kerberos. The step numbers match with the numbered arrows in figure below[9].

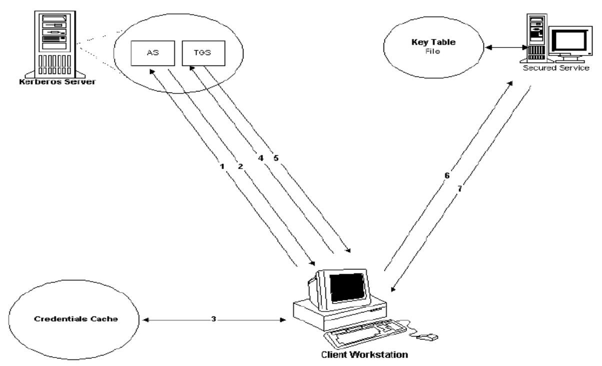


Figure 4. The Kerberos Authentication Protocol[9]

Step 1: The user begins to use a Kerberized application by entering the user name and password. Optionally, the user can request for specific ticket flags and specify the key type to be used for constructing the secret key. The user can also accept the default, configured for the client. The user sends the following information to the Authentication Service (AS) to obtain credentials[9]:

- **Client**, **Server**, **T**, **N**; where
- **Client** indicates the user name, also referred to as the principal name. **Server** indicates the Application Server
- **T** indicates the time stamp and
- **N** indicates nonce

Step 2: If the AS can decrypt the message successfully, it issues a temporary session key, which is encrypted with the user's secret key (a key derived from the user password, which is stored in the KDC), and a TGT encrypted with the TGS's secret key. The TGT contains the name of the user and a copy of the session key (a randomly generated temporary encryption key) to be used by the user and the Server for any subsequent communication.

Step 3: The user decrypts the session key. The TGT and the session key are stashed in the user's credential cache. The credentials are used to obtain tickets for each network service the principal wants to access. This protocol exchange has two important features:

- The authentication scheme does not require that the password be sent across the network, either in encrypted form or in clear text.
- The client (or any other user) cannot view or modify the contents of the TGT.

Step 4: To obtain access to a secured network service such as rlogin, rsh, rcp, ftp, or telnet, the requesting client application uses the previously obtained TGT in a dialogue with the TGS to obtain a service ticket. The protocol is the same as used while obtaining the TGT, except that the messages contain the name of the server and a copy of the previously obtained TGT.

Step 5: The TGS returns a new service ticket that the application client can use to authenticate the service.

Step 6: The application client tries to authenticate to the service on the application server using the service ticket obtained from the TGS. The secure application validates the service ticket using the server's service key present in the key tab file. Using this service key, the server decrypts the authenticator and verifies the identity of the user. It also verifies that the user's service ticket has not expired. If the user does not have a valid service ticket, then the server will return an appropriate error code to the client.

Step 7: (Optional) At the client's request, the application server can also return the time stamp the client sent encrypted in the session key. This ensures a mutual authentication between the client and the application server.

II. INTRODUCTION TO PKINIT

Kerberos is a successful, widely deployed single sign-on protocol that is designed to authenticate clients to multiple networked services, e.g., remote hosts, file servers, or print spoolers. Kerberos 5, the most recent version, is available for all major operating systems. Kerberos 5 continues to evolve as new functionalities are added to the basic protocol[1].

One of these extensions, known as Public Key Cryptography for Initial Authentication (PKINIT), which modifies the basic protocol to allow public-key authentication and in the process adds considerable complexity to the protocol[3]. Here we report a protocol-level attack on PKINIT and discuss the constructive process of fixing it. Kerberos is based on conventional cryptography. That is, it relies on symmetrical cryptographic algorithms that use the same key for encryption as for decryption[9].

PKINIT is intended to add flexibility, security and administrative convenience by replacing this static shared secret with two pairs of public/private keys, one assigned to the KDC and one belonging to the user[2].

Following points are discussed here -

- Security constraints in existing system
- Public key cryptography for initial authentication in Kerberos

A. Kerberos with Public Key Cryptography

Kerberos authentication is based on use of *symmetric* cryptography. The implication is that symmetric encryption keys must be exchanged between parties as a precondition to being able to perform authentication. For human users, passwords are used to derive symmetric encryption keys for authentication and exchange of subsequent session keys.

In practice, every user and service principal must go through an initial setup process where passwords or symmetric encryption keys are exchanged with a KDC. This can be done securely with proper procedures by an administrative[2].

Furthermore, both parties can compromise the shared secret key. Changing passwords and keys can also be a burden for users and system administrators.

The appeal of asymmetric cryptography is that there are two keys, where anything encrypted with one key can only be decrypted with the one-and-only corresponding key[2]. When asymmetric "key pairs" are generated, it is common practice to treat one as a "public key" that can be freely shared with other parties while the other key in the pair is kept as a "private key" that should not be shared with any other parties. This significantly simplifies key sharing, since the public key can be freely exchanged between parties provided the private key is not disclosed.

The primary concern with distributing public keys is determining whether or not a public key really belongs to a specific party. This is generally solved through use of public key certificates, which include the name of the party (subject) and their public key in a document that is digitally signed by some "Certification Authority" or "CA." Any party that receives a certificate can validate the CA's signature to confirm that the public key really belongs to the subject named in the certificate.

Kerberos has been extended to take advantage of public key technologies, primarily for initial authentication of users requesting TGT's, though other extensions have been proposed for including public key certificates in Kerberos tickets.

B. PKINIT with Kerberos

Public Key Cryptography for initial authentication (PKINIT) specifies extensions to KDC Authentication Services that allow clients (users) to be initially authenticated by presenting their public key certificates instead of using previously shared secret passwords[1]. Only the initial authentication procedure is changed; all other Kerberos interactions remain the same.

In particular, clients continue to receive Ticket Granting Tickets (TGTs) that will be used to subsequently request tickets for services, and services continue to authenticate users via tickets issued by KDCs. A service has no need to support public key cryptography, or even know how the client initially authenticated to the KDC[1].

From an application perspective, client and service interactions are exactly as before. An important consequence of using PKINIT is that users can more easily be given an account in a realm they have never visited before[2]. If an employee or contractor with a trusted certificate moves to a new part of an organization, their account can be waiting for them with no need to register a new password.

Furthermore, use of public key certificates helps to moderate risks with weak user passwords, since the user's private key is used during initial authentication instead of a password. A user may still enter a password, but only on their local workstation or device to unlock the private key. Password changes tend to be handled at the user device level, and do not have to be coordinated with other systems.

At least theoretically, using PKINIT, user principals no longer need to be registered in the KDC database before initial authentication, since the KDC does not need to look up the corresponding shared secret for a user with a certificate[3]. Service principals must still be registered in the KDC database along with a shared symmetric encryption key. A KDC database could be much smaller.

However, in practice, users still need to be registered to store policy and other information. PKINIT does not reduce the need for this information. With PKINIT, the administrative overhead for maintaining the KDC database should be reduced, and user password changes would no longer need to be supported for users with certificates.

Another benefit is that certificates can be associated with hardware cryptographic tokens, including "smart cards" and USB devices with embedded "smart chips." Some biometric authentication devices can also be used with certificates, but where the biometric device is used to unlock access to the user's private key. Of course, these benefits come at the expense of deploying a Public Key Infrastructure (PKI) that includes Certification Authorities (CAs), certificate request and issuing services, certificate revocation procedures, status checking services, and directories for retrieving certificates for specific users or other principals. However, PKI has become an integral part of several vendor platforms, and it is widely available.

The KDC database can store self-issued certificates associated with user principals. This avoids reliance on a CA, since the KDC can trust a public key in its database, and no password secret is needed for the user. However, other advantages such as easy migration between realms and support for applications beyond Kerberos tend to require a PKI[4].

One concern with certificates used in authentication is that they tend to have relatively long lifetimes on the order of months to even a couple of years. This leads to greater exposures when private keys are compromised, or a user's certificate needs to be revoked for some cause. This means that a KDC might allow a user with a revoked certificate to authenticate, and subsequently access application services.

There are two approaches for mitigating this risk[5]:

1. Distribution of Certificate Revocation Lists (CRLs)
2. Use of Online Certificate Status Protocol (OCSP) services .

Use of OCSP services is particularly attractive, as it allows certificate status to be checked in real time as part of the initial authentication process. PKINIT can be further extended to integrate OCSP checks during Kerberos initial authentication.

Scalability*:* Similarly, public-key cryptography simplifies the registration of new keys into the key distribution center (KDC) since public keys can be safely communicated over a remote connection to the KDC database. PKC improves the scalability of KDC maintenance by affording administrators the freedom of remote access without the burden of a priori security.

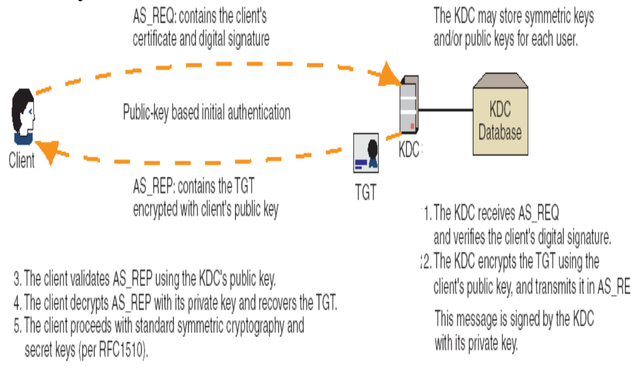

AS_REQ: contains the client's certificate and digital signature
The KDC may store symmetric keys and/or public keys for each user.
Public-key based initial authentication
Client
KDC
KDC Database
TGT
AS_REP: contains the TGT encrypted with client's public key
1. The KDC receives AS_REQ and verifies the client's digital signature.
2. The KDC encrypts the TGT using the client's public key, and transmits it in AS_RE
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 5. PKINIT Overview

Improved security: Public-key cryptography improves Kerberos security because modifying public keys in the public-key infrastructure is much more difficult than passively reading secret-keys in traditional Kerberos databases.

Performance issues: While public-key cryptography can improve scalability and security aspects of Kerberos, it can also harm performance. Traditionally, Kerberos has been applied on a small enough scale that performance bottlenecks at key distribution centers did not occur. But recent systems are implementing Kerberos in very large networks with many entities participating in the authentication process.

Furthermore, the computational requirements of public-key cryptography are generally higher than for secret-key cryptography for the following reasons:

1. The calculations involved during key generation and encryption and decryption routines are computationally expensive. For example, DES uses table lookups and XOR operations whereas the public-key RSA (Rivest, Shamir, Adleman) algorithm uses exponentiation and multiplication for encryption and decryption. In fact, hardware implementations of RSA are about 1000 times slower than DES.
2. Public-key cryptography generally requires much larger keys than conventional secret-key cryptography.

For these reasons, most proposals to include PKC in Kerberos attempt to minimize the number of public-key computations that occur.

C. Public-key cryptography for Cross- Realm Authentication in Kerberos (PKCROSS)

The primary benefits of PKCROSS[5] involve "simplifying the administrative burden of maintaining cross-realm keys," 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 6.

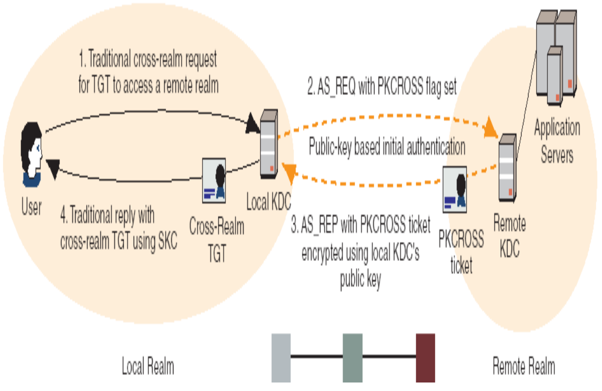

1. Traditional cross-realm request for TGT to access a remote realm
2. AS_REQ with PKCROSS flag set
Application Servers
Public-key based initial authentication
User
4. Traditional reply with cross-realm TGT using SKC
Cross-Realm TGT
Local KDC
3. AS_REP with PKCROSS ticket encrypted using local KDC's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

Figure 6. PKCROSS overview

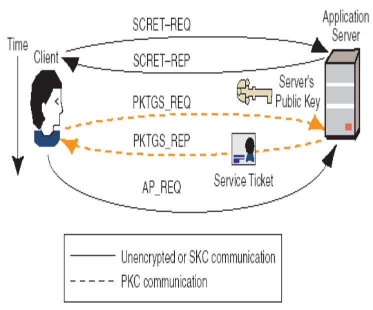
The public-key extensions proposed in PKCROSS take place only between pairs of key distribution centers (i.e. KDC-to-KDC authentication). They are, thus, transparent to end-users requesting cross-realm tickets.

The PKCROSS ticket is used to achieve mutual authentication between the local key distribution center and the remote key distribution center. The messages exchanged between the two key distribution centers closely follow the PKINIT specification, with the local key distribution center acting as the client. When the remote key distribution center issues a PKCROSS ticket to the local key distribution center, it trusts the local key distribution center to issue the remote realm TGT to its local client on behalf of the remote key distribution center.

In summary, PKINIT facilitates public- key based authentication between local Kerberos clients and their local key distribution center; 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 framework.

D Public-key based Kerberos for Distributed Authentication (PKDA)

The PKINIT and PKCROSS protocols are embraced and extended to create a new protocol, called "Public-key based Kerberos for Distributed Authentication" (PKDA)[5]. It provides enhanced privacy for Kerberos clients, as well as increased scalability and security within the Kerberos framework. Client-side privacy is increased by simply "moving the client identity fields from unencrypted to encrypted portions" of the authentication messages.



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: Ther 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 7 PKDA overview

Increased scalability and security is attempted by moving authentication procedures away from centralized key distribution centers to between the individual clients and application servers on a network.
It also eliminate the single point of failure a key distribution center's centralized collection of keys imposes on the Kerberos environment (improved security).
The PKDA design differs from all other proposed public-key enhancements to Kerberos by completely avoiding the centralized key distribution center in the Kerberos framework. Using public-key cryptography for every service ticket request renders the symmetric key maintained in the key distribution center completely unnecessary. This includes cross-realm authentication.
In fact, since PKDA doesn't rely on the key distribution center 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 Fig. 4 describe the PKDA exchanges that occur when an unauthenticated client requests a service from an application server.

III. RESULTS AND MEASURES

A. *Results*

All administrators are familiar with the problems Kerberos was designed to mitigate. Those problems include, password sniffing, password filename/database stealing, and the high level of effort necessary to maintain a large number of account databases.
A properly deployed Kerberos Infrastructure will help you address these problems. It will make your enterprise more secure.
Use of Kerberos will prevent plaintext passwords from being transmitted over the network. The Kerberos system will also centralize your username and password information which will make it easier to maintain and manage this data.
Finally, Kerberos will also prevent you from having to store password information locally on a machine, whether it is a workstation or server, thereby reducing the likelihood that a single machine compromise will result in additional compromises.

In a large enterprise, the benefits of Kerberos will translate into reduced administration costs through easier account and password management and through improved network security. In a smaller environment, scalable authentication infrastructure and improved network security are the clear benefits.
Some of the dangerous attack found in public-key encryption mode is given below:

1. Message Flow - Figure 8 shows the AS exchange message flow in the attack[3]. The client C sends a request to the KAS K with its own credentials, which is intercepted by the attacker I. Attacker constructs his own request message using the parameters from C's message. This allows the attacker to decrypt this part of the message using his private key, learn the key k, and use this to learn the key AK and send request to KAS. Now KAS view this request as C receives it. In turn, KAS provide service to this request. Attacker I modify the data received from KAS and send to client C, who assumes that the received data is send by KAS[7].

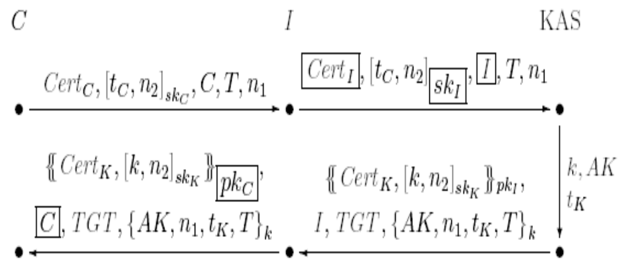

C
I
KAS
$Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1$
$\boxed{Cert_I}, [t_C, n_2]_{\boxed{sk_I}}, \boxed{I}, T, n_1$
k, AK
t_K
$\{\!\{Cert_K, [k, n_2]_{sk_K}\}\!\}_{\boxed{pk_C}},$
$\boxed{C}, TGT, \{AK, n_1, t_K, T\}_k$
$\{\!\{Cert_K, [k, n_2]_{sk_K}\}\!\}_{pk_I},$
$I, TGT, \{AK, n_1, t_K, T\}_k$

Figure 8. Message Flow in the Man-In-The-Middle Attack on PKINIT-26,[3]

where TGT = $\{AK, I, t_K\}k_T$

2. Attacker impersonate Servers - The attacker may intercept C's requests in the TG and CS exchanges and impersonate the involved servers rather than forwarding altered messages to them.

3. Attacker Observes Traffic - Once the attacker learns AK in the AS exchange, he may either mediate C's interactions with the various servers (essentially logging in as I while leaking data to C so user believes that he has logged in) while observing this traffic or simply impersonate the servers in the later exchanges.

B. *Solution for Attack*

The attack outlined in section 4.1 was possible because the two messages constitute, then the current version of PKINIT (PKINIT 26) was inefficient to handle it. More precisely, the attack shows that, although a client can link a received response to a previous request, user cannot be sure that the KAS generated the key AK and the ticket granting ticket TGT appearing in this response for the particular user. In fact, the only evidence of the principal for whom the KAS generated these credentials appears inside the TGT, which is unclear to user.

This suggests one approach to making PKINIT protected to this attack, namely to require the KAS to include the identity of this principal in a component of the response that is integrity protected and that the client can verify. An obvious mechanism is the sub message signed by the KAS in the reply. In PKINIT-27 (and subsequent versions), whenever a client C processes an AS reply containing server generated public-key credentials, the KAS previously produced such credentials for C.

1. Abstract Fix - Because of the attack the client cannot verify that the received credentials (the TGT and the key AK) were generated for him. This problem can be fixed by including the KAS include C's name in the reply in such a way that it cannot be modified whole route and that C can check it[5].

With this abstract fix in place, the PKINIT exchange in public-key encryption mode is depicted in Figure 9, where box highlights the modification with respect to PKINIT-26[5].

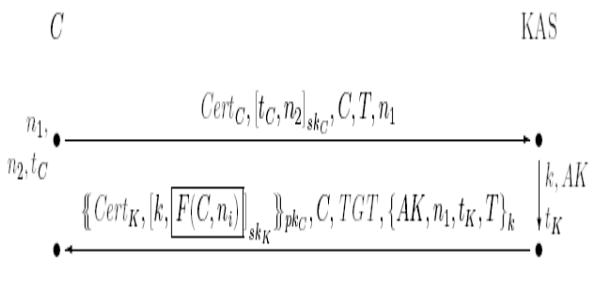

C
KAS
n_1, n_2, t_C
$Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1$
k, AK, t_K
$\{Cert_K, [k, \boxed{F(C, n_i)}]_{sk_K}\}_{pk_C}, C, TGT, \{AK, n_1, t_K, T\}_k$

Figure 9. Abstract fix of PKINIT[3]

Now the client can verify the KAS received credentials for him and not for another attacker. In fact, an honest KAS will produce the signature ([k, F(C, n_i)]sk_K) only in response to a request from C.

2. Solution Adopted In PKINIT-27 - In PKINIT-27, clients or C's name is included in the signed portion of the reply and also replaces the nonce n2 there with a keyed hash ("checksum" in Kerberos terminology) taken over the client's entire request. This approach also overcomes the attack. The IETF Kerberos Working Group decided to include the checksum based approach in PKINIT-27[3]. The message flow of this version of PKINIT is displayed in Figure 10.

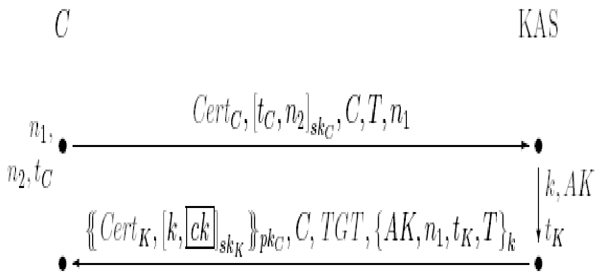

C
KAS
n_1, n_2, t_C
$Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1$
k, AK, t_K
$\{Cert_K, [k, \boxed{ck}]_{sk_K}\}_{pk_C}, C, TGT, \{AK, n_1, t_K, T\}_k$

Figure 10. Fix of PKINIT adopted in version 27[3]

C. *Measures*

Single point of failure: It requires continuous availability of a central server. When the Kerberos server is down, no one can log in. This can be mitigated by using multiple Kerberos servers and fallback authentication mechanisms. Kerberos requires the clocks of the involved hosts to be synchronized. The tickets have a time availability period and if the host clock is not synchronized with the Kerberos server clock, the authentication will fail. The default configuration requires that clock times are no more than 10 minutes apart. In practice Network Time Protocol daemons are usually used to keep the host clocks synchronized. The administration protocol is not standardized and differs between server implementations. Since the secret keys for all users are stored on the central server, a compromise of that server will compromise all users' secret keys. A compromised client will compromise the user's password

Together, the PKINIT and PKCROSS specifications 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. The complexity of public-key computations and length of its messages makes PKDA generally less efficient than the simpler PKCROSS protocol for cross-realm (key distribution center-to-key distribution center) authentication. So, while PKDA may achieve better security and privacy characteristics than other public key cryptography proposals, it offers no greater improvement in scalability.
Public-key cryptography enhancements to the traditional Kerberos standard incorporate a public-key infrastructure

into the scope of underlying systems trusted by Kerberos. Therefore, any weakness in the public-key infrastructure will inherently degrade the reliability of Kerberos.

Although no security flaws have been associated with the PKINIT, PKCROSS or PKDA specifications, nothing can be concluded about their reliability until they have been implemented together with a public-key infrastructure and applied for widespread public review.

IV. CONCLUSIONS

Adding Kerberos to a network can increase the overall security available to the users and administrators of that network. Remote sessions can be securely authenticated and encrypted. In addition, Kerberos allows the user and service principal's database to be managed securely from any machine that supports the Kerberos protocol.

The public-key based protocols—PKINIT, PKCROSS, and PKDA—add public-key cryptography support at different stages of the Kerberos framework. However, they all attempt to improve Kerberos scalability and security by simplifying key management and utilizing trustworthy public-key infrastructures.

REFERENCES

[1] Boldyreva, A. Kumar, V. Sch. of Comput. Sci., Georgia Inst. of Technol., Atlanta, GA: Extended Abstract: Provable-Security Analysis of Authenticated Encryption in Kerberos **:** IEEE Symposium on Security and Privacy, 2007. SP '07.

[2] Boldyreva, A. Kumar, V. Sch. of Comput. Sci., Georgia Inst. of Technol., Atlanta, GA: Extended Abstract: Provable-Security Analysis of Authenticated Encryption in Kerberos This paper appears in **:** Security and Privacy, 2007. SP '07. IEEE Symposium on Publication Date: 20-23 May 2007

[3] Cervesatol, A. D. Jaggard, A. Scedrov, J.-K. Tsay, and C. Walstad, Tulane University: Breaking and Fixing Public-Key Kerberos. IEEE Communications (2007)

[4] Downnard, I. Public-key cryptography extensions into Kerberos Potentials, IEEE Publication Date: Dec 2002-Jan 2003 Volume: 21

[5] El-Hadidi, M.T. Hegazi, N.H. Aslan, H.K. Dept. of Electron. & Electr. Commun, Cairo Univ., Egypt; Performance analysis of the Kerberos protocol in a distributed environmentThis paper appears in: Computers and Communications, 1997. Proceedings., Second IEEE Symposium on Publication Date: 1-3 July 1997 On page(s): 235 – 239

[6] Hellewell, P.L. van der Horst, T.W. Seamons, K.E. Brigham Young Univ., Provo: Extensible Pre-authentication Kerberos**:** Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual

[7] Ke Jia , Xiaojun Chen Guanghua Xu : The improved Public Key Encryption Algorithm of Kerberos Protocol based on Braid Groups,[2008] IEEE Communications

[8] Zrelli, S. Medeni, T. Shinoda, Y. Japan Adv. Inst. of Sci. & Technol., Nomi **:** Improving Kerberos Security System for Cross-Realm Collaborative Interactions Research, Innovation and Vision for the Future, 2007 IEEE

[9] Alan H. Harbitter, Daniel A. MenascC "Performance of Public-Key-Enabled Kerberos Authentication in Large Networks" 1081-601 1/01 $10.00 *0* 2001 IEEE