Kerberos protocol: an overview

Distributed Systems Fall 2002

Carlo Baliello
Faculty of Mathematical, Physical and Natural Science
Università degli Studi di Udine, Italy
790311-P537

Alessandro Basso
Faculty of Mathematical, Physical and Natural Science
Università degli Studi di Torino, Italy
780608-P217

Cinzia Di Giusto
Faculty of Mathematical, Physical and Natural Science
Università degli Studi di Udine, Italy
791017-P425

Hassan Khalil
École Nationale Superieure des Télécommunications de retagne, France
771218-P390

Daniel Machancoses
Facultad de Informática
Universidad Politécnica de Valencia – España
751001-P179

ABSTRACT

The Kerberos Authentication Service, developed at MIT, provides a trusted third-party authentication to verify users' identity. Here it is presented an overview of this protocol. The article can be logically divided in two parts, the first one describes the protocol, in the perspectives of the client and the server, focusing on how Kerberos achieve authentication. It is also given an idea of which are its limitations. The second part, instead, deals with practical arguments concerning Kerberos: it goes deep in some applications of Kerberos at two different levels: Cisco and the Operative System Windows 2000; and after that some results about performance are presented.

1. Introduction

Authentication is the process of proving one's identity to someone else. As humans, we authenticate each other in many ways: We recognize each other's faces when we meet, we recognize each other's voices on the telephone.

An authentication protocol would run before the two communicating parties in the system run some other protocol. The authentication protocol first establishes the identity of the parties to each other's satisfaction; only after authentication do the parties get down to the work at hand. It is a fundamental building block for a secure networked environment.

Kerberos [1] is an authentication service developed at MIT (Massachusetts Institute of Technology).that uses symmetric key encryption techniques and a key distribution centre; it is an add-system that can be used with existing network.

Kerberos provides a means of verifying the identities of principals on an open (unprotected) network. This is accomplished without relying on authentication by the host operating system, without basing trust on host address, without requiring physical security of all the hosts on the network, and under the assumption that packets travelling along the network can be read, modified, and inserted at will.

Kerberos performs authentication under these conditions as a trusted third party authentication service by using conventional cryptography. It trusted in the sense that each of its clients believes Kerberos's judgement as to the identity of each of its other clients to be accurate.

The problem that Kerberos addresses is this: a distributed system in which users at workstations wish to access services on servers distributed throughout the network. We would like for servers to be able to restricted access to authorized users and to be able to authenticate requests for service. In this system the following three threats exist:

1. A user may gain access to a particular workstation and pretend to be another user operating from that workstation.
2. A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.
3. A user may eavesdrop on exchanges and use a reply attack to gain entrance to a server or to disrupt operations.

In any of these cases, an unauthorized user may be able to gain access to services and data that he or she is not authorized to access. Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users.

Kerberos provides the following requirements:

1. Secure: a network eavesdropper should not be able to obtain the necessary information to impersonate a user.
2. Reliable: Kerberos should be highly reliable and should employ a distributed server architecture, with one system able to back up another.
3. Transparent: the user should not be aware that the authentication is taking place, beyond the requirement to enter a password.
4. Scalable: the system should be capable of supporting large numbers of clients and servers.

2. Kerberos - version 5 authentication dialogue [2]

A full-service Kerberos environment, consisting of a Kerberos server, a number of clients and a number of

application servers, requires that the Kerberos server must have the user ID (UID) and hashed passwords of all participating users in its database.

All users are registered with the Kerberos server. Such an environment is referred as a realm. Moreover, the Kerberos server must share a secret key with each server and every server is registered with the Kerberos server.

A simple authentication procedure must involve three steps:

1. The client C requests the user password and then send a message to the AS of the Kerberos system that includes the user's ID, the server's ID and the user's password.
2. The AS check its database to see if the user has supplied the proper password for this user ID and whether this user is permitted access to the server V. If both tests are passed, the AS accept the user as authentic and must now convince the server that this user is authentic. Thus the AS creates and sends back to C a ticket that contains the user's ID and network address and the server's ID. Then it is encrypted with the secret key shared by the AS and the server V.
3. C can now apply to V for the service. It sends a message to V containing C's ID and the ticket. V decrypts the ticket and verifies that the user ID in the ticket is the same of the one which came with the ticket. If these two match, the server grants the requested service to the client.

This scheme is correct, but this is not enough in a distributed environment: it has some leaks related to security and requires that the user introduce the password every time he needs to access to a service. Indeed, the client sends its password unencrypted to the AS; an eavesdropper could capture the password and use any services accessible to the victim. Moreover, it is better to minimize the number of times that the user has to enter a password.

To solve these additional problems, we introduce a scheme for avoiding plaintext passwords and a new server, known as the **Ticket-Granting Server** (TGS). The new service issues tickets to users who have been authenticated to AS. Each time the user require access to a new service, the client applies to the TGS using the ticket supplied by the AS to authenticate itself. The TGS then grants a ticket to the particular service and the client saves this ticket for future use.

The new scenario (Fig. 1) is as follow:

1. The client request a ticket-granting ticket on behalf of the user by sending its user's ID to the AS, together with the TGS ID, indicating a request to use the TGS service.
2. The AS responds with a message, encrypted with a key derived from the user's password that contains the ticket for the TGS. The encrypted message also contains a copy of the session key used by C and the TGS. In this way, only the user's client can read it. The same session key is included in the ticket, which can be read only by the TGS. Now C and the TGS share a common key.
3. Armed with the ticket and the session key, C is ready to approach the TGS. C sends the TGS a message that includes the ticket plus the ID of the requested service. In addition, C transmits an authenticator, which includes the ID and address of C's user and a timestamp; it is encrypted with the session key known only by C and TGS. Unlike the ticket, which is reusable, the authenticator is intended to use only once and has a very shot lifetime.

   The TGS can decrypt the ticket with the key that it shares with the AS. This ticket indicates that the user C has been provided with the session key K_c. Thus the TGS decrypt the authenticator with K_c gained from the ticket and check the name and the address from the authenticator with that of the ticket and with the network address of the incoming message. If all match, then the TGS is assured that the sender of the ticket is indeed the ticket's real owner.
4. Then the TGS replies to the client, sending a message encrypted with the common key that they share. It

includes a new session key to be used with the server V that must provide the service, the ID of V, the ticket valid for a specific service and the timestamp of the ticket. The ticket itself includes the new session key.

5. C now has a reusable service-granting ticket for V. When C presents this ticket, it also sends an authenticator. The server can decrypt the ticket, recover the session key and decrypt the authenticator.

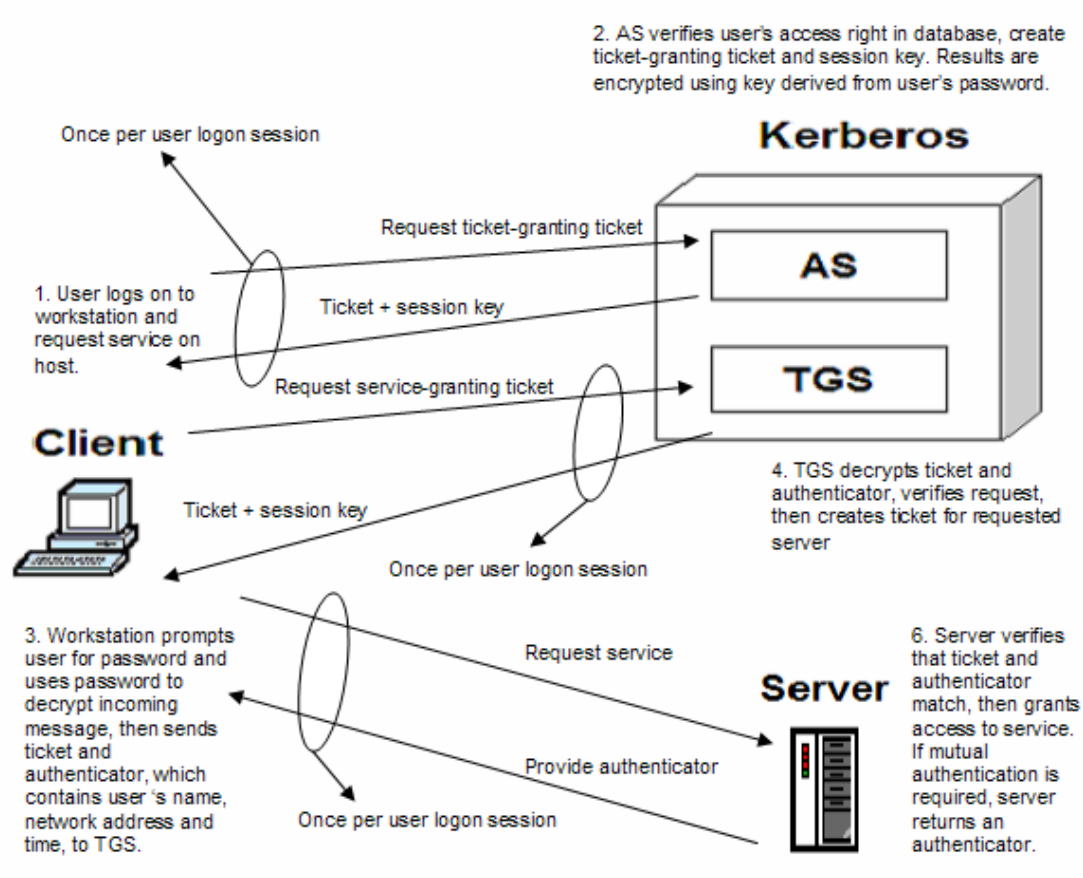


Fig. 1 Kerberos Authentication Dialogue

Finally, at the conclusion of this process, the client and the server share a secret key. This key can be used to encrypt future messages between the two or to exchange a new session key for that purpose.

The Kerberos system is also able to manage more complicated situations which involve more than one realm. Usually, networks of clients and servers under different administrative organizations constitute different realms.

Sometimes, user in one realm may need access to server in other realms and they need to be authenticated before to use services provided by those servers. Kerberos has a mechanism for supporting such inter-realm authentication.

The only requirement requested is that the Kerberos server in each interoperating realm shares a secret key with the server in the second realm. The two Kerberos server are registered with each other.

The communication mechanism between a client and a server in two different realms is the following:

1. The client C in the α realm needs a ticket in order to communicate with the server in the β realm. Thus, the user's client follows the usual procedures to gain access to the local TGS and then request a ticket-granting ticket for the remote TGS in realm β.

2. The client can then apply to the remote TGS for a service-granting ticket valid for the desired server in that different realm.

3. The client can now use the ticket obtained from the remote TGS with the server V in the other realm.

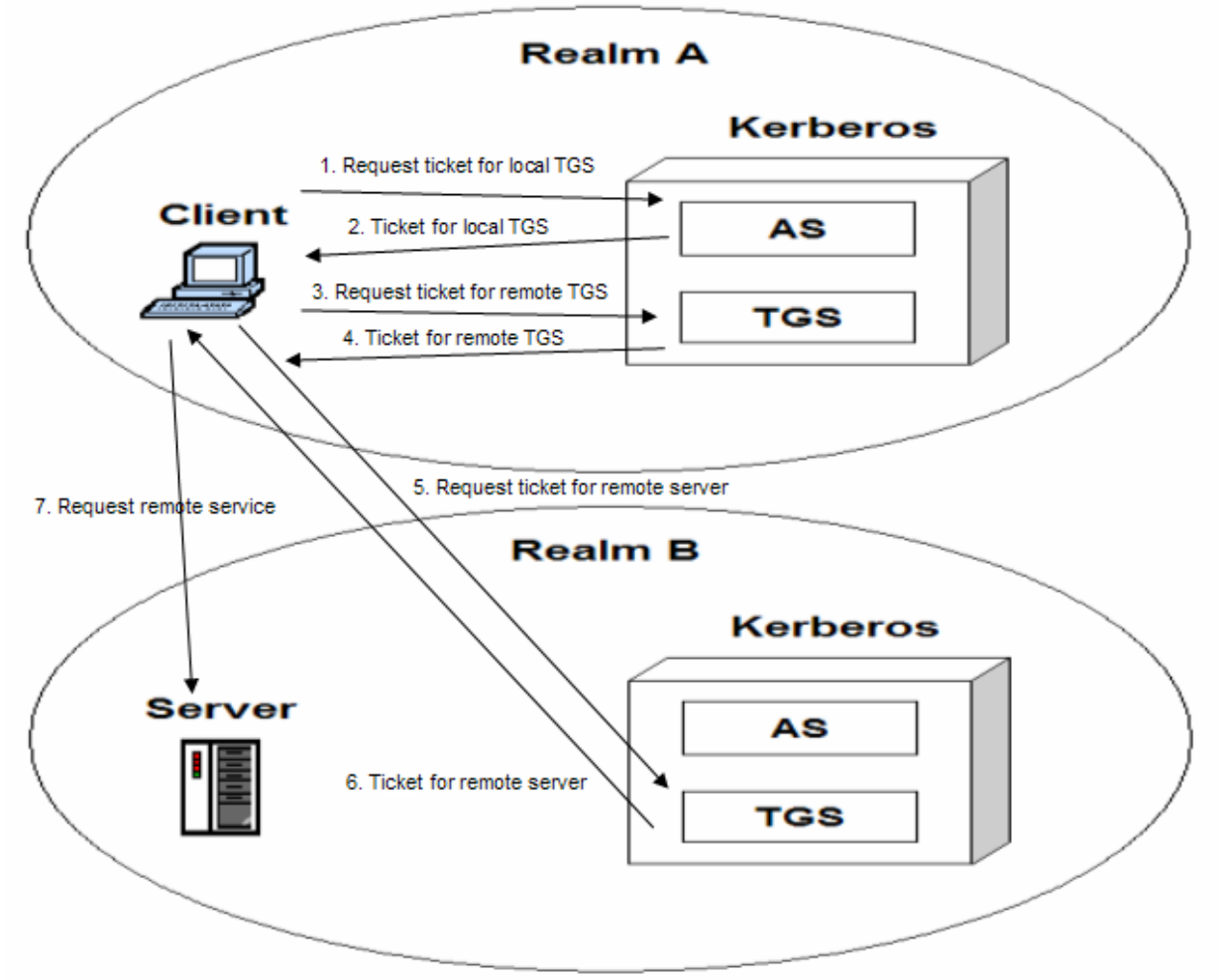


Fig. 2 Inter realm communication

3. Kerberos limitations [3]

Since Kerberos has many strengths, it has a number of limitations and some weakness in its protocol which have made Kerberos is not as extended as it should be.

One of the main problems of Kerberos (which is not relative to security of the protocol) is that any application which wants to use the Kerberos protocol, have to be modified in the code in order to establish a secure communication. This means high costs in terms of time and money, and is not reliable for all applications nor enterprises.

The timestamps are used to establish secure communication. Therefore, servers have to be synchronized within no more than a few minutes, and for instance, times servers are required to synchronized often their clocks. But then, "if a host can be misled about the correct time, a stale authenticator can be replayed without any trouble at all". A solution to this problem could be performed by "adding challenge/response as alternative to time-based authentication".

Another problem relative to security relies on the high dependency on the Kerberos server. If this server goes down, then the whole network goes down as well. This is something very expensive and not desirable in a distributed system. In addition to this, all security relies on the server, so if someone can access the Kerberos server, then all network connection could be hacked.

There is another potential problem which is that someone could steal from the network the message from the

AS to the client, this message, as we have seen above, is encrypted by the client key which is derivated from the client password. Therefore, someone could try to guess this key and then he would be able to identify himself as the original client. "Kerberos is not effective against password guessing attacks; if a user chooses a poor password, then an attacker guessing that password can impersonate the user". A solution proposed is the using of "exponential key exchange to provide an additional layer of encryption [...] it involves the two parties exchanging numbers that each can use to compute a secret key. An outsider, not knowing how the numbers were calculated, cannot easily derive the key".

Kerberos by itself cannot guarantee that the password will not travel along the net. It can happen that "the user enters a password to a program that has already been modified by an attacker (a Trojan horse), or if the path between the user and the initial authentication program can be monitored, then an attacker may obtain sufficient information to impersonate the user." Kerberos must be combined with other techniques to address these limitations.

Kerberos is designed to authenticate clients which are users, but "Kerberos is not a host-to-host protocol". Eventhougth, in last version 5, it has been extended to user-to-user authentication.

These limitations have made Kerberos became a protocol not as extended nor used as it should be. Most of the networks only need an encrypted communication protocol to establish a minimum security, therefore sometimes Kerberos is replace by programs as simple and transparent as SSH.

4. Kerberos' applications

4.1 Cisco's implementation of Kerberos client support [4]

To implement a Kerberos security system with a Cisco router remote, users attempting network services must pass through three layers of security before they can access network services.

The first one is called Authentication to the Boundary Router and describes the operations to follow in the authentication process. A remote user who successfully initiates a PPP session to the corporate site is prompted by the router in order to register himself with login and password. Although in this phase the user is inside the firewall, to gain access to the network services, it still must authenticate to the **Key Distribution Centre** (KDC). This because the TGT issued by the KDC is stored on the router and is not useful for additional authentication unless the user physically logs on the router.

Therefore the next step is Obtaining a TGT from a KDC. At this point, the client launches the KINIT program which is part of the software provided with Kerberos protocol. KINIT finds the user's identity and requests a TGT from the KDC. When the KINIT program receives the encrypt TGT, it prompts the user for the password to decrypt the ticket if it is successfully, the user has a TGT and can communicate securely with the KDC.

The third and last layer describes how the client, with a TGT, authenticates to network services within a given Kerberos realm. To make it possible the router must share a secret key with the KDC. To do this, you must give the router a copy of the SRVTAB you extracted on the KDC. This file contains the passwords or randomly generated keys for the service principals you entered into the KDC database.

The most secure method to copy SVRTAB files to the hosts in your Kerberos realm is to copy them manually in each host in turn. For the router, instead, you must transfer them via the network using the TFTP. Finally the user is authenticate to the network service.

The entire process described above is repeated each time a user wants to access a network service in the Kerberos realm.

4.2 Kerberos component in Windows 2000 [5]

Windows 2000 uses the domain's Active Directory to implement in the KDC the action of getting some information about users from the Global Catalogue.

The KDC is located on every domain controller, as is the Active Directory service. Both services are started automatically by the **Local Security Authority** (LSA) of the domain controller and they run in the process space of the LSA. It's supposed that neither service can be stopped.

Windows 2000 ensures availability of these services by allowing each domain to have several domain controllers, all peers. Any domain controller can accept authentication requests and ticket-granting requests addressed to the domain's KDC.

4.3 Kerberos Policy

In Windows 2000, Kerberos policy is defined at the domain level and implemented by the domain's KDC. Kerberos policy is stored in Active Directory as a subset of the attributes of domain security policy. By default, policy options can be set only by members of the Domain Administrators group.

Kerberos policy includes these settings:

- Maximum user ticket lifetime (default is ten hours).
- Maximum lifetime that a user ticket can be renewed (default is seven days).
- Maximum service ticket lifetime (default is ten hours).
- Maximum tolerance for synchronization of computer clocks (default is five minutes).
- Enforce user logon restrictions (default is enabled).

A dynamic-link library (**SSP** = security support provider) supplied with Windows 2000 implements the Kerberos authentication protocol. SSP, by default, is loaded by the LSA in the system booting phase and it may be used either to authenticate network logons and client/server connections. The choice depends on the capabilities of the computer on the other side of the connection.

System services and transport-level applications access SSPs through the Microsoft Security Support Provider Interface (**SSPI**). This interface is used to enumerate the providers available on a system and selected one it tries to obtain an authenticated connection with it.

If the Kerberos SSP has been selected, this method generates a *KRB_AP_REQ* message from the client. The application on the server's side of the connection responds with the SSPI method *AcceptSecurityContext*, which generates a *KRB_AP_REP* message from the server. Once the connection has been authenticated, the LSA on the server uses information from the client's ticket to build an access token. The server then invokes the SSPI method *ImpersonateSecurityContext* to attach the access token to an impersonation thread for the service.

4.4 Credentials Cache

Tickets and keys obtained from the KDC are stored in a credentials cache protected by the LSA. If security principal logs off or the system is shut down, all objects stored there are destroyed. Anyway, LSA keeps a copy of an interactive user's hashed password in order to obtain a new TGT silently from the KDC without interrupting the user's logon session.

4.5 Smart Card Logon in Windows 2000

Moreover, Windows 2000 implements extensions to the Kerberos protocol that permit initial authentication using public key certificates rather than conventional shared secret keys. In standard Kerberos logons, indeed, the key derived from the user's password is used for both encryption and decryption (**symmetric cryptography**).

To support smart card logons, Windows 2000 implements a public key extension to the Kerberos protocol's (**asymmetric cryptography**). Two different keys are needed, one to encrypt, the other to decrypt. Together, the keys needed to perform both operations make up a private/public key pair. The private key is known only to the owner of the pair and is never shared. The public key can be made available to anyone with whom the owner wishes to exchange confidential information.

When a smart card is used in place of a password, a private/public key pair stored on the user's smart card is substituted for the shared secret key derived from the user's password. In the public key extension to the Kerberos protocol, the initial exchange is modified so that the KDC encrypts the user's logon session key with the public half of the user's key pair. The client decrypts the logon session key with the private half of the pair.

The Kerberos SSP on the client computer sends the user's public key certificate to the KDC as pre-authentication data in its initial authentication request, KRB_AS_REQ. The KDC validates the certificate, extracts the public key, and uses it to encrypt a logon session key. It returns the encrypted logon session key, along with a TGT, in its reply to the client, KRB_AS_REP. If the client is in possession of the private half of the key pair, it will be able to use the private key to decrypt the logon session key. Both the client and the KDC then use the logon session key in all further communications with one another. No other deviation from the standard protocol is necessary.

This operation can be done when the user inserts a smart card into a card reader attached to the computer and when Windows 2000 are configured for smart card logon.

(The extensions for public key authentication are based on a draft specification submitted to the IETF working group by a number of third parties with interests in public key technology.)

5. Performance Analysis [6]

In this part we are going to discuss which are the performance, in terms of throughput and mean time delay, of a distributed system that implements the Kerberos protocol.

We consider a computer network (an Ethernet LAN with transmission rate C = 10 Mbits per second, with a cable of length 2500 meters and a propagation speed of $2*10^8$ meters per second) employing multiple TGS's to establish secure communication between a number of clients and servers, in this environment a queuing analysis is carried out to determine the average time for message transfer.

Before starting we shall make some assumptions on the system we are going to analyse, in order to simplify and make more clear the calculation.

Since in the Kerberos protocol, that we have previously described, the TGS plays the major role, we assume that the security manager comprises only TGS's, in number of k.

These security servers (TGS) meet the traffic demand generated by the m clients and the n servers existing in a computer network, so there are two possible configurations:

- the clients could be clustered in different dispersed locations and each cluster is served by one TGS,
- the servers could be in different sites and each group of servers should be managed by a different TGS.

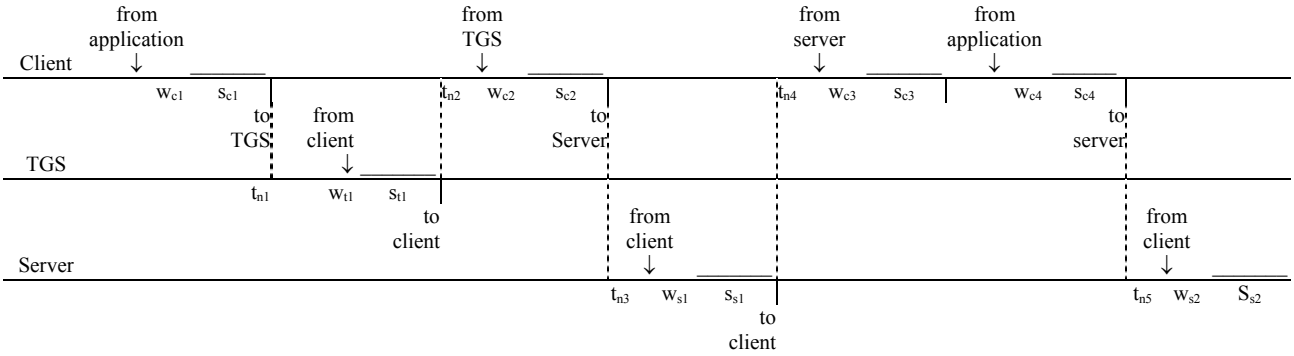
The requests for receiving tickets from a TGS arrive at each client according to an exponential distribution with mean rate λ requests per second, so the same value characterizes the mean arrival rate of tickets from a TGS to a

client and from a client to a server, it follows that it is the same for the authenticators sent from a server to a client. Moreover the mean arrival rate of messages exchanged between a client and a server is v messages per second. For making things more simple, all the clients are identical and have the same statistics, same assumption also for the servers and for the TGS's.

We assume the following message lengths for the Kerberos protocol:

Key = 8 bytes	Ticket = 32 bytes	Authenticator = 24 bytes
Timestamp = 8 bytes		Mean message length = 125 bytes

Now that we have made all the assumptions we could start the queuing analysis.



This picture illustrates the time sequence diagram showing the operations needed to serve a request from the client. At first an application issues a request for accessing a specific server, this request has to wait for a time w_{c1} before it can be processed due to the fact that other request could be in the client waiting for other services (being encrypted, in the message transfer phase, etc.).

Once admitted, the client responds to the request by producing an unencrypted message whose service time is s_{c1}, since here no encryption is involved we could assume that s_{c1} is negligible. Such message accesses the network and is transferred over it during a time t_{n1}. Once it arrives at the TGS, the message waits for a time w_{t1} after which a response message is produced and encrypted during a time s_{t1}=(8+32)*8/TDES, where TDES is the rate of symmetric key encryption in bits per second.

Again the message from the TGS is transferred to the client over the network during a time t_{n2}. At the client, this encrypted message waits for service for an interval w_{c2} and is then served in an interval s_{c2}=(8+24)*8/TDES. The output of the client is a message containing an encrypted ticket and an authenticator which travels over the network during a time t_{n3}. Once this message arrives at the desired server, it waits for a time w_{s1} after which it is processed during s_{s1}=(32+24+8)*8/TDES and the server produce a timestamp: $STAMP_{c,s}$.

The client waits for $STAMP_{c,s}$ which arrives after travelling over the network during a time t_{n4}. This message waits at the client for a time w_{c3} before it is admitted into service.

The client decrypts $STAMP_{c,s}$ in time s_{c3}=8*8/TDES and later if it wants to communicate with the server it waits a time w_{c4} before it proceeds to encrypt the information message in time s_{c4}=125*8/TDES. The message travels over the network during a time t_{n5}, before it arrives at the designated server. There it waits for a time w_{s2} after which it is processed by the server during a time s_{s2}=125*8/TDES.

It is reasonable to assume that all client and server messages spend the same average time waiting for service, therefore $M[w_{c1}] = M[w_{c2}] = M[w_{c3}] = M[w_{c4}]$ and $M[w_{s1}] = M[w_{s2}]$, moreover we assume that $M[t_{n1}] = M[t_{n2}] = M[t_{n3}] = M[t_{n4}] = T_1$ and $M[t_{n5}] = T_2$.

Thus the average message time T_{av}, assuming r sessions per ticket $v = r*\lambda$, is:

$$T_{av} = \frac{1}{r}\{M[3w_{c1}] + E[w_{t1} + s_{t1}] + s_{c2} + M[w_{s1}] + s_{s1} + s_{c3} + M[t_{n1} + t_{n2} + t_{n3} + t_{n4}]\} + M[w_{c1}] + M[s_{c4}] + M[w_{s1}] + s_{s2} + M[t_{n5}]$$

and knowing that the TGS is modelled as an M/D/1 queue as it processes messages with deterministic lengths, and the clients and the server are modelled as M/G/1 because they process messages with variable length, after the substitutions we obtain:

$$T_{av} = \frac{1-\left(\frac{160mnv}{kr * TDES}\right)}{r\left(\frac{TDES}{320} - \frac{nmv}{rk}\right)} + \frac{mv(1+r)\frac{13107210^6 r}{r^2(TDES)^2}}{1-\frac{mv\left(\frac{1}{r}+1\right)(1000r+512)}{(r+1)TDES}} + \frac{nv(3+r)\frac{34816+10^6 r}{r^2 TDES^2}}{1-\frac{nv\left(\frac{3}{r}+1\right)(1000r+320)}{(r+3)TDES}} + \frac{832+2000r}{rTDES} + \frac{4}{r}T_1 + T_2$$

As we can see from the above expression, the performance is a function of the number k of TGS's, of the number of sessions per ticket (r), and of the encryption rate TDES.

By applying the function to fixed data we can deduce some facts: first for low encryption rates the system performance is not limited by the physical network throughput, but it depends on throughput of queues at the client, at the server and at the TGS.

Secondly for an encryption limited system, assuming an Multi-Tasking processing mode, increasing the number of TGS's has the effect of increasing the overall throughput while reducing the average time delay. This is due to the fact that encryption limitation is no longer caused by the TGS, but becomes client/server limited. This is also obvious from the fact that if we have a high value for r say r = 10, the system performance is independent of the number of TGS's.

Thirdly for a network limited system the performance is almost independent of the number of TGS's because the throughput of the physical network determined the overall throughput of the client, of the server and of the TGS queues.

Last exchanging fewer messages between the client and the TGS, for a given number of client-to-server messages, the performance of a network limited system is improved considerably because in this case less traffic is applied to the physical network.

So we can conclude that when we are designing a computer network with security managers, we should maximize the encryption rate using efficient encryption schemes and hardware implementation, and then reduce the frequency at which clients access a security manager, so we can achieve higher overall throughput and smaller average in time delay.

6. References

[1] J. Kohl C. Neuman, RFC 1510, *The Kerberos Network Authentication Service (V5)*, September 1995

[2] William Stallings, *Cryptography and Network security (Principle and Practice)*, Upper Saddle River N.J., Prentice Hall 1999

[3] Steven M. Bellowin and Michael Merit from AT&T Bell Laboratories, *Limitations of the Kerberos Authentication System*, Winter '91 USENIX Conference Proceedings, USENIX Association, 1991

[4] 1992--2001 Cisco Systems, Inc. All rights reserved http://www.cisco.com/univercd/cc/td/doc/product/software/ios121/121cgcr/secur_c/scprt2/scdkerb.htm, August 2001

[5] © 2002 Microsoft Corporation. All rights reserved, Windows 2000 Kerberos Authentication - White Paper http://www.microsoft.com/technet/treeview/default.asp?url=/TechNet/prodtechnol/windows2000serv/deploy/kerberos.asp

[6] Mahmoud T. El-Hadidi, Nadia H. Hegazi, Heba K. Aslan, *Performance Analysis of the Kerberos Protocol in a Distributed Environment*, 2nd IEEE Symposium on Computers and Communication (ISCC 1997).