
0.0.1 Introduction

Kerberos is a complex concept to do authentication and authorisation in a network, but its one of the most used methods to do this, albeit most of the users arent aware of this fact.

At the end of the section explaining the theory of Kerberos there will be many terms known to the Windows Administrator. The whole Microsoft Active Directory authentication system is based on Kerberos Technology. Of course our friends in Redmon didnt embraced this standard without extending the system with unstandardized components. But at the end its still Kerberos. Okay, its Kerberos plus extensions and LDAP plus Extensions.

0.0.2 The aims of Keberos

To understand the concepts of Kerberos i will summarize the design objectives of it at first. Many seemingly overcomplex mechanisms become perfect and adequate solutions when you take this aims into considerations.

A password ...

- ... must not travel unencrypted over the network.
- ... must not stored on any client.
- ... must be discarded on the client right after use.
- ... must not be stored unencrypted even on the servers implementing the authentication.
- ... must not be entered more than once in a worksession.

An entity requiring authentication ...

- ... must be able to authenticate itself as the legitimate instance service.¹

Authentication informations must be placed on a central server to enable ...

- ... the central administration of the authentication informations.
- ... the user to change his password at a single location for all services at once.
- ... to have only one repository for authentication informations instead of having several ones.

¹Otherwise, the service would know, that you are a legitimate user, but do you know, if the service ist the legitimate one?)

0.0.3 The components of kerberos

Once you have understood the concept of Kerberos, its easy to setup (well, its even easy to setup without any understanding, when you follow this tutorial). But to help you with to understand the concept, i will use the example of the security of a customer.

When you come to the lobby of the customer, you have to show your personal identity card. After this, you get a RFID card to open the entry gate and to open further doors at the customer premise. dependent to the status this card has different rights: A visitor like me can just open the entry gate with the card and its just valid for a day to a week. An important employee of the customer can open all doors with its card and this card is valid for a year.

Kerberos works exactly this way. But the nomenclature isnt that easy.

Encryption

Much of the inner working of Kerberos is based on cryptography. Kerberos make vast usage of symetric cryptographic algorithms. As i told you before, one of the aims of Kerberos was to remove the need to transport the password over networks. When using encryption this is really simple.

A password can be used as a shared secret. I want to give you an real life example: Lets assume you are a spy. You have a new contact officer in the country, but you both havent worked together. How should you know if the contact office isnt a guy from the counter intelligence, and how should the contact officer know that you arent just a agent provocateur. Obviously you cant say "Hey, im a spy, are you my new contact officer". But there is a way to circumvent this. The old contact officer told you both that "playing golf at the National Golf Club" means "going to to the pub "Dancing Horse" ". This is a shared secret between you and the contact officer.

You ask the contact officer on a party, if he or she wants to play golf at the next weekend. The contact officers ansers with "Of, course and tells you that he is member at the National Golf club and suggests this as a good location". Now the authentication is easy. If you both get to the pub "Dancing Horse", you found your contact officer and the contact officer found you. If one of you both walks to the National Golf Club, the other knows that this person wasnt aware of the shared secret thus wasnt the person the other thought. You didnt had to disclose the shared secret, you just needed it to "decrypt" the conversation to do the right thing to authenticate your self.

The authentication in Kerberos works in a really similar manner. And encryption is used on several other locations as well. In the further text i will explain for example how the

system can give the client an authentication to access a service in such a way, that the client cant alter or even read the authentication.

Realm

When we get back to the the example of the customer security the realm is equivalent to the area of the customer location. The realm defines the border of the validity of your authorisation. To take an example from hamburg: Its obvious that my Sun Badge isnt valid for example at the Airbus premises or at the AMD Fab in Dresden. I need to reauthenticate with my personal identiy card and get a new badge that authorizes me for a short time to enter the premises of the customer.

Kerberos knows this concept too. Its called *realm*. By an convention the domain name in capital letters is used for a realm. For example: A realm in my network at home would be `MOELLENKAMP.ORG`, as my internal domain name server uses this domain for my systems. The realm at Sun may be `SUN.COM`

Principal

In Kerberos speak the Principals are the identities in the authentication. Every acting entity in the Kerberos system has a principal name in the concept. So not just peoples have a principal in this concept, Kerberos specific services have such a name, hosts have such a name and even services under the control of Kerberos have such a name.

A Principal is constructed as following:

```
Component 1/Component 2/.../Component n@Name of the Realm
```

For example the principal for myself would be `joerg` or fully written `joerg@MOELLENKAMP.ORG`. The principal for my fileserver would be `host/silmarillion` and the principal for the NFS service on my fileserver would be `nfs/silmarillion`. The advantage to give everything a identity is the fact, that services or hosts can identify themself each other or to the client, making man-in-the-middle attacks impossible.

Keys

As i wrote before in the section about Encryption many mechanisms of Kerberos are based on on shared secrets between two principals. This shared secret is used to encrypt the communication between the communication partners. This is not only done to obfuscate the content of the communication: You are just able to encrypt the communication when you are aware of the shared secret, you cant react accordingly to the answer, if you are

not aware of the secret. By using this mechanism, you have established an authentication mechanism as well.

Long Term Secret: Long time secrets are shared secrets valid for a longer time (weeks or months).

Session Keys: As the name has already told you two communication partners use for a communication a session key.

Key Distribution Center

The equivalent of the nice women in the lobby of this customers is the Key Distribution Center (KDC). In the world of Kerberos this is central point where all pieces fit together. Its the central location for the authentication of the users but its as well the place where the keys are distributed.

The Key Distribution Center consists out of two components:

Authentication Server: This component authenticates the principals.

Ticket Granting Server: This component issues all tickets in use of the Kerberos realm.

Albeit both server are distinct components of the KDC, they are not implemented on different servers.

The name "Key Distribution Center" comes from its main work after authenticating the user. It distributes session keys.

Ticket

One of the central concepts of Kerberos is the ticket. A ticket is the authorisation to use a service. Lets assume you want to go into the super-secret R&D lab of Sun Microsystems deep under the earth. The friendly person in the lobby (those with the super secret laser weapons) give you an envelope with the strict advise not to open it, as the envelope would self-destruct when you try to read or alter it. Then you go to the guard after you took the elevator to an room thirty floors under the earth. The friendly person (with surgical removed humour and the Higgs-Boson projector gun) in that lobby. This guard opens the envelope, checks the letter, nod her head and opens the door for you. You can enter the super secret research lab at sun with scientists and aliens testing Rock VI², and mind bending rays for analysts and journalists ³

²No, there is no missing y

³You didn't thought, Ashlee left the Register on his own mind, but something went bad, he went to the New York Times and not to "Console Gaming Today" as we have planed, but hey ... it was a prototype

Fun aside. The ticket in Kerberos is this ticket is this letter in a envelope. Of course as we speak about computer it is an encrypted bunch of data.

Ticket Granting Ticket

There is a special type of ticket. Its the ticket where the authentication of a client starts. This ticket is called *Ticket Granting Ticket*

It would be nonsense to check your identity on the customer site by looking at you personal identity card and calling the visited person, every time you want to open a door to enter a building. But the nice woman in the lobby gave you a RFID-card, so they checked already your authorization. So every door guard or door lock has just to check the validity of the RFID-card and if you authorized to enter this special building or this door.

In Kerberos speak the lobby security gave you a Ticket Granting Ticket. This ticket is a normal ticket, with an exception: Its a ticket that authorizes you to use the Ticket Granting Service. As long you have a valid ticket for the Ticket Granting Service, you dont have to go to the Authentication Server again. The single sign on property of the Kerberos system is implemented by the two staged approach of authentication service and ticket granting service.

Authenticator

0.0.4 Introduction to the authentication with Kerberos

Now you know a lot of new jargon, but how does the authentication works. There are two main cases that happens in this process:

- authenticating the client and issuing the initial ticket
- issuing subsequent tickets

Authenticating the client

Lets start with first request. The new client hasnt worked so far with the kerberized services today, thus we have to authenticate the user at frist.

The process starts with the *AS_REQUEST*. This is the first step. This requests contains the following data:

- The principal name of the user asking for authentication. Lets assume, i want access to my system, this would be `joerg@MOELLENKAMP.ORG`.
- The principal name of the service. For example the ticket granting service in the `MOELLENKAMP.ORG` realm would be `krbtgt@MOELLENKAMP.ORG`.
- A list of IP-Numbers. This list can be an empty one. This would lead to the acceptance of the client from any IP-Address. One or more IP numbers would limit the access to the named IP numbers.
- At last the requested lifetime for this ticket.

The client sends this request without any encryption to the Authentication Server. The Authentication server accept this requests and checks if the principal names exist in the database of the KDC. As the `krbtgt` service is a standard service of Kerberos, it should exist in any case.

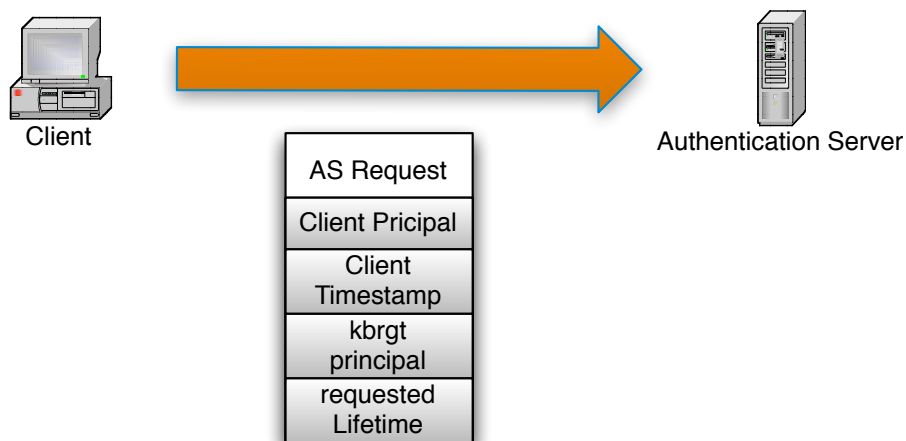


Figure 0.1: AS_REQUEST Packet

When both principals exist the KDC database, it starts to assemble the AS_REPLY. At first it generates a random key. This key is called the Session Key $SK_{Client2TGS}$. After this step it generates the Ticket Granting Ticket. This TGT contains:

- The principal name of the user on which behalf the AS_REQUEST has been transmitted
- The principal name of the Ticket Granting Service in the realm.
- The IP list
- A timestamp

-
- Lifetime
 - the generated Session Key.

This ticket is encrypted with the long term secret of the ticket granting service. But we dont send this as a reply. The client couldnt decrypt it, as its encrypted with a key the client isnt aware of. So it would be just garbage for the client. But the TGT is used in the AS_REPLY.

Now the Authentication Server assembles the rest of the AS_REPLY. It consists out of.

- the principal name of the service, in our example the principal of the ticket granting service.
- a timestamp
- the lifetime of the ticket
- the generated Session Key

This part is encrypted with the long term secret of the client. Both components are concatenated and form the AS_REPLY. Some informations seem to be redundant, but keep in mind that either the client or the Ticket Granting Server just can read half of the ticket, thus you have to transmit this information twice.

Okay, after the transmission of the AS_REPLY to the client, the client software asks for the password. This password is transfered into the shared long term secret of the client principal by a hash function. If the user has entered the correct password, it will transform to the correct shared secret and thus the client will be able to strip the encryption of the first part of the reply. Now it takes the TGT part and stores it in a cache ⁴.

Issuing the subsequent tickets

At this moment, the user has authenticated but he holds just a ticket for a special service, the Ticket Granting Service. But the user want to do real work, thus he or she has to gather a ticket for a service. To do so, the client sends a Ticket Granting Server Request *TGS_REQ* to the Ticket Granting Service. As you may remember, this service lives on the KDC as well.

At first the client generates the Authenticator. As explained earlier this is an additional security measure. The authenticator consists out of:

- the name of the client principal
- a timestamp

⁴This cache is called the credential cache

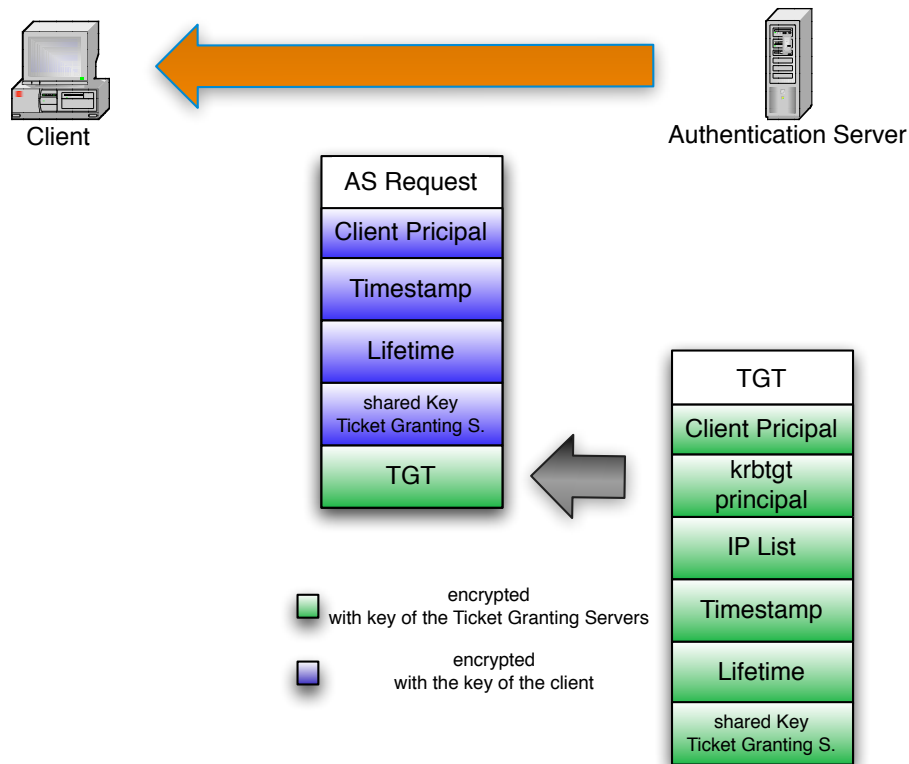


Figure 0.2: AS_REPLY Packet

This authenticator is encrypted with the session key negotiated in the issuing of the TGT. This authenticator doesn't get transmitted alone. It's just a part of the *TGS_REQ*. The data in this request is:

- the name of the service principal
- the requested lifetime for the ticket
- the authenticator
- the already encrypted TGT

This packet is transmitted to the ticket granting service. The name of the service principal and the requested lifetime isn't encrypted in this packet. The ticket granting service receives the information and starts to check the validity of this ticket. At first it looks into the *TGS_REQ* for the service principal. When this principal exists in the database, it gathers the shared key for the service *krbtgt* (The Ticket Granting Server) and decrypts the TGT with it. The TGS takes the shared key negotiated in the TGT and decrypts

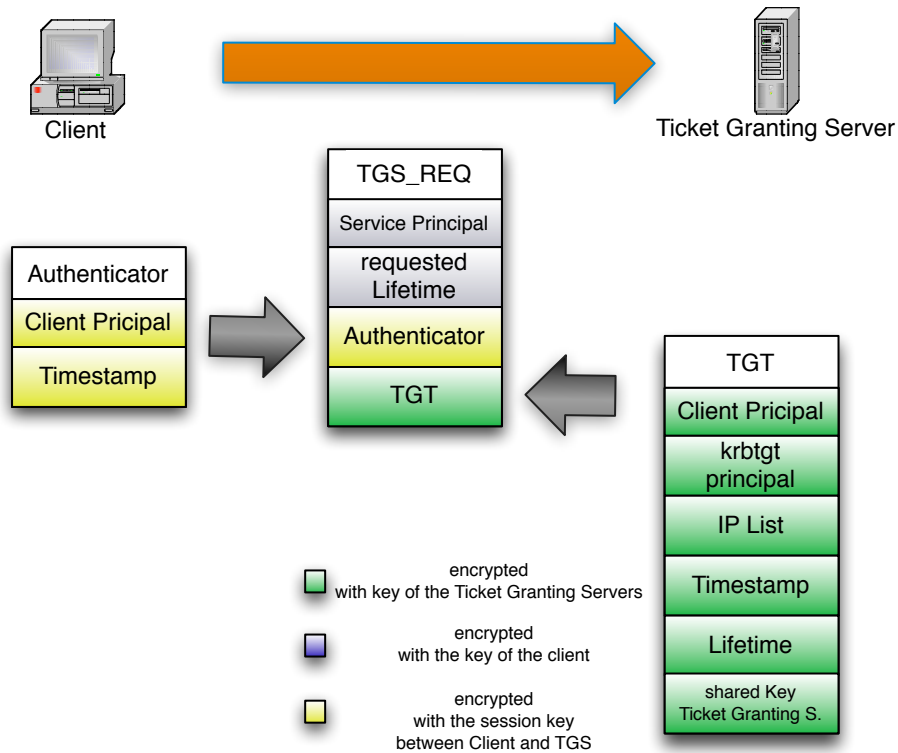


Figure 0.3: TGS_REQ Packet

the Authenticator with this key. A request must fulfill several conditions before a service ticket is issued by the TGT:

- the TGT must not be expired
- the principal name of the client must exist in the database of the KDC
- the authenticator must exist in the replay cache.
- when the IP list is not empty, the request must be transmitted from one of the IP numbers mentioned in the field.

When these four conditions are fulfilled, that the TGT was transmitted by a legitimate user. The TGS can start now with the assembly of the TGS.REP package. For the communication with the server Kerberos will use a new session key. So it has to generate a Key at first.

After this, the KDC generates the Service Ticket. It consists out of:

- the name of the client principal

- the name of the service principal
- an IP list
- a timestamp
- the lifetime of the ticket
- and the session key for the connection between client and the service.

This Service Ticket gets encrypted with the long term secret of the service. Now the TGS_REP packet is assembled. The following components are inside this reply:

- the principal name of the service
- a timestamp
- the lifetime of the service ticket
- the session key for the communication between client and the service.

This part of the TGS_REP will be encrypted with the session key negotiated between the client and the Ticket Granting Server. After this the service ticket is appended to the first part and the TGS_REP packet is completed.

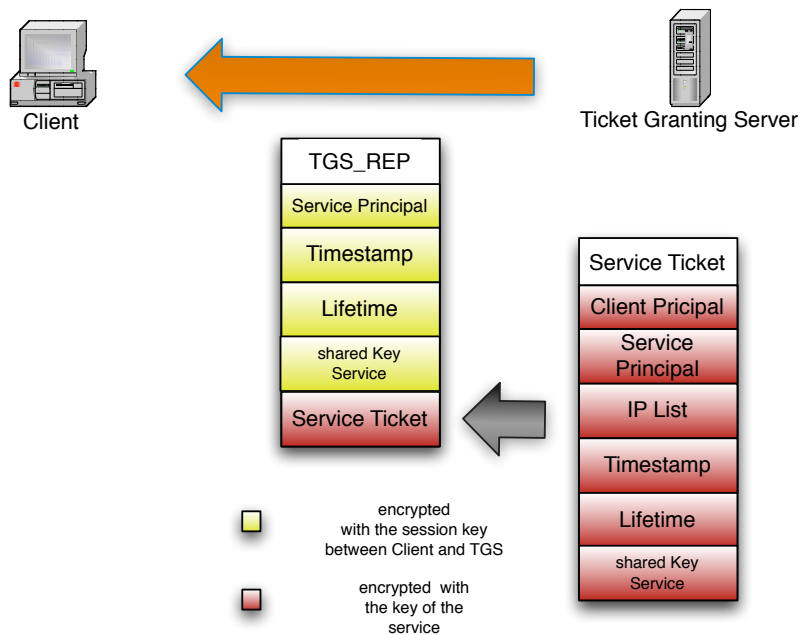


Figure 0.4: TGS_REP Packet

As with the Ticket Granting Ticket some information look at first to be redundant. But its similar to the TGT: They are encrypted with different key, thus informations needed for both partners in a communication relation has to be transmitted twice.

Okay, with this service ticket, we can connect to an Service Server. To connect to such a server, we send an Application Request AS_REQ packet to the system. At first a new authenticator is build for this request. This authenticator is constructed with the following components:

- the principal name of the client
- the timestamp

This authenticator will be encrypted with the session key negotiated between the Service principal and the Client principal. The already encrypted Service Ticket is appended to this and the AP_REQ is transmitted to the Service Server.

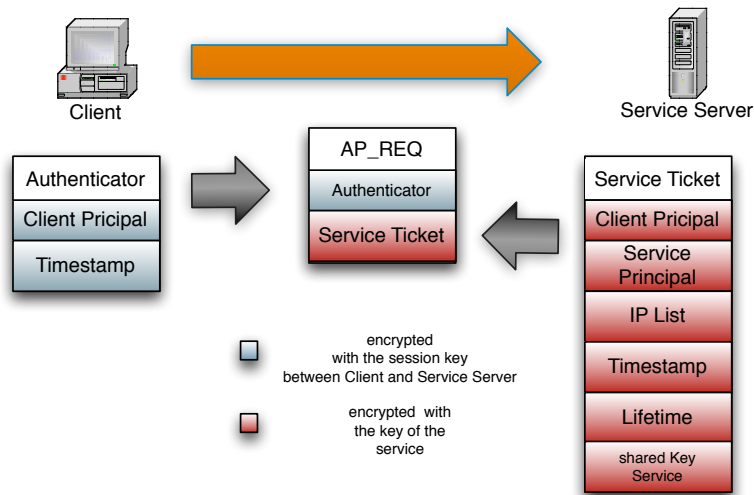


Figure 0.5: TGS_REP Packet

Then the system receives the AS_REQ, it decrypts the service ticket with the long term secret of the Service.

The Service server grants access to the service, when the following conditions are fulfilled:

- the ticket is still valid and hasnt expired so far.
- the principal name of the client is identical in the authenticator and in the service ticket.

-
- the authenticator is not in the replay cache and it hasn't expired so far ...
 - The IP number of the requesting client is in the IP list of the ticket (in the case the IP list isn't empty)

There is no mandatory answer to this Request. You are just authenticated to use the service and the service should react accordingly like delivering mails via IMAP or providing a shell via NFS. Nevertheless there is a AS_REP packet specified in the Kerberos Protocol. It's just a authenticator containing

- the client's principal
- the timestamp in the AS_REQ incremented by one second

This authenticator is encrypted with the session key negotiated for the communication between the client principal and the service principal. By doing so, the service authenticates itself to the client, as only the KDC, the client and the service are aware of this session key.

0.0.5 Prerequisites

0.0.6 Installing the master

0.0.7 Installing the slave

0.0.8 Configuring the client

0.0.9 Using kerberized SSH

0.0.10 Using kerberized NFS

NFS has the reputation of being unsecure. I have a strong opinion where this reputation has its source. NFS has a problem with Linux. Linux has a halfway decent NFS implementation but many advanced features are missing.

One of these advanced features is the integration of Kerberos into NFS.

0.0.11 Conclusion