# KERBEROS

Kerberos is a general framework in order to provide authentication in distributed systems. This framework is providing safe access to resources of a network. Obviously also programs can be authorized to use resources, not only humans. Today Kerberos 5 is present.
The main idea of Kerberos is that users that want to use the service of another party they have to make a request to the KDC for a ticket, and than they have to present this ticket to that party who offers the service. Of course every user share a secret key with KDC. Not only human are users, many times also applications. The difference is that while humans have not strong capabilities and their keys are derived from passwords, applications can store long keys that are stored when that application is setted up.
Scenario: **A** needs to access services provided by **B**. A has to authenticate in order to use these services. In some cases also B has to be authenticated. Then there is a trusted authority **C**, that is a reference Server trusted by every party, and it is known with the name of **KDC** (Key Distribution Center), because it provides session keys. KDC is storing a Master Key for every participant, so every user has a master key that share with the KDC. The approach is to generate tickets allowing users to access services. There can be a timeout for expyring tickets. Alice use the ticket to start a conversation to Bob. It is important that the keys are stored physically in a safe place.
KDC has a master key itself, it is used to encrypt all the other keys. So KDC stores all the master keys of the users encrypted by means of KDC's master key.
In order to have a good Kerberos implementation, the machines must have their clocks synchronized, because in Kerberos timestamps are much important. So an attacker can attack by changing a clock.

## Kerberos preliminary implementation

The idea is simple. Alice wants to use a service of Bob, she is sending a request to KDC for the ticket and the KDC is providing it obviously encrypted, after that Alice will use this information with Bob. All the information provided by KDC is encrypted by the master key of Alice $K_A$ and such information contains the session key $K_{AB}$ and the *ticket* for Bob. This ticket contains further informations, and is encrypted by means of the master key of Bob $K_B$, therefore only Bob can read such a ticket. Such ticket contains the identity of Alice of course, the session key and some other stuff like the *lifetime*. So basically when Alice asks to KDC to want to talk with Bob, KDC replies with $K_A(K_{AB},K_B(\text{ticket}))$. So Alice will decrypt the message obtaining the session key and the ticket encrypted, and will send the encrypted ticket $K_B(\text{ticket})$ to Bob in order to show she is allowed to use Bob's service.

## Simplified version

Alice wants to use a Bob's service, so she has to ask the ticket for Bob. She sends to the Server "I'm Alice, I want to talk to Bob, this is my nounce N". The server replies sending to Alice *ticketB* (which is encrypted with $K_B$) and the encryption by means of $K_A$ of the session key $K_{AB}$, the *nonce N*, the *lifetime* L and the identity of Bob. Alice then checks the *nonce* N and learns about ticket lifetime L. Then Alice sends to Bob the ticketB and the authenticator which is $K_{AB}(A,t_A)$, Bob checks identity if Alice in ticketB and in authenticator and checks if they are the same. In particular, Bob decrypts the ticket with his key, obtaining identity of Alice and the session key, then with the session key he will decrypt the authenticator and checks if the identity of Alice in it is equal to the identity in the ticket. Moreover, he checks the timestamp in order to see if the ticket is still valid. Then Bob sends to Alice the encryption of $t_A$ by means of the $K_{AB}$.

Now we have some practical problems, because for every session between Host and KDC, the messages should be protected with the master key, every time Alice requires a ticket it has to be encrypted. Too much overhead. We have 2 possible solutions:

- everytime Alice needs the Server she has to retype the password.
- the other possibility is that the master key is stored locally.

None of the 2 solutions is considered good (we don't like to retype the password everytime and we don't like to store the master key because it is not secure).
A solution is the following:
**TGT (Ticket-granting Ticket):** this is a meta ticket. (d'amore presents us his sticker). The idea is that Alice is having a ticket for asking tickets, so everytime she needs a ticket she shows a certain ticket instead retyping the password everytime.
The high-level approach is the following:
Slide 13: Kerberos has two parties, one of them is the Ticket-granting Server (TGS). There are 6 steps (grouped by groups of 2 interactions for both the directions).
Steps:

1. Alice login the workstation for the 1st time today and requests the **ticket-granting ticket** to the Authentication Server (AS). AS checks a local database.
2. AS sends to Alice the ticket-granting ticket and the session key $S_A$.
3. Alice now wants to use a particular service, she requests the service-granting ticket showing the ticket-granting ticket to the TGS (that is the Server that provides tickets for services).
4. TGS answers Alice with the requested ticket and a session key valid between Alice and the service.
5. Alice now can request the service to the Server because she has the service-granting ticket
6. The server answers with *server authenticator*.

In details what happens is the following:
Alice at the beginning (first interaction) does the login typing her password and the KDC derives from it a session key $S_A$ (that is meant to be valid for the session of interactions of the entire day for example. Keep in mind that KDC won't store such session key $S_A$). This key has a fixed life time (as we just mentioned it can be 1 day). Then KDC gives Alice a TGT that is including such session key $S_A$. This TGT is encrypted by means of the master key of the Server $K_{KDC}$, because the TGT is meant to be sent to the Server. All these informations are sent to Alice encrypted by means of $K_A$ of course.
When then Alice needs a service, she has to ask for a ticket, and in order to get the ticket she uses the TGT. So Alice shows TGT to the Server, which will decrypt it and invent the session key $K_{AB}$, then creates the ticket (encrypted with $K_B$), and the server sends back to Alice all these information encrypted with $S_A$.
So now Alice has the ticket, in order to use it she sends it to Bob, togheter with the timestamp encrypted by $K_{AB}$ (as happened in the simplified version of Kerberos). What it happens is that first Bob decrypts the ticket, which will contain $K_{AB}$, and so he will be able to decrypt the timestamp in order to trust Alice. At the end, Bob will reply with an incremented timestamp in order to get authenticated by Alice.

A common question is: ok with TGT I avoid to retype every time the password. But what about the Session Key $S_A$? The difference is that the session key is used by Alice (stored in RAM, not in Secondary Storage) in that session and after logout (or some fail of the system) such key is lost, so it has to be performed another login.

*"Skip slides 17-18-19"*

Authenticator is often the encryption of a timestamp. Authenticator is used to avoid the replay of messages sent to the same Server and of course such messages should be constructed by some framework for granting the Data Integrity.

Authentication Server (AS) and Ticket-Granting Server (TGS)

Are KDC and TGS the same? They are separate entities for historical reasons. Commonly the Server who provides the TGT is the Authentication Server (AS) and the users interact with it once a day (or once until the TGT si valid), while the Server who provides the service tickets is the Ticket-Granting Server (TGS). Obviously all this is nomenclature, what is important is the concept. The important thing is that we understand that the Server who provides the TGTs and the Server that provides the Service Tickets are different entities.