*"Do you like cybersecurity? What if your grandmom asks you "hey son, tell me what is this stuff? Cybersecurity?" how you explain that to some people that are completely unaware not about security but about information technology? Can you define it? Think about that, we will be back to this topic in order to talk to all kinds of people."*

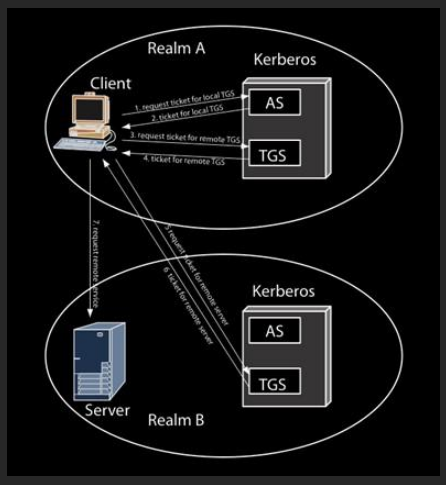
## Kerberos: Realms

There are several cases in which different networks need to cooperate, that is all of a sudden an user of one network wants to use a service of an user belonging to another network. Is it possible? Yes, **Realms** is a way to implement that.

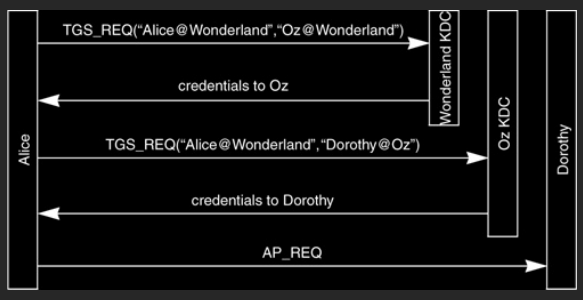
A Realm is like a domain, a network where there are users, KDCs and the **Master KDC**. This environment is very common in large enterprises. Each Realm has a different master KDC (we have seen that in addition of a KDC there can be other KDCs, for example read only KDC for better performances), what is important is if you have let's say 2 realms, your master KDC should be a principle of the master KDC of the other realm and vice-versa. This is the requirement. This means that there must be some key agreed between different master KDCs. You can setup the things so that you have just one master key shared between all possibles KDCs, this is actually unrealistic.

A very critical part of using a service in Kerberos is beeing authenticated, but who can authenticate a user? Only KDC of the same Realm! So once the authentication is made locally, then the user can ask some service coming from another Realm. The main scheme is the following:

Suppose we have a client in the Realm A which wants to use a service provided by the server in Realm B. First of all the client authenticates to its local KDC (in particular to the AS). Once authenticated, the client gets the TGT which will be used to ask to the local TGS for the ticket in order to communicate to the KDC of the other realm. Once obtained the ticket, the client can talk with the TGS of the other realm in order to obtain the ticket for the server. Why this? First of all, KDCs store informations only about users of the local realm, so first of all the client has to authenticate locally. Then a TGS can provide tickets only for local services or ticket for TGSs of the other realms, not for services of other realms. All this for scalability reasons of course.




### Another example




Alice's Realm is *Wonderland* while Dorothy's Realm is Oz. When Alice needs a service from Dorothy, she needs a ticket for Dorothy. Who is in charge to give tickets for Dorothy? The Ticket-Granting Server of Dorothy's realm. So Alice issues a request to her local TGS in order to talk to the TGS of Oz Realm. Then she asks to Oz's TGS for the ticket for Dorothy.

Everytime the network grows and it is needed to add another Realm, every KDC must know about that Realm, so there is a linear number of keys to manage for every new Realm. This is why we have a quadratic effort and it is not so nice. Kerberos 5 has fixed this issue, it has introduced the hierarchy of Realms, but we do not want to go in further details.

Expert analysts keep saying that Kerberos is not secure, it has a point of weakness that is the following: the compromission of KDC means that the attacker can impersonate every principal of the Enterprise, so this thing makes us think that Kerberos does not rely on the security where if there is a failure on some sector only that sector will be affected. The compromission of KDC means the compromission of the whole system. With Realms we close the topic Kerberos.

*"Let's remark the path we are following: after **confidentiality** and **data integrity** we started **authentication**, talking about challenges/response approach based on symmetric keys, then we considered the introduction of a third party, practical consequences, Kerberos and so on. Now we still have to check and understand other strategies in order to achieve authentication. We are not done yet!"*

## Authentication based on public keys

The basic problem, that we have already seen, is: if we want to setup authentication based on public key we prepeare a message, we do something on it like encrypting it with our private key and that's the tag, then if the receiver can decrypt with our public key he authenticates us. The main problem is *can you trust the public key*? How can Alice be sure that the public key stored on Bob's site is really associated with Bob's identity and it is not the public key of the attacker? The solution is to rely on a trusted third party.

**Needham-Schroeder public key authentication:** Needham-Schroeder also tried implementation of authentication through public key. The scheme is simulating the presence of a *trusted authority* that provides public keys to users. In order to be authenticated Alice can ask the authority "I am Alice, I want to talk to Bob" and the reply is the identity of Bob, the public key of Bob, and the digital signature of this whole message (so the encryption of the whole message by means of authority's private key, so if we succeed in decrypting with authority's public key we can trust the content of the message). Alice checks the digital signature, then generates a *nonce* N and sends to Bob the nonce and her identity, encrypted by means of the public key of Bob. Now Bob can decrypt the message (by means of his private key), but then he wants to check the identity of Alice, so Bob is just doing what Alice has done, so he sends to the authority "I am Bob, I want to talk to Alice". So Bob is requiring the public key of Alice. The authority sends to Bob the identity of Alice, the public key of Alice and the digital signature of the message. Then Bob checks the digital signature, gets the public key of Alice and generates the *nonce* N' and sends to Alice both the *nonce*sencrypted by means of Alice's public key. Alice checks and sends to Bob his *nonce* encrypted by the public key of Bob. After all these steps, *both the parties are authenticated*.

The scheme seems natural, but there can be an attack against it.

**Needham-schroeder public key *attack*:** it is a man in the middle attack. Just like Alice and Bob, Trudy can talk to the server. The attack is made of 2 sessions. The first session (R1) is the authentication between Alice and Trudy, the second session (R2) is the authentication between Trudy (that pretends to be Alice) and Bob. The attack can have place only on a certain precondition: Trudy must be able to issue an authentication from Alice to Trudy. Once Trudy has succeded to convince Alice to issue an authentication with her (Alice sends $K_{PT}(N,A)$ to Trudy), Trudy takes the content and sends to Bob $K_{PB}(N,A)$. So Bob will reply with $K_{PA}(N,N')$. Of course Trudy cannot decrypt this, so she forwards to Alice this message, then Alice will decrypt and will send to Trudy $K_{PT}(N')$. Trudy just obtain N' and sends it to Bob encrypted with $K_{PB}$ winning Bob's challenge. So it is just a man in the middle attack where Trudy forwards messages and acts in the middle. The only required thing is that Trudy is able to force Alice to issue an authentication with her, and if they belong to the same enterprise or something like this it is not so difficult because Trudy maybe has some reasons to invite Alice to be authenticated by Trudy.

There is so a fixed variant, that is quite similar to the previous one. The only thing that changes is that when in the second session Bob wants to be authenticated by Alice, he sends $K_{PA}<B,N,N'>$ and we immediately get

that the previous attack does not work because Trudy cannot send to Alice the identity of Bob while she is talking to Alice, because Alice does not know the presence of Bob.

We have not introduced yet the *definition* of *Digital Certificate*. The Digital Certificate is the message, sent by the trusted autorithy C, which contains the public key of someone togheter with the *digital signature* of this message.

When Alice asks to the authority C for the key of Bob, and the authority replies with $<K_{PB}, Sig\_C(K_{PB},B)>$, this message can be considered a Digital Certificate that certifies that $K_{PB}$ is the Public Key of Bob.

# X.509 STANDARD

In this standard 3 authentication protocols have been defined: *one-way authentication*, *two-way authentication* and *three-way authentication*. Remark that they define protocols, way of doing things, but do not specify encryptors or hash functions to be used because they change in the time.

One-way authentication protocol: is very simple, if Alice wants to be authenticated by Bob she sends a single message. The message contains: the digital certificate of Alice (taken by the trusted authority), a message $D_A$ containing somewhat, and the digital signature of this message. The message $D_A$ contains:

- Timestamp $t_A$
- The identity of Bob
- The session key $K_{AB}$ encrypted by means of Bob's public key $K_P$

Note that Alice has to know in advance the key of Bob, so we can think that Alice is the client and Bob is the Server. Moreover this scheme is not symmetric.

Actually there is some ambiguity about the names of these protocol. Indeed, with one-way authentication means *one-message-based*, two-ways authentication means *two-messages-based* and so on.

Two-way authentication (mutual authentication) protocol: there will be a message from Alice to Bob (A → B) and a message from Bob to Alice (B←A). It is similar to the one-way case, but here Nonces are used. In $D_A$ and $D_B$ will appeare something like $P_B(k)$ (for Alice) and $P_A(k')$ (for Bob). They are some **proposed** session keys. This because it is not rare to encrypt messages A→B with a session key and B→A with another session key. Both parties know both session keys. Notice: in the first message, Alice is sending the proposed session key encrypted by the public key of Bob, she is encrypting it without having the digital cetificate of Bob, this is not realistic. It becomes realistic if we accept that the 2 parties are not completely symmetric (like a client and a Server). We should assume that Alice has a good reason to know the identity of Bob.

Three-way authentication protocol: it is meant to be robust against replay attacks. The scheme is A→B, B→A, A→B. In the first message Alice is providing the message, her digital certificate, and the digital signature of the message. The message ($D_A$) is containing several components, a proposal of a session key, the identity of Bob, a *nonce* and a timestamp O. Timestamps are represented by O because they are **optional** (the protocol can also be implemented without them). The replay of Bob will be similar, it will contain another *nonce* and the confirmation of the proposed session key sent by Alice. The last message will be a message from Alice to Bob containing the two *nonces* and the identity of Bob, all of this signed by a digital signature.

# PKI: Public Key Infrastructure

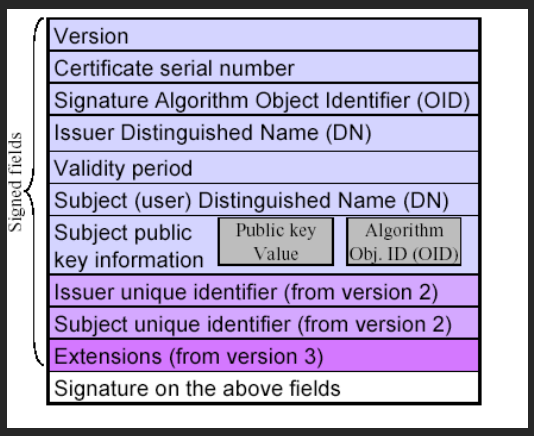
Certificates are provided by **trusted** Certification Authorities. The specific Certification Authority provides certificates just for the users in its domain. When someone asks for the public key of an user, the CA replies with the certificate containing the public key of the requested party and this certificate is signed by menas of the private key of the CA. This also means that we have only to remember just one public key, that is the CA's public key.

There are 2 types of CA's. One type is the legal one, it means that it is an organization registered in some public register and it is approved as a Certification Authority, and its certificates are valid wherever (actually there can be some countries where that CA is not trusted). For many countries there are official certification authorities that are trusted after a process of analysis… But what happens if such legal CA's are used for certifying a public key of some user and such certificate is sent abroad in another country where that CA's is not trusted? Just unknown! Should you trust it or not? It's up to you! There is no general rule.

Anyway you can consider the particular scenario where you are developing some application, that application is working in your local network, protected from the outside world, so what is happening within the local network. Then you can build a private CA that is meant to work within the local network.

So at the end there are public CAs and private CAs.

The X.509 standard describes how a digital certificate should be prepeared, in particular X.509 describes the structure of a Digital Certificate. The scheme is the following:




The certificate is composed by many fields. Why such a high number of fields? The reason is that many fields are related to the management of such digital certificate, the practical management. For instance there is the **serial number** field: every certificate is having a **serial number** that is unique among all the digital certificates of the same CA. Then there is the **validity period** that doesn't need to be explained. There is also the name of the user (the identity), the public key of the user, and information about the algorithm in order to use that public key. The *extensions* field can contain extensions, that are additional fields.

All the fields are signed by the certification authority. How to check the digital signature? You need the public key of the CA. Who is giving you it? Another CA. But you want to trust also that CA. It generates a chain. This is a problem. We will be studying how to solve it.

### X.509 certificate's fields

**VERSION**: there are three versions, version 1, version 2, version 3. If version 1 is used we find **0** in this filed, if version 2 we find **1**, if version 3 we find **2**.

**SERIAL NUMBER**: integer number cuncatenated with the CA's name. This is a unique identifier.

**SIGNATURE ALGORITHM**: it describes the algorithm used for signing this certificate

**ISSUER:** it is the name of the CA creating the certificate. It is not just a name but it is a name that follows the X.500 standard. We do not go in further details.

**VALIDITY**: it contains 2 subfields, the time the certificate becomes valid and the time for which after it the certificate is not valid anymore. So it is an interval of validity.

**SUBJECT:** name (according to X.500 standard) of the entity whose key is being certified.

**SUBJECT PUBLIC KEY:** it contains two subfields that are the algorithm with which the public key has been created, and the public key itself.

…

# Hierachy of CAs

Now let's come back to the problem of how Alice and Bob can check their certificates. If Alice and Bob use the same CA there is no problem. What if they do not use same CA? This can be a problem, in general the CA's form a hierarchy, for example it may be that the public key of Alice is certified by the certificate provided by Alice's CA, and the certificate provided by this CA is certified by Bob's CA. Note: certifying a certificate of a certain CA means providing (within a certificate) the public key of that CA. View slide 21.

### Certificates Revocation Lists (CRL)

Remember last time we were talking about certification and recovation, we insisted about the need of revocate certificates for several reasons. First reason is that CAs normally do that as a business. This is also a good practice because maintaining keys for a long time is somewhat helping the attackers. Other reason is that private key is been attacked and this is very important, in this case this thing is very effective in the sense that this issues a revocation of a certificate. Another reason is that somebody having the certificate he will change the function (the job) so digital certificate is revocated by the authority. In order to implement this whole feature there are the so called *certification revocation lists (CRL)*.

The CRL has a certain format, and this format is specified again by X.509 standard. There is an importan part of the structure, this part is in the middle, where there are the entries. Each entry has the *Certification Serial Number*, the *Revocation Date*, and the *CRL entry extensions*. The last fields is the signature of the digital certificate.

There are 2 fields that have to be explained:

**THISUPDATE**: the date of the publication of the list.

**NEXTUPDATE**: the expected date of the next CRL to be issued.

## OCSP

It is a new protocol, an alternative tu CRLs because when the digital certificate you are obtaining from the website (we talk about websites but these things hold for every authentication based digital certificate), it is possible to use informations coming from the digital certificate. If the certificate is Version 3, there is a certain extension where the browser can get an url and connect to it in order to make queries, and the main use is to make a query to get the CRL of that CA. This is done according to this protocol OCSP.

Extended Validation (EV): it is a type of certificate that is offering more informations with respect to a normal one. You will prefere to use for your Server an EV certificate, also because in the web there are many organization that collect informations about the reputations of Servers and services, and of course who is offering EV certificates will get a higher reputation.

*"When you are asking for a digital certificate, you have to provide the motive why you are getting it. What is the usage of such key? Server? Digital signature? Both? Authentication of a browser? Normally you get a certificate for **one** usage, so if you want to be able to make digital signatures and to setup your Server for multiple usage you will need multiple digital certificates."*

## PGP: PrettyGood Privacy

It is something used for security of emails. It is used at application level. I want to mention a interesting point about **PGP**. The idea of its creator was “I don’t like certification authorities, i want a distributed approach”. He believed in a *Web of Trust*, that is something that is proving the correctness of the associations between a public key and an identity not by means of a digital certificate issued by a CA but by means the trust obtained from some other users. The creator did not want centralized authorities, but a distributed approach. Imagine I generate a pair of keys, I show on my website my public key and I ask for visitors of the website “This is my public key, import yours in you key ring and sign my public key by means of your digital signature”. Key Rings for PGP are allowing that. Normally when you do that you import a public key that is associated with an email address, in some cases also a name, but the strong association is public key and email. There are also PGP Servers where you can upload you key ring. When you need the public key of somebody because you want confidentiality and you don’t know such public key what do you do? You can try several path, some website, if I find on a website can I trust it? One solution is asking a PGP Server for that key, PGP Server tells you also the number of digital signatures for each public key, so that if the number is low you do not trust them, if the number is high you are lead to trust them. So the problem is starting, at the beginning with low numbers nobody trust nobody, the numbers have to grow up. This is building the **Web of Trust**.