Alin Tomescu, CSE408
Tuesday, April 12th, Lecture #20

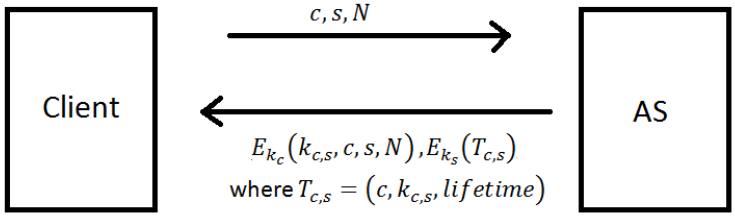
Kerberos v5 and PKI

Kerberos v5

Three main building blocks:

- protocol for initial ticket
- protocol for additional tickets
- protocol for server connection setup

Protocol for initial ticket



The client extracts $k_{c,s}$ from $E_{k_c}(k_{c,s}, N)$

There's going to be a user that sits down at a client. The client talks to the KDC (the AS), which has a database of all the users and their keys, and it also has a database of all the servers on the system and their keys.

User will type in their username c and their password $pass$, and the client will compute $k_c = hash(pass)$, and send a message to the KDC saying that user c is talking to him and he wants a ticket for server s. The client also puts a nonce n in the message. Server uses c to look up k_c and he also looks up k_s from s. He generates a session key $k_{c,s}$ and responds with $E_{k_c}(k_{c,s}, N), E_{k_s}(T_{c,s})$, where $T_{c,s} = (c, k_{c,s}, lifetime)$. This message is missing the name of the client and server, which violates the "explicit messages" principle. Even though the nonce N is associated with c, s and AS, apparently that won't suffice.

Issue: $E_{k_c}(k_{c,s}, N)$ can be brute-forced offline and an attacker could find a client's password.

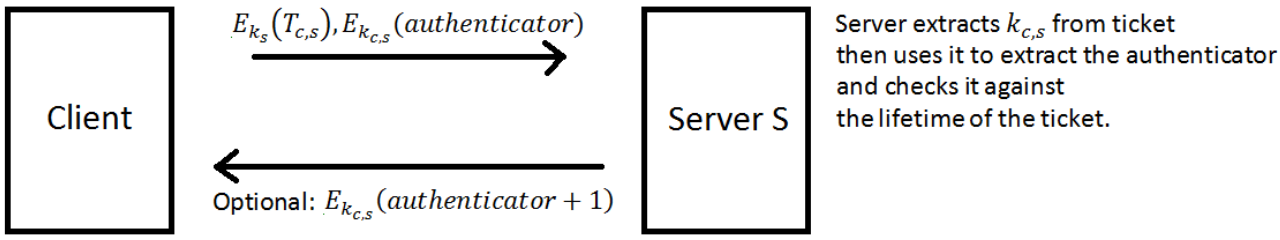
Issue: Bad guy can change $S\ to\ S'$ (as the client sends his first request) and he can "force" the client to talk to S' instead of S. To fix this, $E_{k_c}(k_{c,s}, N)$ should be $E_{k_c}(k_{c,s}, c, s, N)$. Add the c in there just to be sure multiple clients are dealt with securely as well.

The client extracts $k_{c,s}$ from $E_{k_c}(k_{c,s}, N)$.

Protocol for client-server connection setup

Now the client and the server have to set up their connections. Server has k_s, client has $k_{c,s}$ and $E_{k_s}(T_{c,s} = c, k_{c,s}, L)$

Alin Tomescu, CSE408
Tuesday, April 12th, Lecture #20



Client sends $E_{k_s}(T_{c,s}), E_{k_{c,s}}(authenticator)$ to the server, where authenticator is a timestamp and checked against the lifetime of the ticket. Server extracts $k_{c,s}$ from ticket and then uses it to extract the authenticator and checks it against the lifetime of the ticket.

Optional: Server responds to prove that it knows $k_{c,s}$ with $E_{k_{c,s}}(authenticator + 1)$.

Protocol for additional tickets

If you are a client and you want more tickets, you don't want to go back to AS. There are subsidiary servers that are there to issue more tickets.

Client has $k_{c,tgs}$, $E_{k_{tgs}}(T_{c,tgs})$ (obtained from the AS, by running the initial ticket protocol with $s = tgs$)

TGS has hi k_{tgs} key and the table of all the servers and their keys that the KDC had: $server \rightarrow key_{server}$

Client does the connection setup protocol above with the TGS. At the end of it the TGS will know $k_{c,tgs}$ and the client's identity c.

Then the client says to the TGS "I want to talk to s, here's a nonce N", by sending s, N and the TGS responds $E_{k_{c,tgs}}(k_{c,s}, s, N), E_{k_s}(T_{c,s})$

Overall protocol

User enters his ID and password into the client, the client will engage the initial ticket protocol with the KDC/AS for a TGS server to obtain $k_{c,tgs}$ and $E_{k_{tgs}}(T_{c,tgs})$. Then the client will perform the additional ticket protocol with the TGS for some server s (this will perform the connection setup protocol between c and the TGS). At the end of it c and the TGS will have the shared key $k_{c,tgs}$ and c and s will have $k_{c,s}$.

Public-key infrastructure

We've got Alice and Bob. Alice has pk_a and sk_a. Alice can't just send her pk_a to Bob because of a MITM attack.

Trent comes into the picture. Alice has pk_t and so does Bob. Also, Alice and Bob have their public and secret keys.

Somehow in an offline protocol, Alice is going to obtain

$$Sig_{sk_t}(\text{alice}, pk_a) = C_{ta}$$

C_{ta} is a **certificate from Trent of Alice's identity**. Bob has to trust Trent, and so does Alice because if Trent wants to pretend to be Alice, Trent can issue himself a certificate saying he's Alice. Trent is trusted in two different ways. Bob is trusting Trent not to impersonate Alice and to issue certificates correctly.

Now, Alice can safely send to Bob her public key as $m = \text{alice}, pk_a, C_{ta}$

Let $C_{ta} = \left(\text{alice}, pk_a, Sig_{sk_t}(\text{alice}, pk_a)\right)$ to ease notation.

So Alice just sends the certificate. Bob is now sure he's talking to Alice. If Bob wants to sign his response to Alice then he needs to send Alice a certificate C_{tb} so that Alice can get his public key securely.

A real certificate will have a lifetime. Lifetime can be a year or two years in something like SSL.

The issue in **public-key infrastructure** (PKI) is who is Trent, how many Trent's are there and how are they related?

The **trust relationship is different**. Alice has to trust Trent not to impersonate her. Imagine there are a Trent and a Terry. Alice's certificate could be signed by Terry, and Bob's by Trent. What matters is that Bob's certificate is signed by someone that Alice trusts, and vice-versa. So Alice needs to trust Trent and Bob needs to trust Terry.

The holder of a certificate doesn't have to trust the issuer, the receiver/verifier has to.

We don't all have to trust the same Trent. This can be interpreted in different ways.