CptS 464/564
Lecture 26

6th December 2004

Authentication with a shared secret key

Suppose A and B want to hold a conversation in which each knows it is talking to the other. If they share a secret encryption key, K, it is easy:

- A -> B: I'm A, let's talk
- B -> A: I'm game, prove that you are A by encrypting RB for me
- A->B: here's K(RB), now do the same for me by encrypting RA
- B->A: K(RA)

To carry on the conversation, A and B will usually set up a *session key* to be used to encrypt further messages. The reason is that keys become more vulnerable as they are used to encrypt more and more data. Using a session key limits the use of the valuable shared secret K.

Kerberos

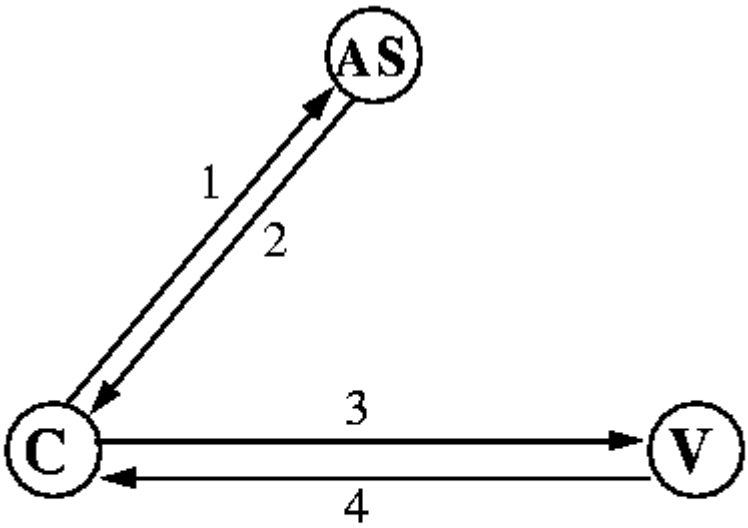
Kerberos is an authentication infrastructure for distributed systems developed in Project Athena at MIT in the late 1980's/early 1990's. It is used in AFS and in

Windows 2000 and later (with extensions by MS). Kerberos is a *shared secret* authentication system. It does not use public-key cryptography techniques. Rather, each *principal* (person, server, *etc.*) registers a secret (think password) with the authentication service which is used to verify the identity of the principal at a later time. (Remember that an authentication service is concerned with the establishing the identity of a principal; an authorization service is concerned with what an authenticated principal is allowed to do.)

Kerberos is, fundamentally, an implementation of the Needham-Schroeder authentication protocol. However, it contains additional protocols that solve a number of other useful and interesting authentication-related problems in distributed systems. The current version of Kerberos, version 5, is described in Internet RFC 1510, www.faqs.org/rfcs/rfc1510.html, released in 1993. A summary is contained in B. Clifford Neuman and Theodore Ts'o, *Kerberos: An Authentication Service for Computer Networks*, IEEE Communications Magazine, 32:9, pp. 33-38, September, 1994 (also www.isi.edu/gost/publications/kerberos-neuman-tso.html) from which the figures in these notes are taken.

Initial authentication

Initial authentication in Kerberos takes place between a client, C, and and the Kerberos authentication service, AS. The goal of authentication is to allow the client to confirm its identity to a particular service (verifier), V. The result of the interaction with the AS is a *ticket* that the client presents to the verifier.



1. as_req: c, v, $time_{exp}$, n
2. as_rep: $\{K_{c,v}$, v, $time_{exp}$, n, ...$\}K_c$, $\{T_{c,v}\}K_v$
3. ap_req: $\{ts,ck, K_{subsession}, ...\}K_{c,v}$ $\{T_{c,v}\}K_v$
4. ap_rep: $\{ts\}K_{c,v}$ (optional)

$T_{c,v} = K_{c,v}$, c, $time_{exp}$ ...

Steps in authentication

1. Authentication-service request: the client sends its own name, the verifier's name, the time at which it wants the resulting ticket to expire, and a random number

2. The AS responds with a *ticket* encrypted with the verifier's secret key. The ticket contains a *session key,* the client's name, and an expiration time. The AS's response also includes the session key, *etc.* encrypted in the client's secret key. Notice that at this point no authentication has actually occurred: anyone could ask the AS for a ticket using anyone else's name.

3. The ap_req message consists of two parts, the *authenticator* and the ticket. The authenticator contains the current time, a checksum, and other things. By decrypting the authenticator using the session key obtained from the

ticket, the verifier authenticates the client (if the checksums match). The role of the timestamp in the ap_req is to prevent replay attacks: the timestamp must be within a small interval centered on the *verifier's* current time and further, the verifier must not have received a request with that timestamp before. Note the requirement for approximate clock synchronization between client and server.

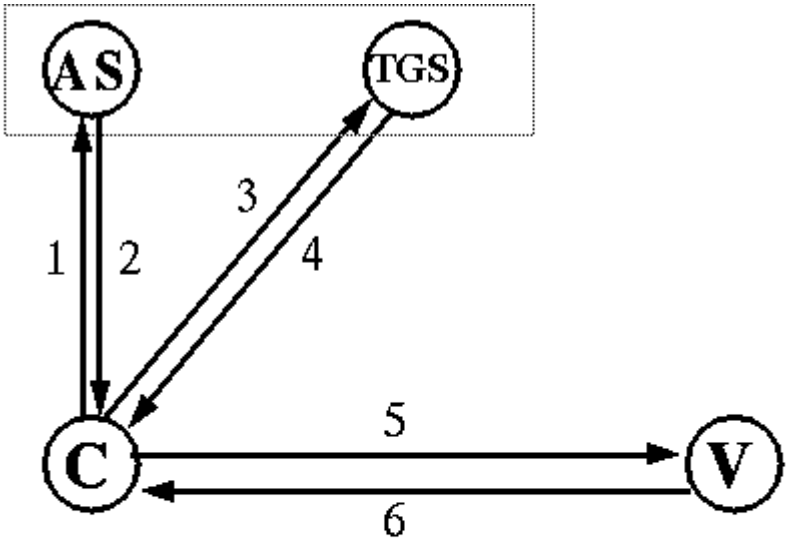
4. The verifier's response to the client allows the client to authenticate the verifier by confirming that it is talking to an agent that knows the secret key of the verifier.

A different ticket and session key is required for each client/verifier pair.

Improving the convenience of authentication

While the basic protocol above will suffice in theory, practical matters now intrude. The protocol above requires the client to have its secret key available in order to authenticate to each verifier. This poses either a substantial inconvenience (suppose you had to type in your password for each service you contacted) or a security risk if the password is stored for later use. Kerberos goes to some length to make sure that tickets have limited lifetimes, but passwords have much longer validity periods, so storing passwords is not a good idea (in the Kerberos philosophy, anyway). The goal in Kerberos is to cache only short-lived authentication information, the tickets and session keys associated with them (together called *credentials*).

To meet this goal Kerberos introduced the notion of a *ticket granting service* that leverages the client's initial authentication (via an AS) for the TGS into tickets for an arbitrarily large number of services.



1. as_req: c, tgs, $time_{exp}$, n
2. as_rep: $\{K_{c,tgs}, tgs, time_{exp}, n, ...\}K_c$, $\{T_{c,tgs}\}K_{tgs}$
3. tgs_req: $\{ts, ...\}K_{c,tgs}$ $\{T_{c,tgs}\}K_{tgs}$, v, $time_{exp}$, n
4. tgs_rep: $\{K_{c,v}, v, time_{exp}, n, ...\}K_{c,tgs}$, $\{T_{c,v}\}K_v$
5. ap_req: $\{ts, ck, K_{subsession}, ...\}K_{c,v}$ $\{T_{c,v}\}K_v$
6. ap_rep: $\{ts\}K_{c,v}$ (optional)

In the first two messages here the identity of the TGS takes the role of the verifier in the previous figure. After the second message the client has credentials for interacting with the TGS. This is called a ticket-granting ticket (TGT). For talking to other verifiers, the client now uses the TGS to obtain credentials, by sending a request much like the first message of the previous figure to the TGS, but the client is authenticated to the TGS (it wasn't authenticated to the AS) and the TGS issues further tickets based on this previous authentication without requiring the client's password again.

In practice, the TGS and AS are implemented in a single server called the KDC (Key Distribution Center).

Key assumptions

- secret keys must be kept secret: this is an obligation of both principals and the Kerberos infrastructure
- secret keys must be unguessable
- loosely synchronized clocks
- names of principals are not reused (at least in the short term). This is a consequence of the use of authentication for authorization. If a principal name appears on an authorization list you wouldn't want that name to be recycled for someone else's use before it was removed from the authorization list.

Additional useful features

- Tickets generated by the AS and by the TGS are distinguishable: some application servers (verifiers), such as one that allows passwords to be changed, may want to insist that the password be present at the time that they act on the client's behalf.
- Tickets may be *post-dated* (they become valid some time after they are issued): useful, for example, for giving credentials to batch jobs that run in the middle of the night. If a ticket was stolen while the job was sitting in the queue they would not be immediately useful. Furthermore, post-dated tickets have to be blessed by the KDC before they become valid so there is at least the opportunity to check a key against a revocation list before it can be used.
- Renewable tickets: as noted above for passwords, longer lifetimes of secret materials pose higher risks. Renewable tickets allow each ticket to be issued with a relatively short lifetime and then renewed upon expiration. Tickets can be checked against revocation lists kept at the KDC during renewal.
- Binding tickets to network addresses: additional (but weak) protection against use of stolen credentials.
- *Proxy tickets:* a client may request a ticket associated with a different client network address than that of the TGT itself. The result is called a proxy

ticket and can be used, for example, to let a print server access files on behalf of the client.

- *Forwardable tickets:* having authenticated to one workstation it would be nice not to have to use a password to use others. A forwardable ticket is a proxy TGT: it allows the recipient to generate additional tickets for the client at a designated network address.

- *Cross-realm authentication:* the namespace in which authentication is performed is called a realm. The realm name is part of each client and verifier name. If a client wants to use a service in a different realm it ultimately needs a ticket for that service which involves *some* KDC encrypting a ticket with the service's secret key. It would be unfortunate if every KDC had to contain secret keys for all principals and indeed Kerberos has a way to avoid this. We have almost all the mechanism needed: the architecture provides a way for a KDC to be authorized to issue TGTs for a different realms. Realms can be organized in a hierarchy or network that provides an authentication path between realms (tickets generated by this process record the authentication path so that recipients of a ticket can tell how it originated.