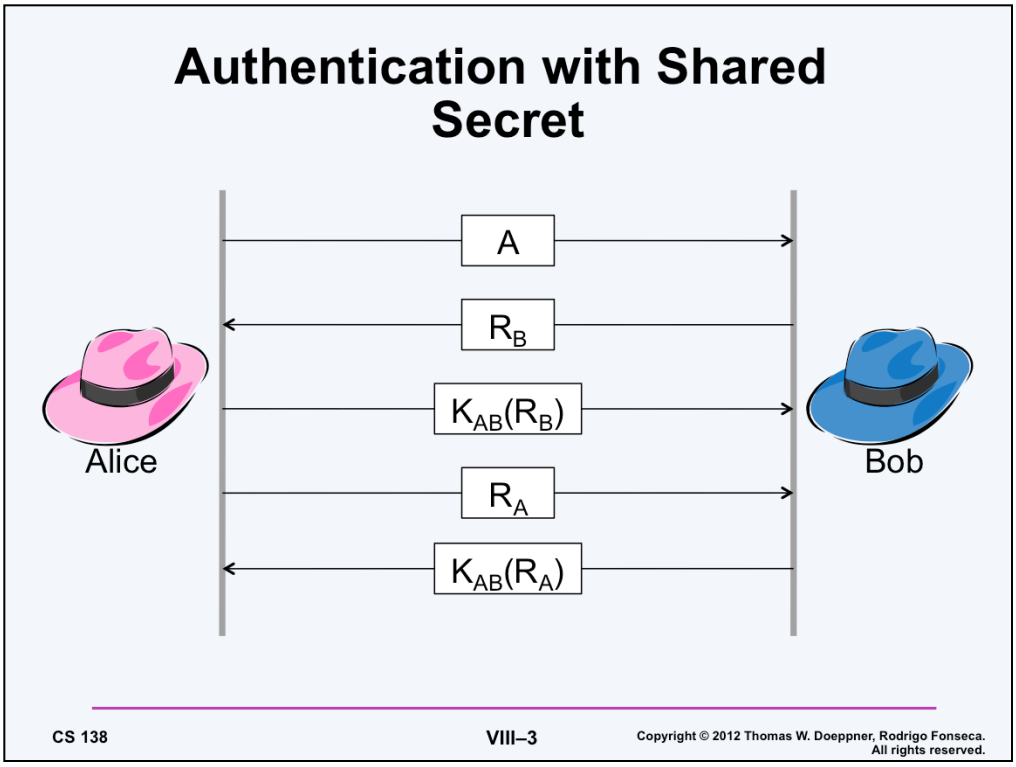
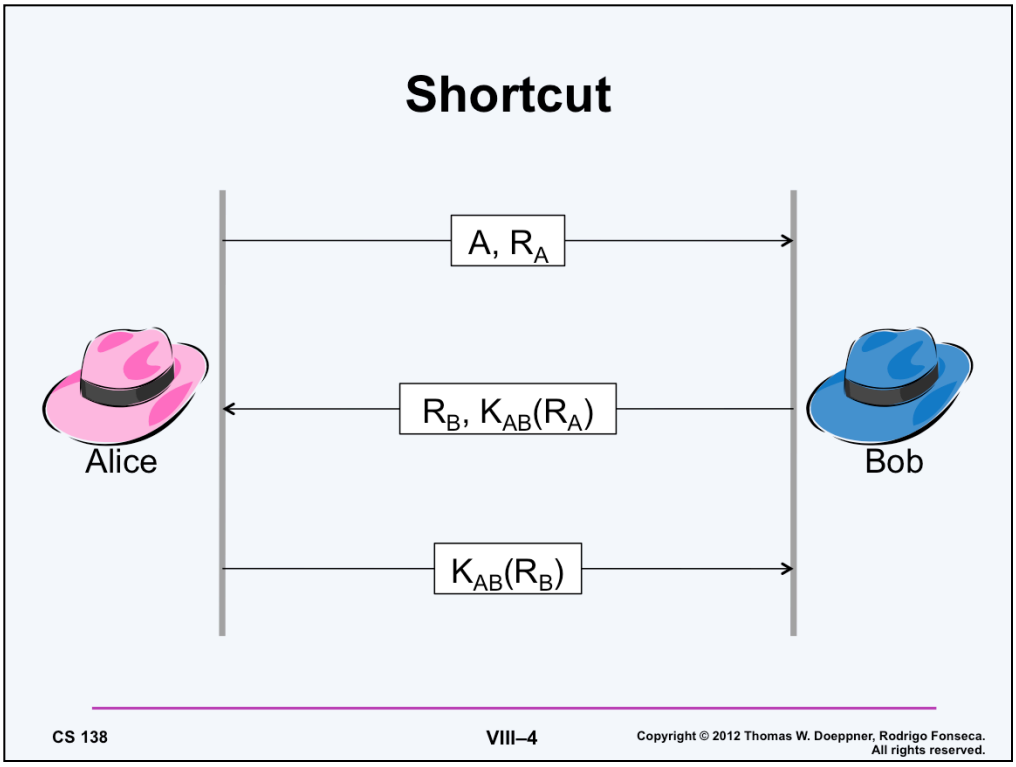
CS 138: Security II

Today

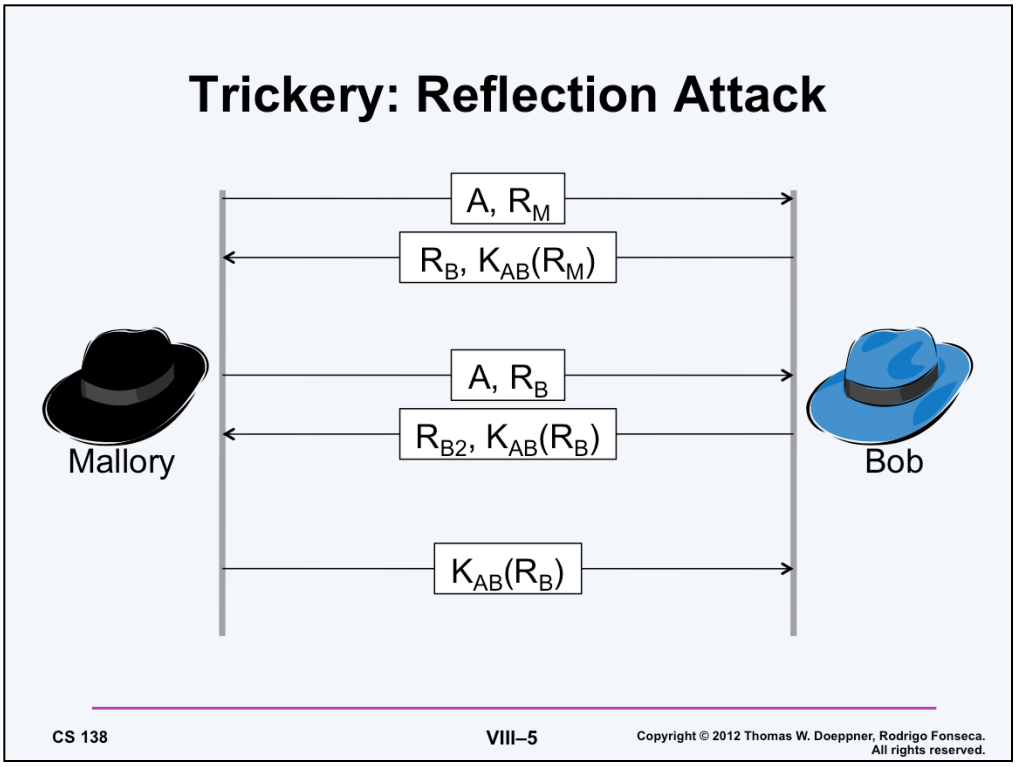
- Secure key distribution
- Authorization



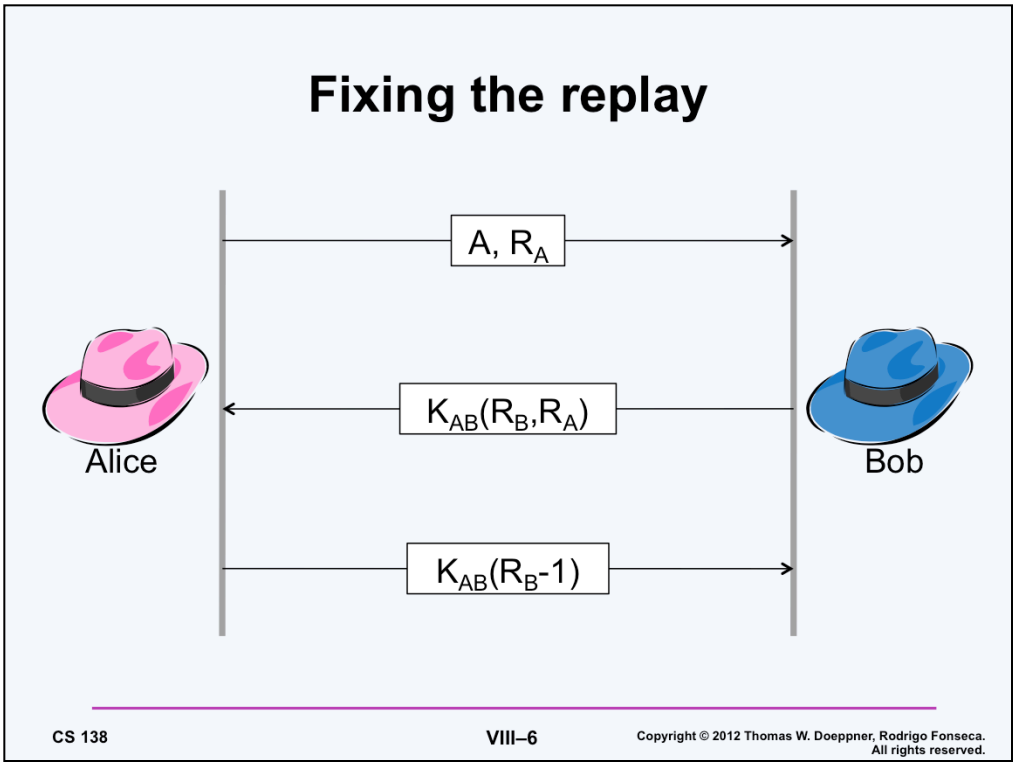
This example is adapted from "Distributed Systems: Principles and Paradigms," by A. Tanenbaum and M. Van Steen, page 398. Here K_{AB} is an encryption scheme based on a key known only to Alice and Bob. Using it, Alice is attempting to prove her identity to Bob, and vice versa.



Here we attempt to optimize the protocol.



Mallory masquerades as Alice. Bob sends her R_B. To prove her identity, she must send Bob $K_{AB}(R_B)$. Mallory doesn't know K_{AB}, and never will, but she gets Bob to provide her with $K_{AB}(R_B)$.



If we require Alice to do a simple operation with the (encrypted) nonce before sending it, Mallory cannot do it.

Problem: n^2 key pairs!

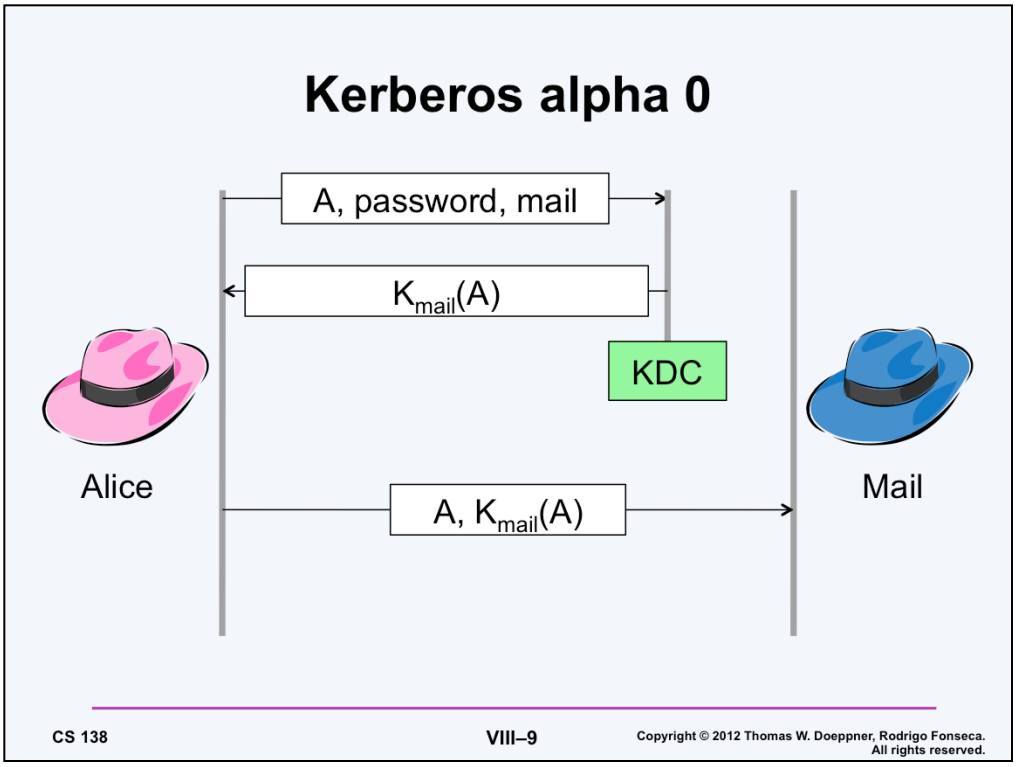
- Alternatives
  - Share keys with a key distribution service
  - Public-key cryptography

Kerberos

- Developed at MIT in the 80's
- Uses a Key Distribution Service (KDC)
  - Based on Needham-Shroeder key exchange
- Our description based on the "play":

  "Designing an Authentication System: a Dialogue in Four Scenes"

  http://web.mit.edu/kerberos/dialogue.html

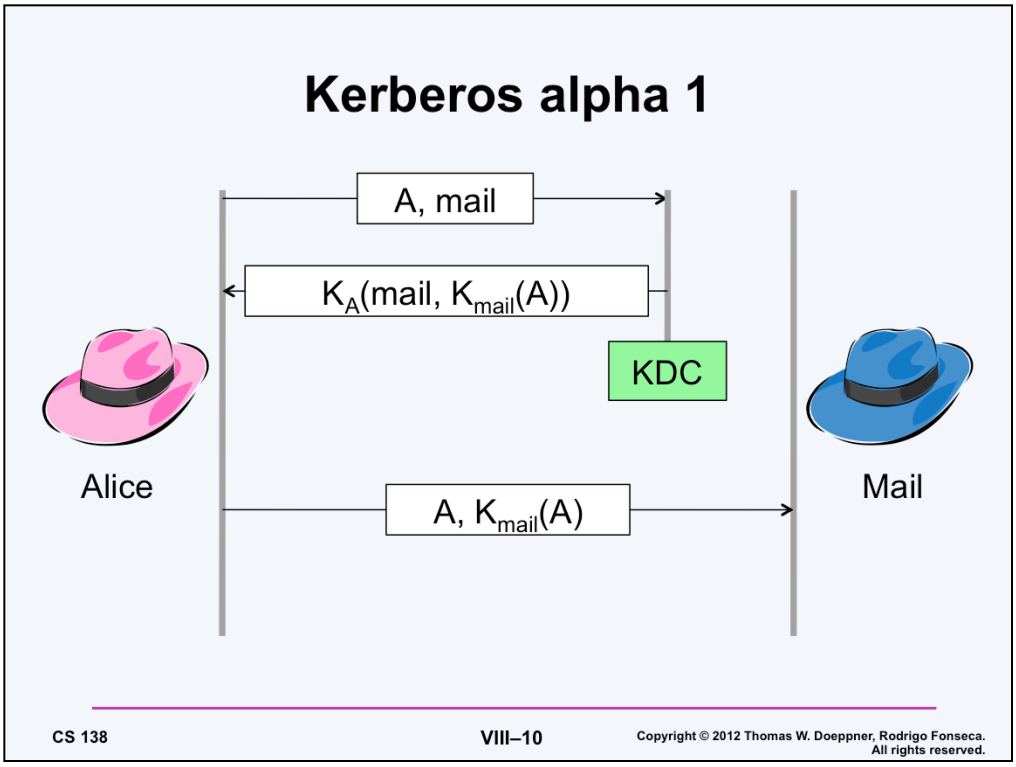


Alice tells the KDC she wants to communicate with the Mail server.
Now the Mail server will know that A has been authorized by the KDC, right?

What happens if I get a copy of the packet, and lie to Mail that I am A?
Fatal flaw: A's password is sent in the clear!

Another problem is that Alice has to enter her password any time she wants to talk to a new service, and get a new ticket.

Another problem is that an attacker can replay the message to the mail server, lie that it is A, and get access.



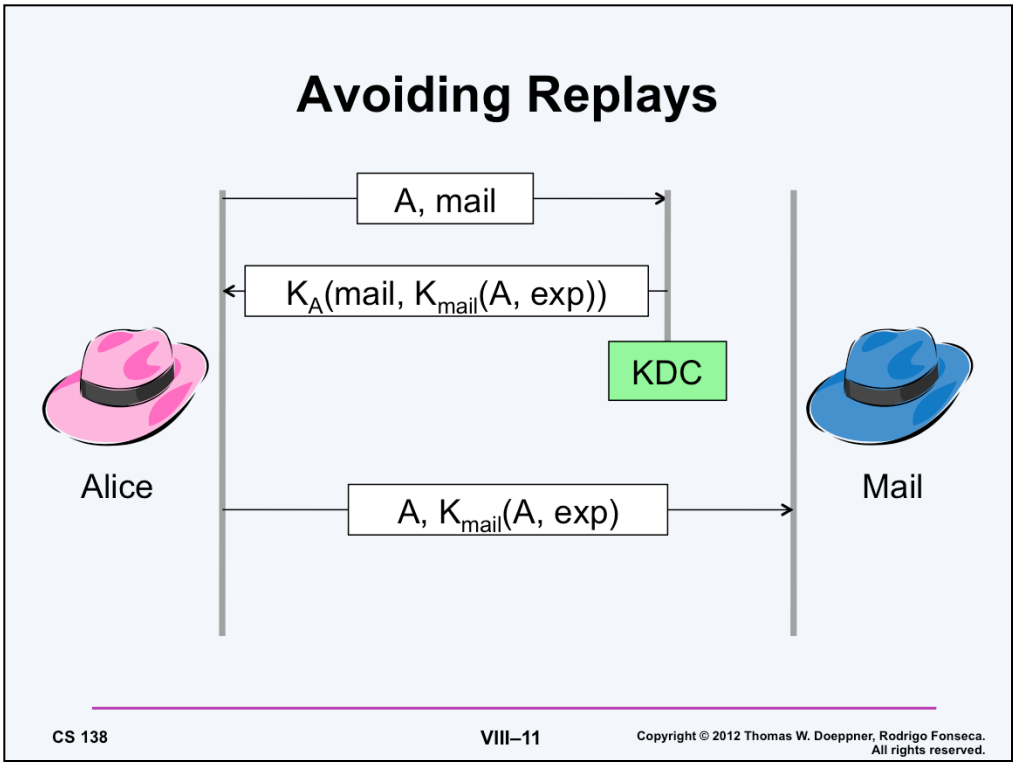
A and K share a secret, use this to solve the first problem, and to prevent eavesdropping of the KDC's response.

What we achieved: A is authenticated to the KDC.

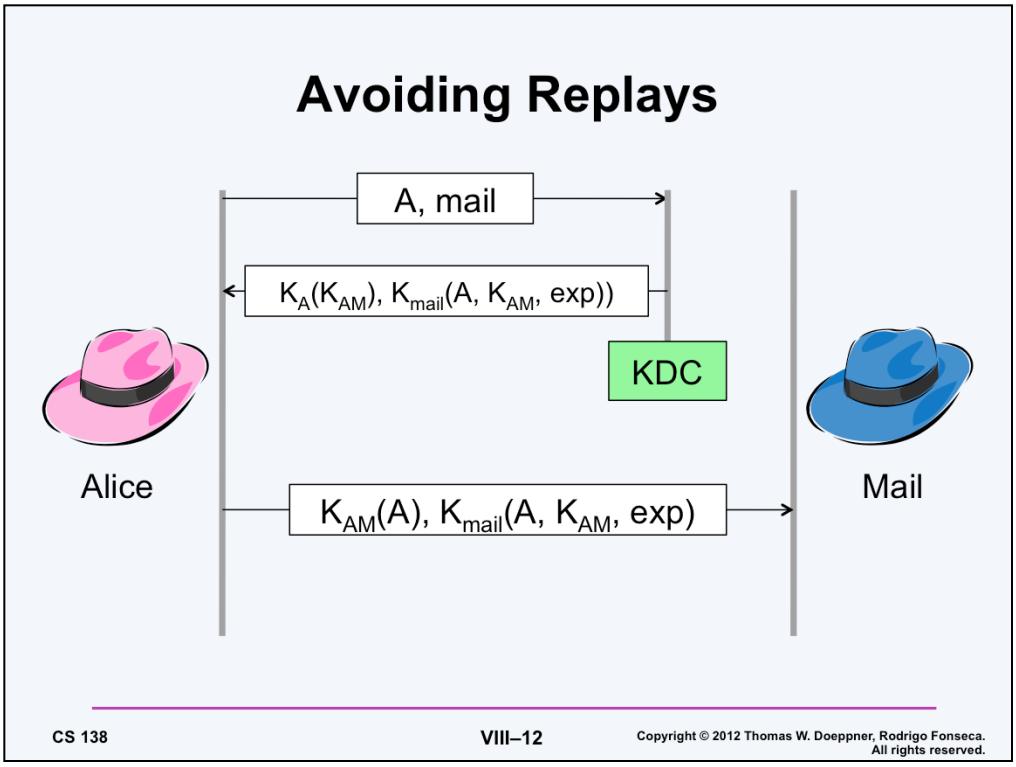
Now, there are still many problems with this scheme.

The mail server knows this, but does not know that A is actually A! (Anyone could claim to be A)

The attacker can still **replay** the ticket and get access!



Can add an expiration time to the ticket.
What does this give us?
The mail server knows that the KDC granted A, within exp, access to mail.
Either too short, and inconvenient, or too long, and insecure.

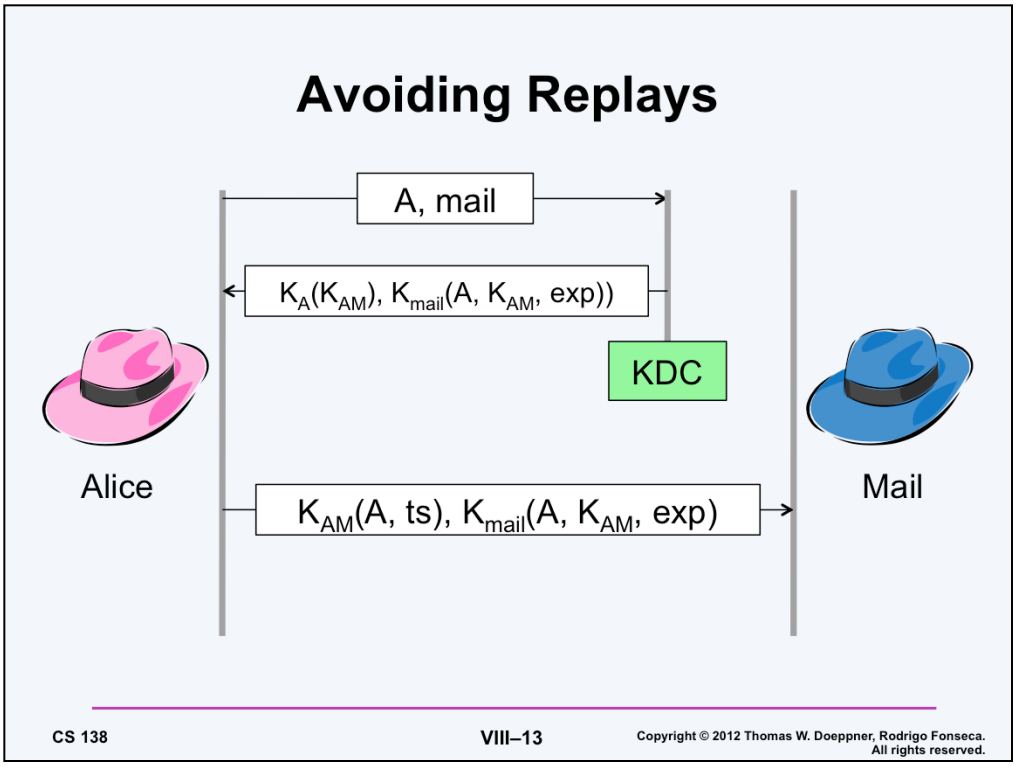


Ok, what does this give us now?

The mail server will know that I've been authenticated, and that I am really who I am saying I am! (only A could have decrypted K_{AM})

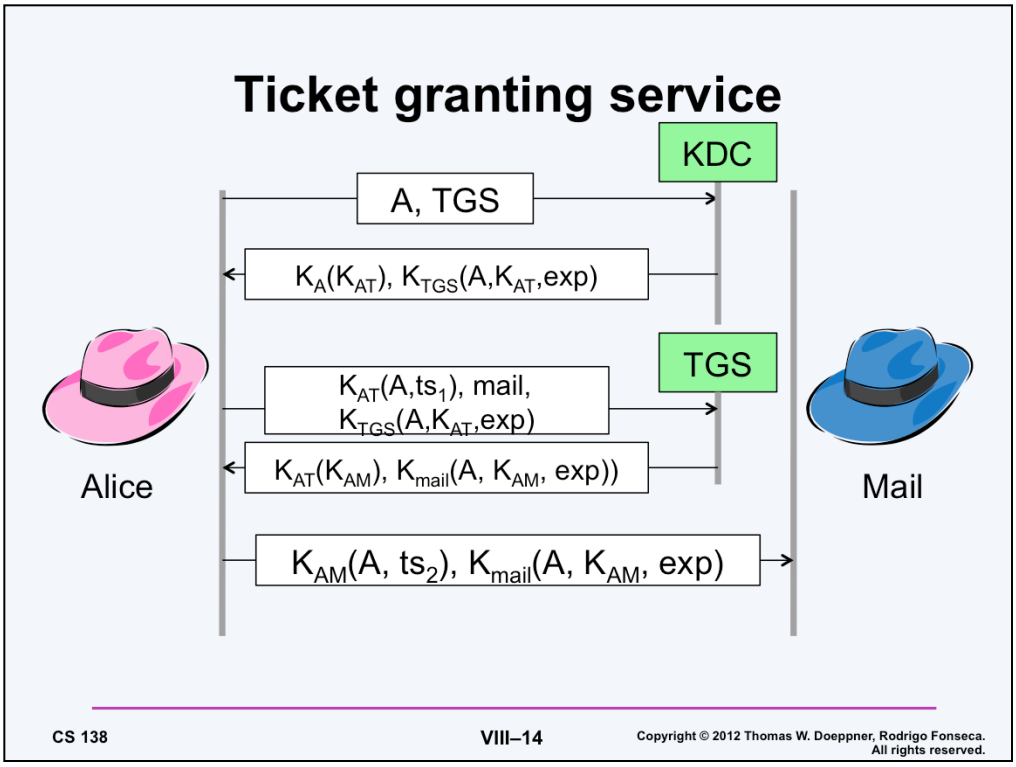
We also have a shared key and can establish secure communication.

Are we done? What happens if Mallory steals my message, can she replay it? Yes!



Ok, this is getting better.
But I still have to contact the KDC for each new service!

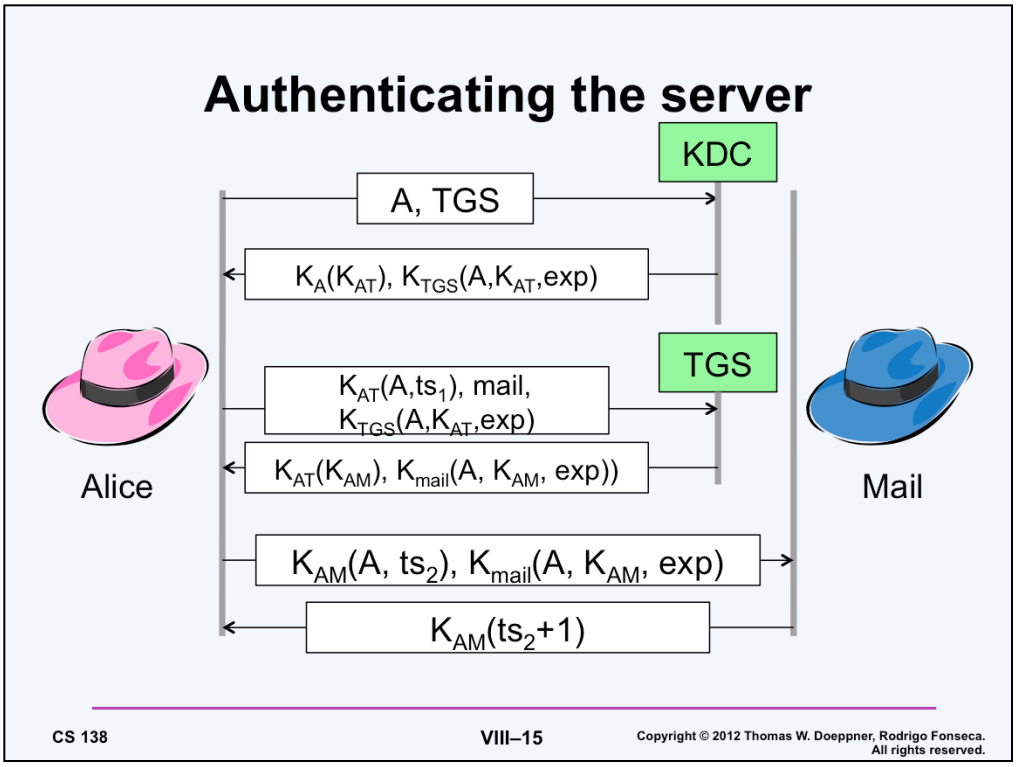
Solution: create a new service



Solving the multiple password entry: TGS
If you have the TGS ticket, A has been authenticated.

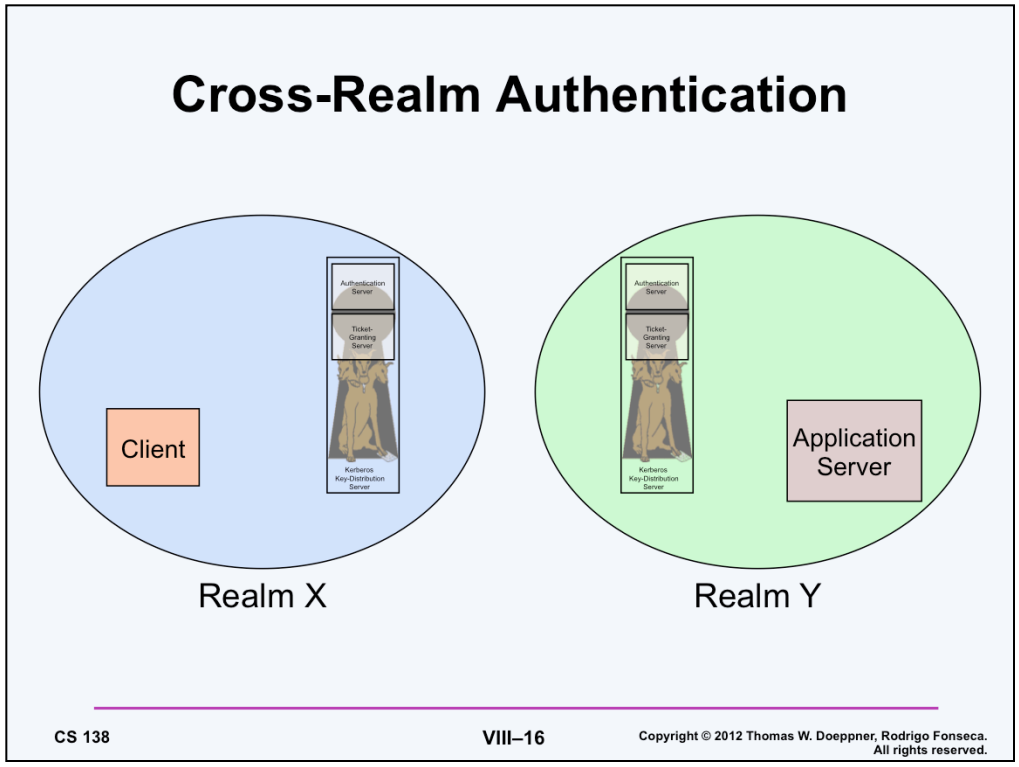
To enter a password only once, we create a Ticket Granting Service, which is a part of Kerberos that doesn't use passwords for authentication, but a Ticket Granting Ticket.

Are we done? No, the client still does not know this is the correct server!



Server authentication

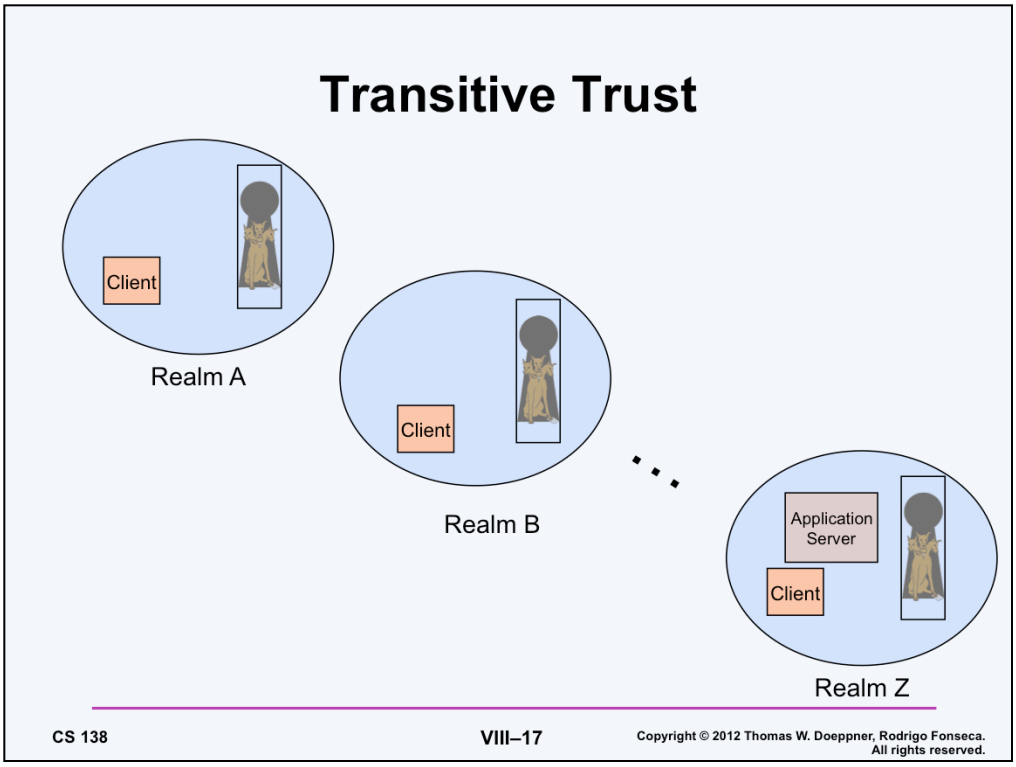
This is very close to the final version of Kerberos.
The steps are also similar to the Needham-Schroeder Key Exchange



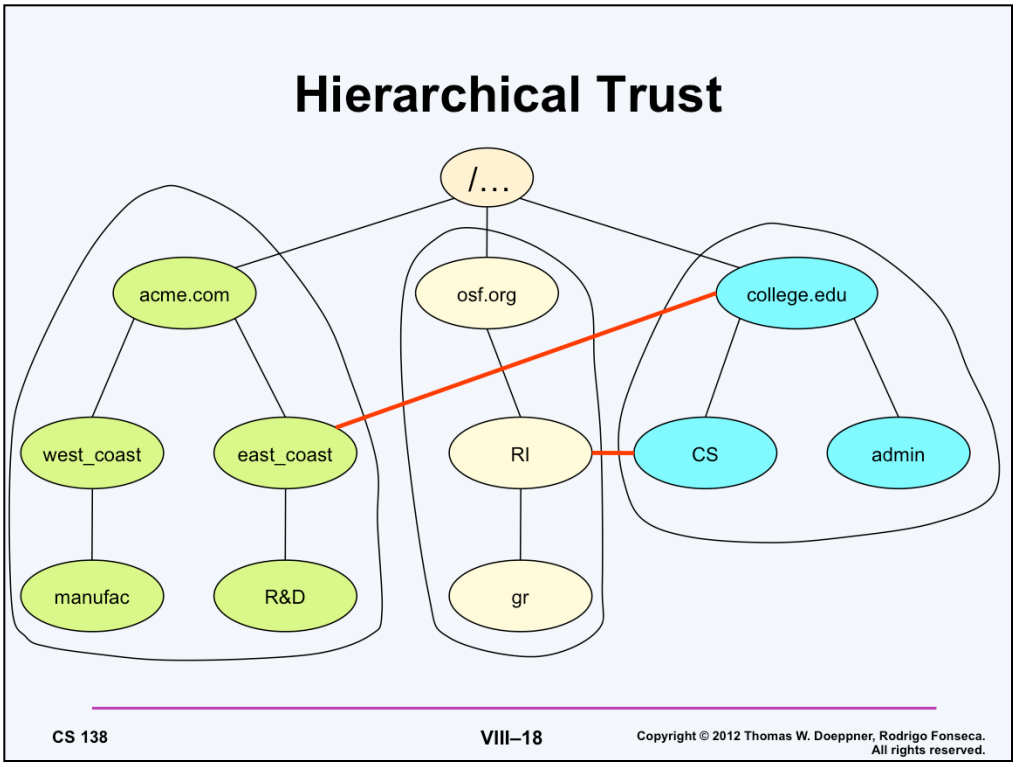
The basic Kerberos administrative entity is the *realm*. Each realm has a separate KDC, which know the passwords for those in the realm but not for those of other realms.

Authentication within a realm depends upon the fact that the realm's KDC has a copy of each party's password. But if one wants to make an authenticated request of some server in a different realm, there is no one KDC that has both party's passwords. Rather than require one to register one's password with a foreign KDC, the KDC's of the two realms agree to trust each other (to a limited degree).

The two KDCs each register an account for themselves in the other. A client in realm X attempting to access a server in realm Y must already have authenticated itself in realm X. Realm X's KDC then prepares a ticket for the client, encrypted in the key associated with the realm-X account in realm Y, which the client sends to realm Y's KDC. This ticket essentially says, "I, the KDC for realm X, do hereby certify this client's credentials. Since you, the KDC for realm Y, and I trust each other, please honor the credentials of my client." Y's KDC, on receipt of this request, prepares a ticket to the desired application server in realm Y and passes it back to the client (via realm X's KDC), which can now contact the application server directly using its certified credentials.



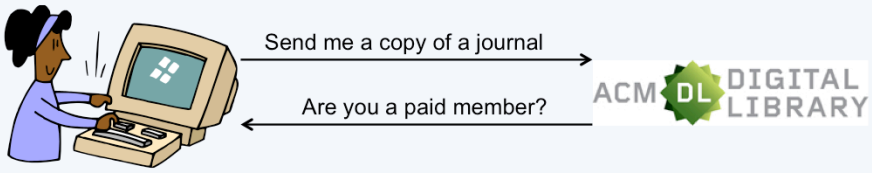
The problem with the cross-realm-authentication technique of the previous slide is extending it to deal with large numbers of realms. One approach is to set up cross-realm relationships with all pairs of realms, but this is clearly only practical with small numbers of realms. One is tempted to use a transitive trust technique, but one may end up trusting realms one had no intention of trusting.



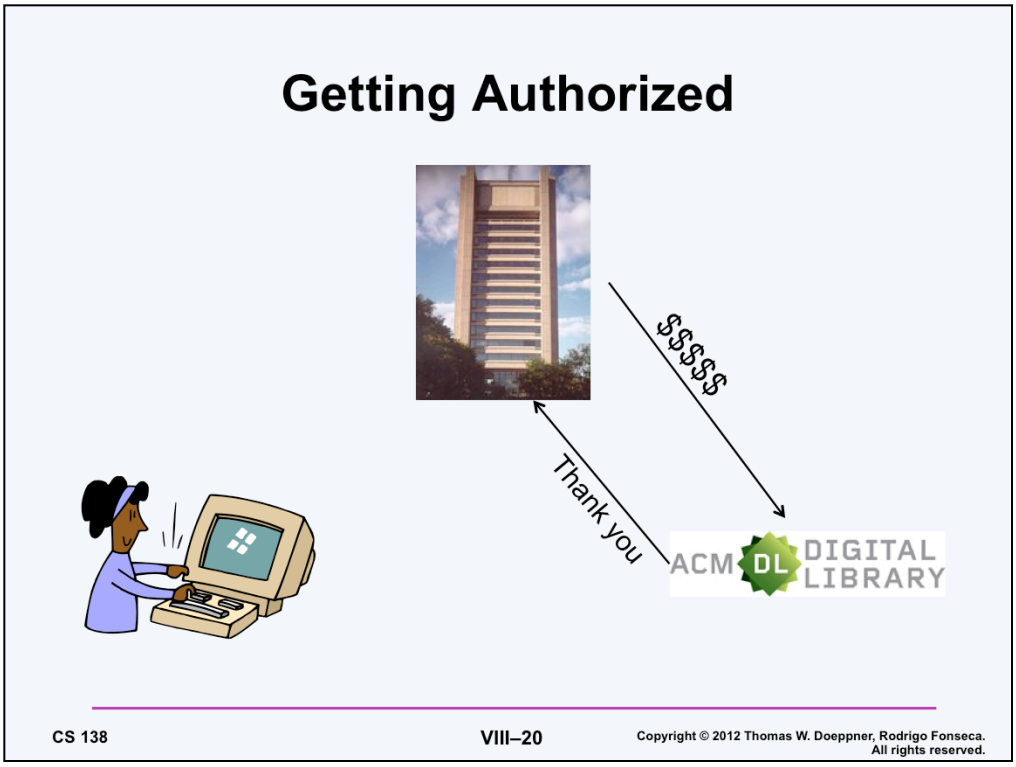
A better solution to extending cross-realm authentication to large numbers of realms involves what is known as hierarchical trust. In this picture, each of the nodes represents a complete realm. The root of the tree and the links to its descendants are fictitious (they are there strictly for convenience of naming), but the subtrees are real. In this case, three organizations have each arranged their realms in a hierarchical fashion. Within each subtree there is a cross-realm relationship established between adjacent realms. Within each organization's hierarchy, transitive-trust relations may be used. For example, the realms /…/acme.com and /…/acme.com/east_coast have a cross-realm relationship, as do /…/acme.com/ and /…/acme.com/west_coast. Though west_coast and east_coast have no direct relationship, they may use transitive trust since they are in the same hierarchy.

For communication across hierarchies a direct relationship must be established. For example, one has been set up between /…/college.edu/CS and /…/osf.org/RI. We would like to exploit transitive trust here; /…/college.edu/CS and /…/osf.org/RI/gr may use transitive trust for authenticated communication. However, we do need to restrict these relationships so that no undesired trust relationships occur. So, once a cross-hierarchy trust link is crossed (such as between CS and RI), one may go down in the hierarchy but not up. Thus no trust is provided between CS and /…/osf.org. Furthermore, any one trust path may contain no more than one cross-hierarchy link. Thus /…/acme.com/east_coast may communicate authenticatedly with /…/college.edu/CS, but not with /…/osf.org/RI.

Getting Authorized

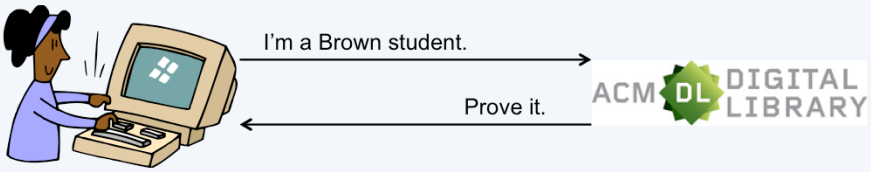


You want to look at a journal article that's in the ACM Digital Library. However, this is not a free service.



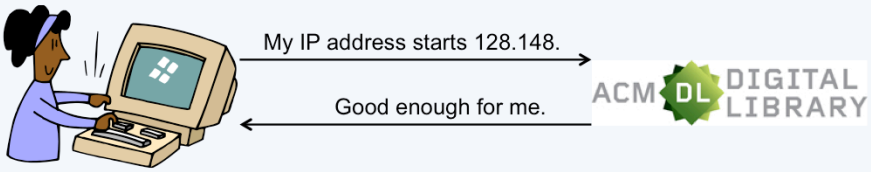
The Brown University Library has paid ACM for the right for Brown students, faculty, and staff to use the ACM Digital Library.

Getting Authorized



You, being a Brown student, assert your right to use the ACM Digital Library.

Getting Authorized



IP addresses starting with 128.148 are assigned only within Brown. Thus, if that's your IP address, you must have some sort of Brown association.

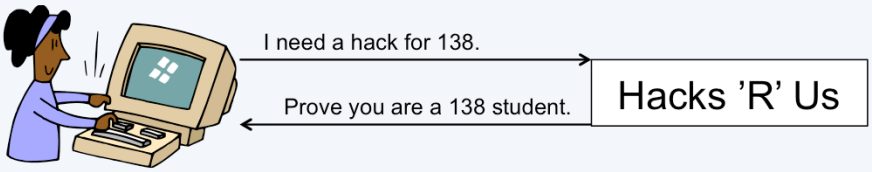
Getting Authorized

$$$$$

Thank you

Hacks 'R' Us

Getting Authorized



Enter Shibboleth

Hacks 'R' Us

CS 138 VIII–25

The history of the term "shibboleth" is interesting (and gruesome) — see http://en.wikipedia.org/wiki/Shibboleth.

Using Shibboleth

- Student
  - logs in to Brown, gets credentials
- Hacks 'r' Us
  - responds to client requests with an authentication request
    - indicates what it requires (e.g., CS138 student status)
- Identity provider
  - contacted by student's browser
  - given student's credentials, returns desired student attributes (CS 138 student)

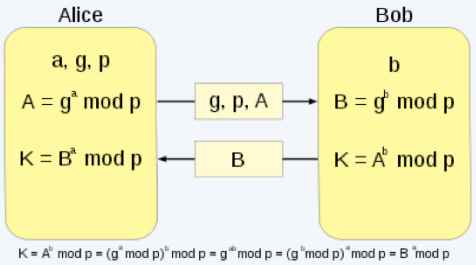
More information on Shibboleth can be found at https://wiki.shibboleth.net/confluence/display/SHIB2/UnderstandingShibboleth.

Shibboleth

- Separates the federation from the authentication
  - Individual IdP's can do what they want
  - Federation makes it more scalable

Diffie-Hellman Key Exchange

- Different model of the world: How to generate keys between two people, securely, no trusted party, even if someone is listening in.



- This is cool. But: Vulnerable to man-in-the-middle attack. Attacker pair-wise negotiates keys with each of A and B and decrypts traffic in the middle. No authentication...

Authorization

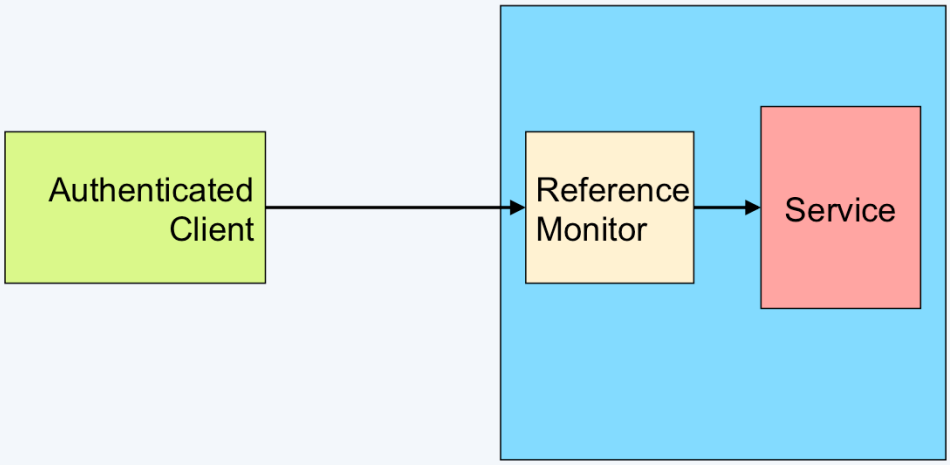
- Is the requestor permitted to perform the requested operation?
- Does this require knowledge of who the requestor is?

An analogy

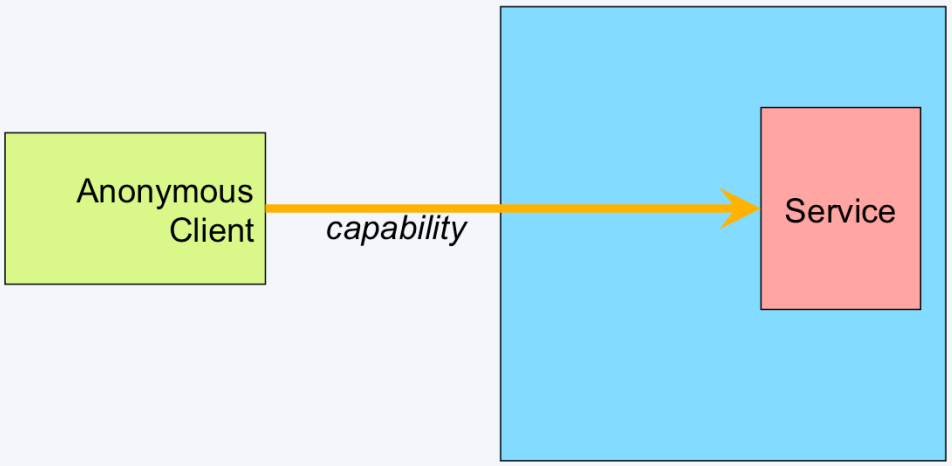
- Alice wants a safe deposit box in a bank
- Two options:
  - Bank maintains a list of who can access the box
  - Bank gives Alice a key (or a combination)
- What are the pros and cons?

This analogy originally from Prof. Wenliang Du, Syracuse University

ACL-Based Authorization



Capability-Based Authentication



Making ACLs Work

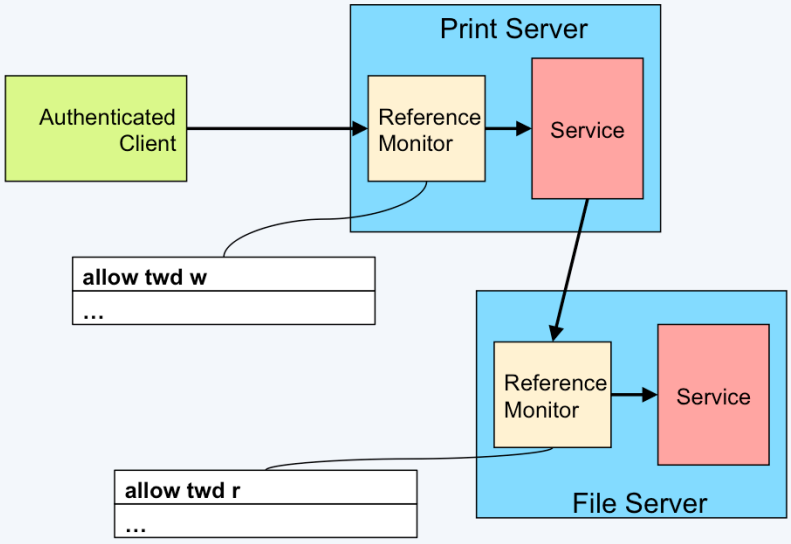
- Client provides credentials
  - privilege attribute certificate (PAC)
    - certificate listing client's credentials
      - e.g., user name, groups, etc.
- Client requests a particular operation
- Server's reference monitor looks up credentials and request in ACL
  - returns permit/deny decision

Privilege Server

- Extend Kerberos into *Privilege Server*
  - maintains user and group database
  - prepares PACs
    - includes them in ticket
    - application-server ticket informs server of all of client's credentials

The discussion of this and the next few slides on impersonation and delegation is based roughly on the DCE security service. PAC stands for *privilege attribute certificate*.

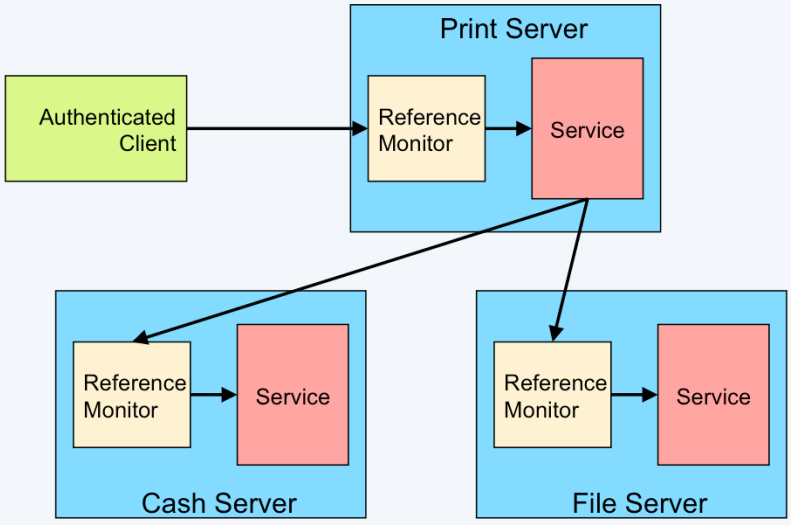
Impersonation



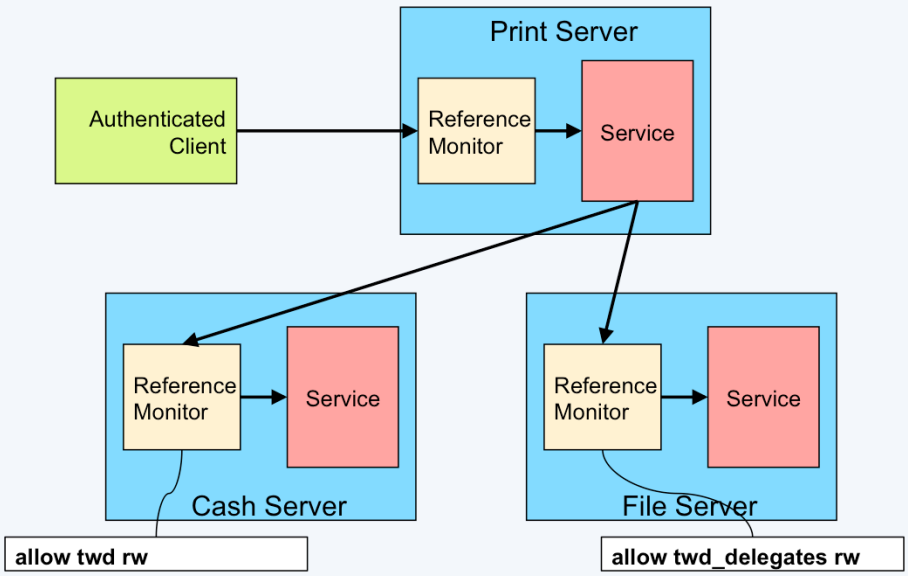
Impersonation using Privilege Server

- Client requests print-server ticket from privilege server
  - asks it to mark PAC "permit impersonation"
- Client sends ticket to print server
- Print server requests file-server ticket from privilege server
  - includes client's print-server ticket
  - privilege server provides file-server ticket containing original client's PAC
    - print server *impersonates* client

Impersonation Problems



Delegation

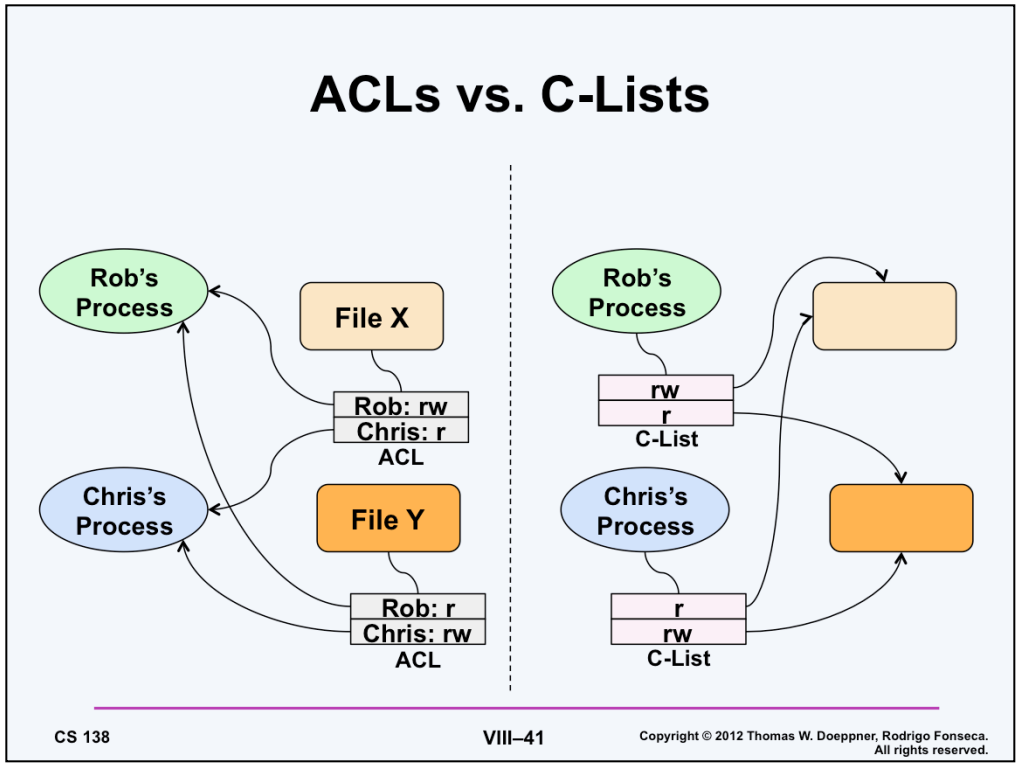


How It's Done

- Client requests print-server ticket with delegation permitted
  - privilege server constructs ticket with client's PAC so marked
- Client presents ticket to print server
- Print server requests delegated file-server ticket from privilege server
  - privilege server returns ticket with both original client's and print-server's PACs
- Print server presents ticket to file server
  - file server checks delegate entries in ACL

Capabilities

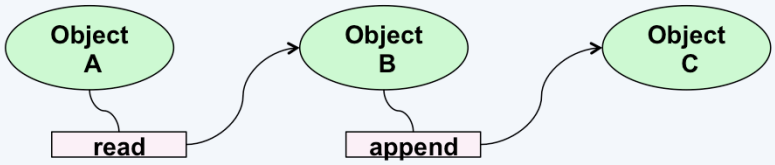
- A capability is both a reference and an access right to a particular resource



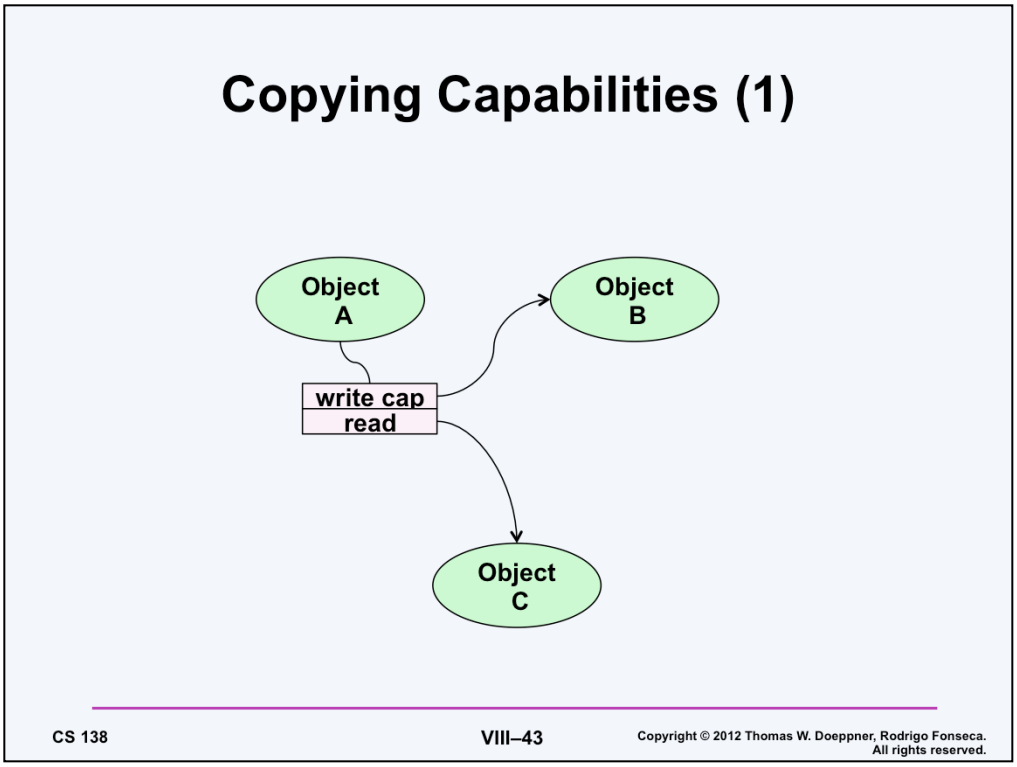
A *capability* is both a reference to a resource and an access right to that resource. Furthermore, it's unforgeable. The set of capabilities possessed by a subject is called its *C-list*. Note that with capabilities, it's not necessary for resources to have names — the capability suffices. Note that the ACLs refer to any process whose user ID is Robor Chris, whereas a C-list merely indicates that a particular process has a capability for the indicated resource.

More General View

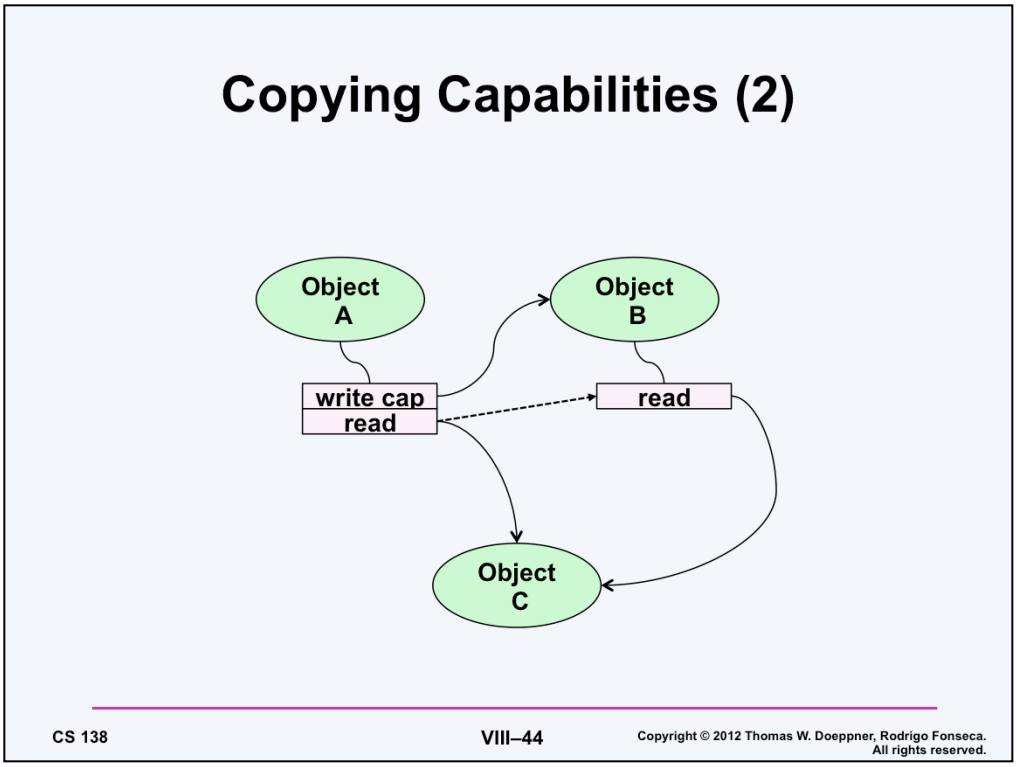
- **Subjects and resources are *objects* (in the OO sense)**



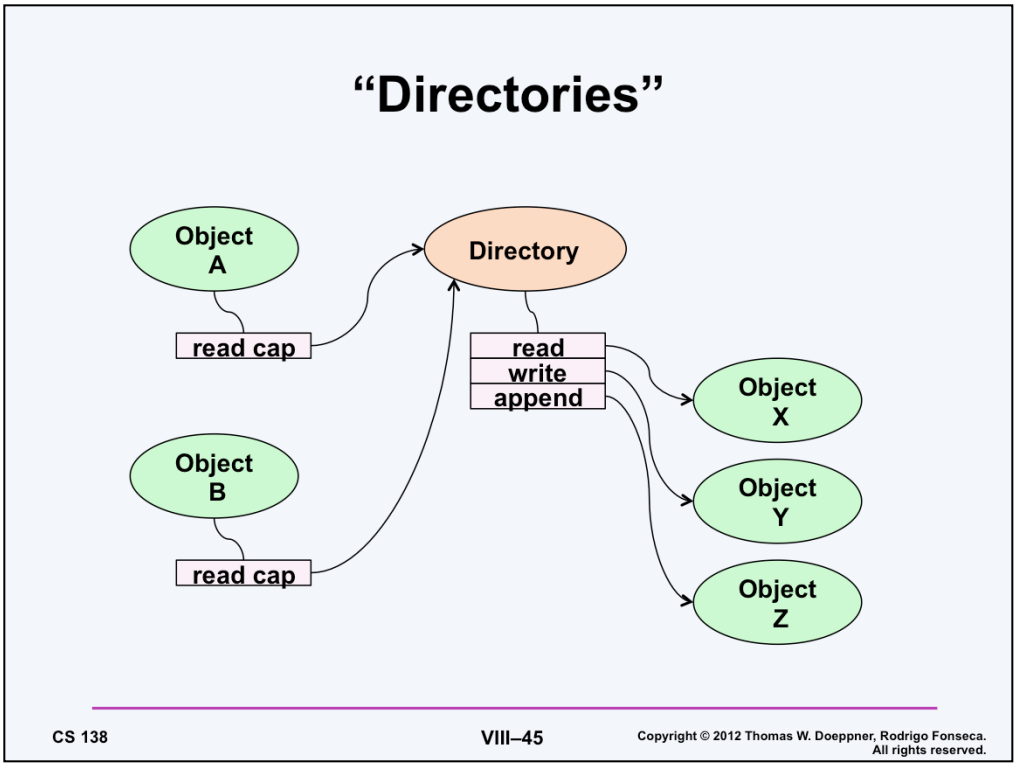
Let's think of both subjects and resources as being general objects (in the object-oriented-programming sense). A capability is a reference to an object that allows the bearer to invoke the indicated operation.



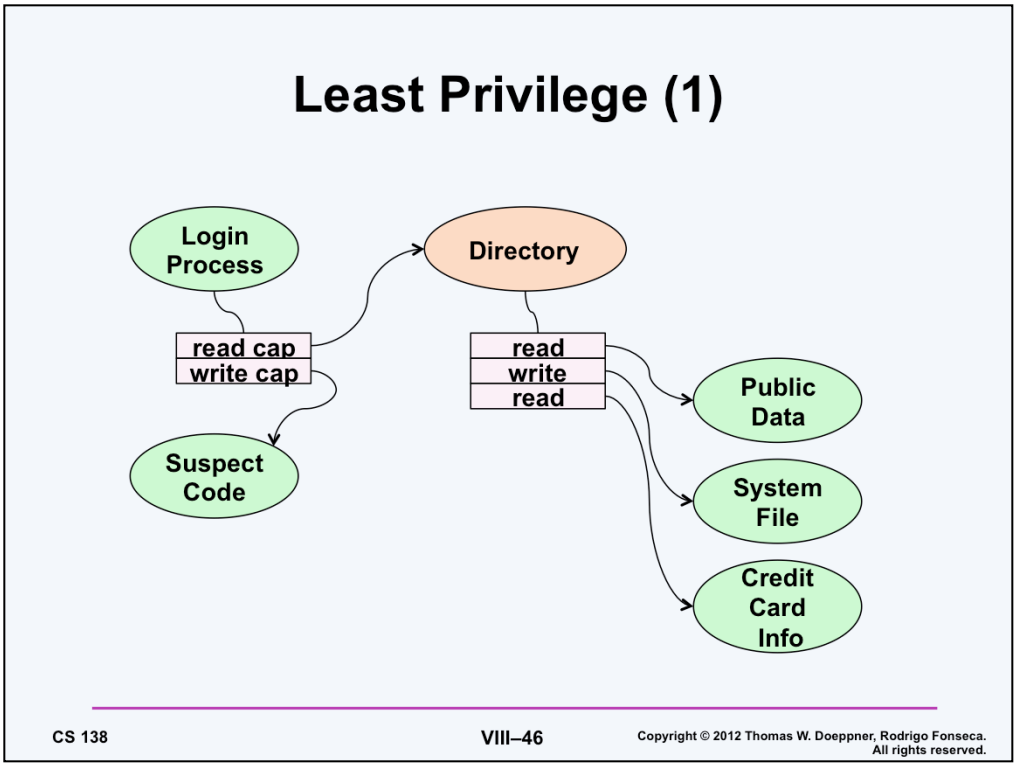
Object A has a “write-capability” capability for object B, which allows A to copy capabilities to B.



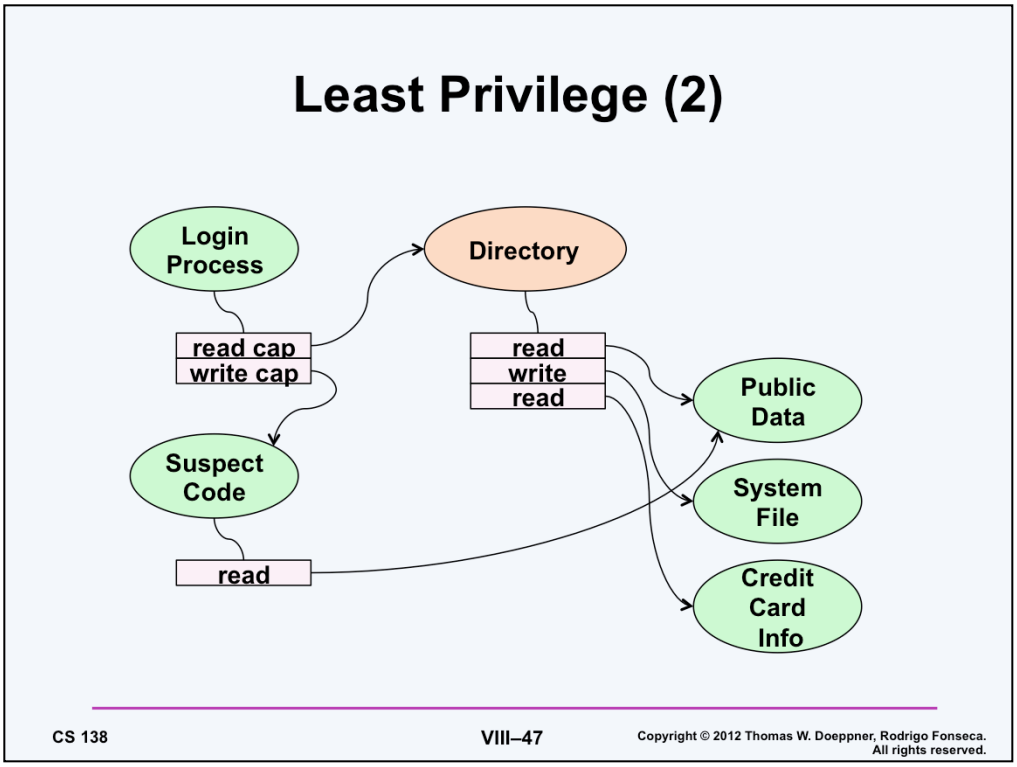
Object A has copied its “read capability for object C” to object B. Note that this provides a form of delegation.



Here we have a "directory object" that provides capabilities to other objects. Object A may use its read capability to fetch capabilities from the directory object.



Here we want to run a program that we've recently downloaded from the web. We create a process in which to run it, giving our login process a write-capability capability to it.



We give the new process a read capability for some public data, but no capability for anything else (and particularly no capability for getting other capabilities from the directory).

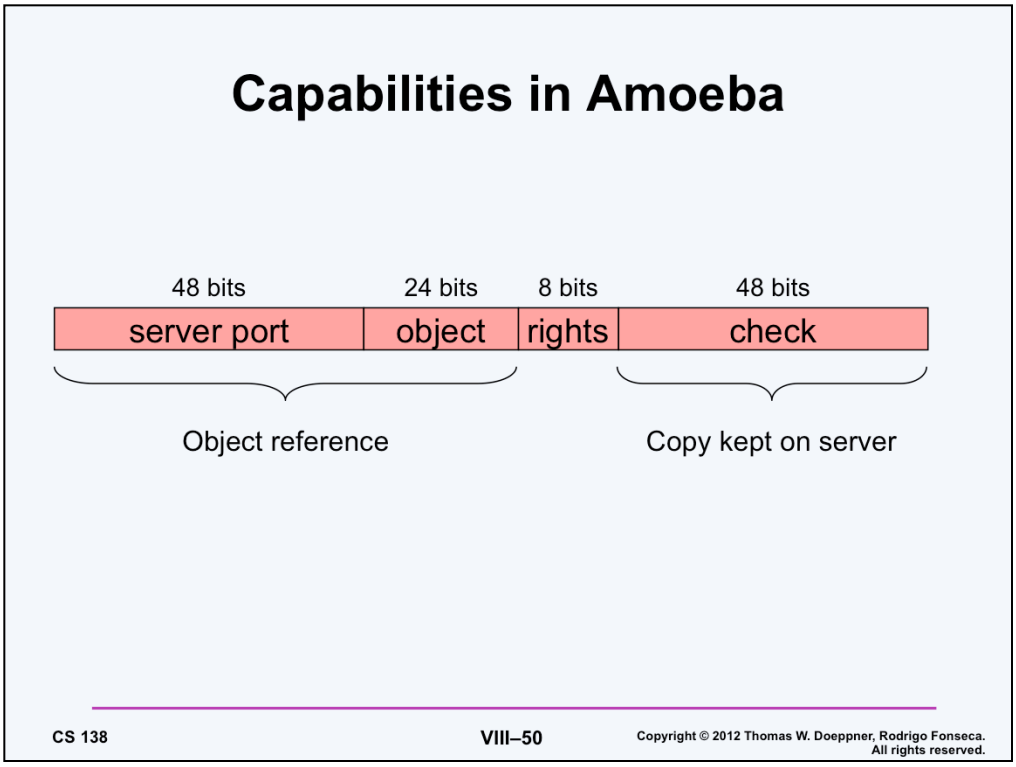
An analogy

	ACL (List)	Capability (Key)
Authentication	Bank must check list	Bank not involved
Forging access	Bank must secure list	Can't be forged
Adding a new person	Owner visits bank	Copy key
Delegation	Friend can't delegate	Friend can give key
Revocation	Owner can remove ex	Harder

- **Sharing online album**
  - **Authorize specific users**
  - **Share by secret URL**

ACLs vs. Capabilities

- ACLs
  - Authentication
    - Reference monitor involved
  - specifying access rights
    - easy
  - least privilege
    - hard
  - delegation
    - Awkward
  - Revocation
    - easy
- Capabilities
  - Authentication
    - No one involved
  - specifying access rights
    - awkward
  - least privilege
    - easy
  - delegation
    - Easy
  - Revocation
    - hard

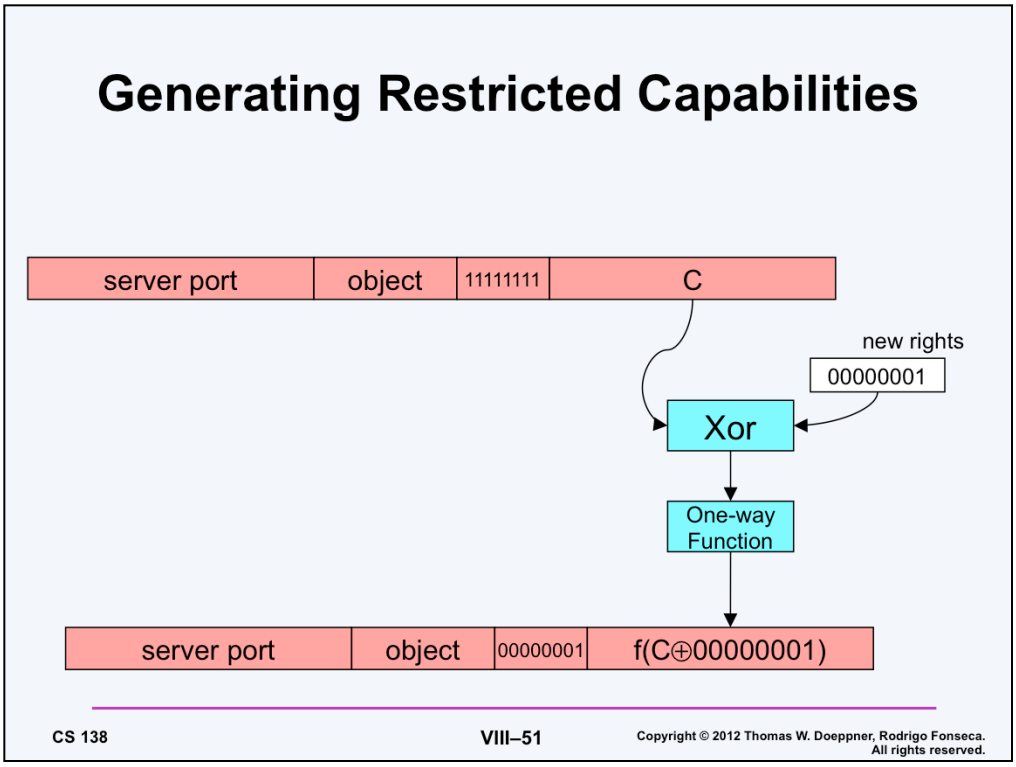


Amoeba is a distributed operating system developed in the 1980s by Andrew Tanenbaum and his students. Among its innovations was the use of distributed capabilities. The description given here is from the textbook by Tanenbaum and Van Steen, starting on page 435.

When an object is created, the server returns to the creator a capability, as in the slide, that includes "owner" rights, i.e., all possible rights to the object. This is indicated by all ones in the rights field. The server-port and object fields together refer to the object: the former is the network address of the server and the latter is the server's ID for the object. The check field is a 48-bit random value chosen by the server. The server maintains a copy of it along with the object, and uses it to verify the capability.

A client presents the entire capability to the server both to reference the object and to identify its rights to the object.

The approach has obvious limitations if eavesdropping of the network is possible. Can you think of an approach for making it more secure?



An object owner can create a new capabilities with restricted access rights as shown in the slide. It places the new rights in the rights field, then constructs a new check by xor'ing the new rights (extended to 48 bits) with the check bits, then applying a one-way function. The result can then be presented to the server, which can verify that the rights were derived form the check bits appropriately.

Note that, by itself, such a capability is subject to sniffing attacks and can be replayed by the attacker. To avoid this, some additional form of protection is necessary, such as using an encrypted channel, perhaps set up by TLS.