Intrusion-Tolerant Authorization Service

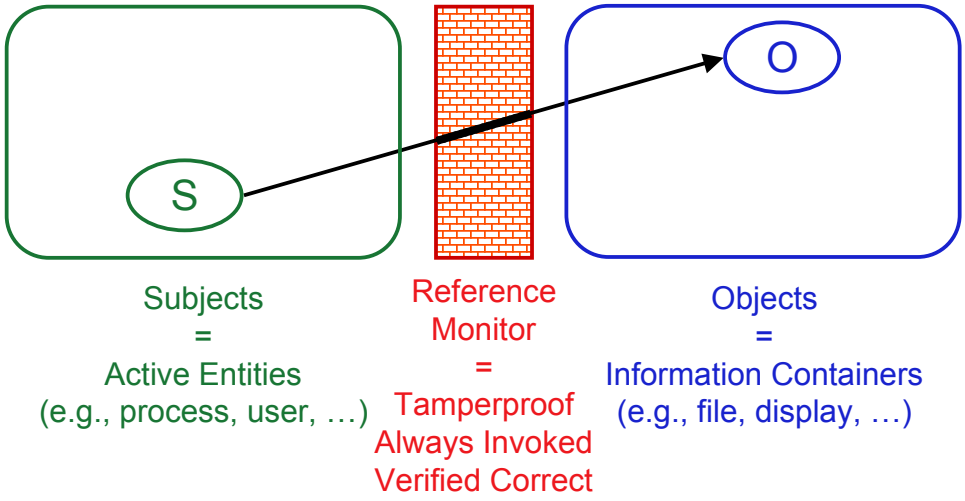
Presented by *Yves Deswarte*
Demo by *Christophe Zanon*

Contributors: *Norredine Abghour, Vincent Nicomette, David Powell*

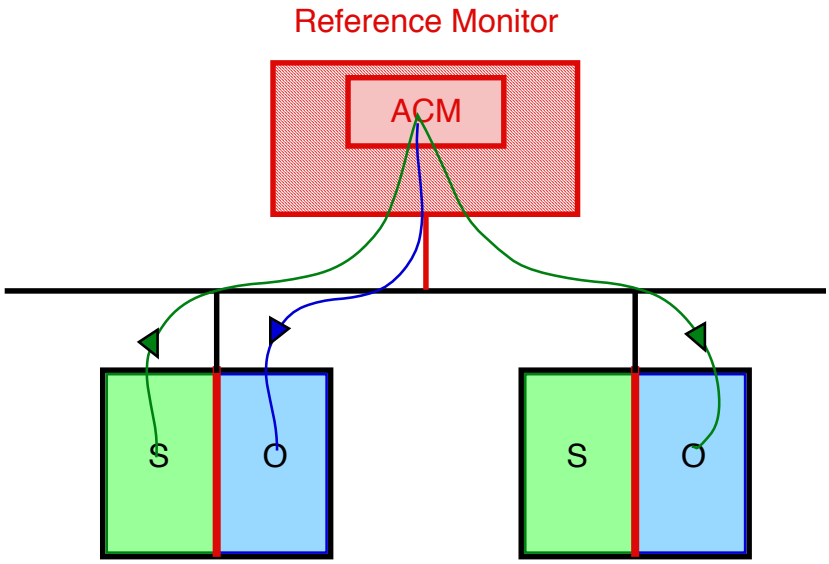
Authorization

- Contributes to protection:
  - Error detection/confinement
  - Intrusion prevention/confinement
- For Internet applications:
  - More flexible than "client-server" paradigm
  - Contributes to privacy:
    personal information is disclosed only on a "need-to-know" basis

Authorization: reference monitor

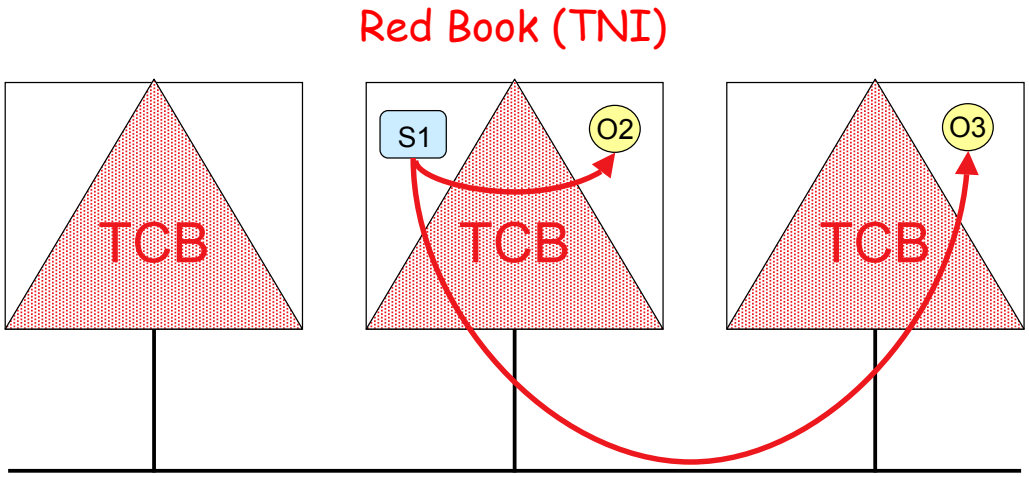


Distributed Authorization ? (1)



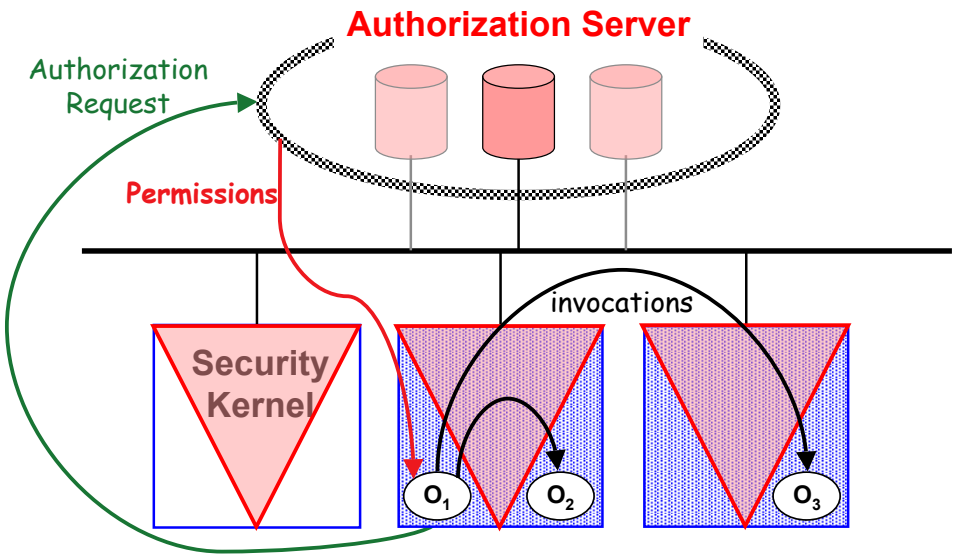
☺ small trusted area, easy administration
☹ bottleneck, single-point-of-failure

Distributed Authorisation ? (2)

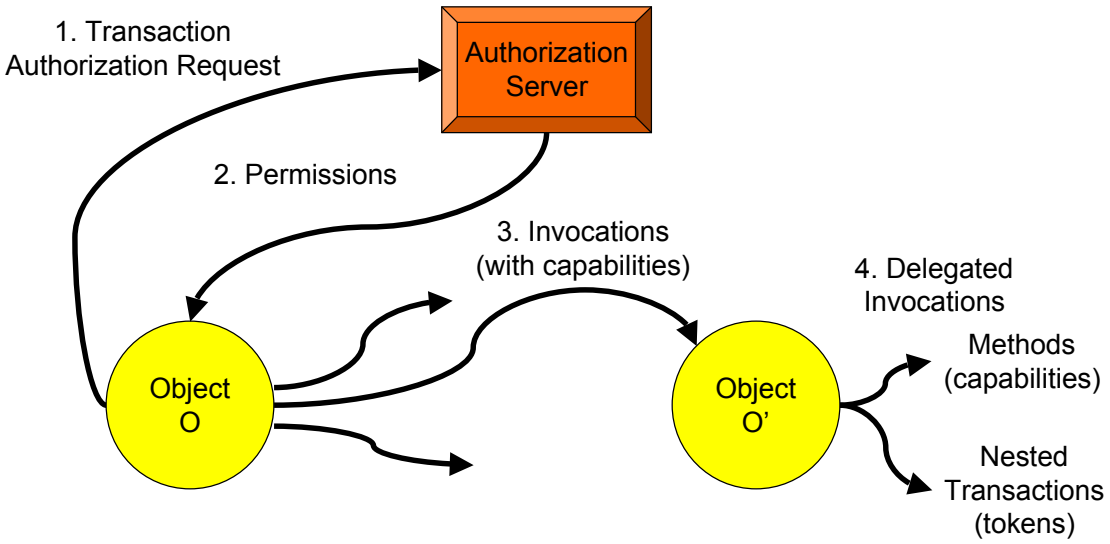


☺ No bottleneck, no single-point-of-accidental-failure
☹ Mutual trust between TCBs = multiple SPoFs
consistency?

Authorization Scheme



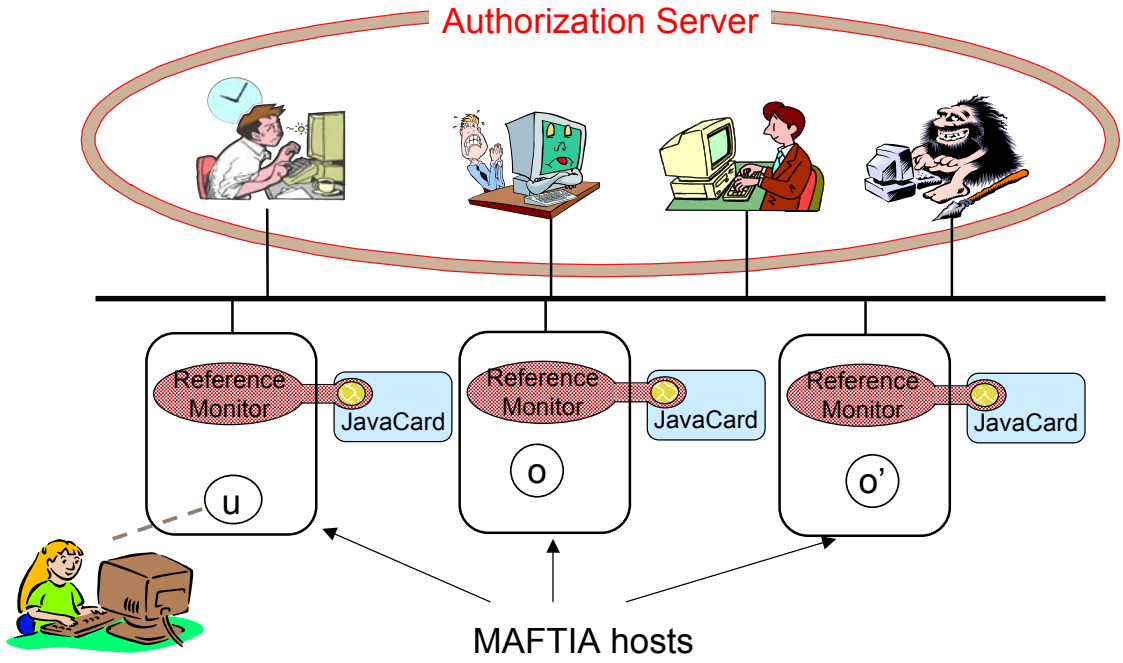
Permissions



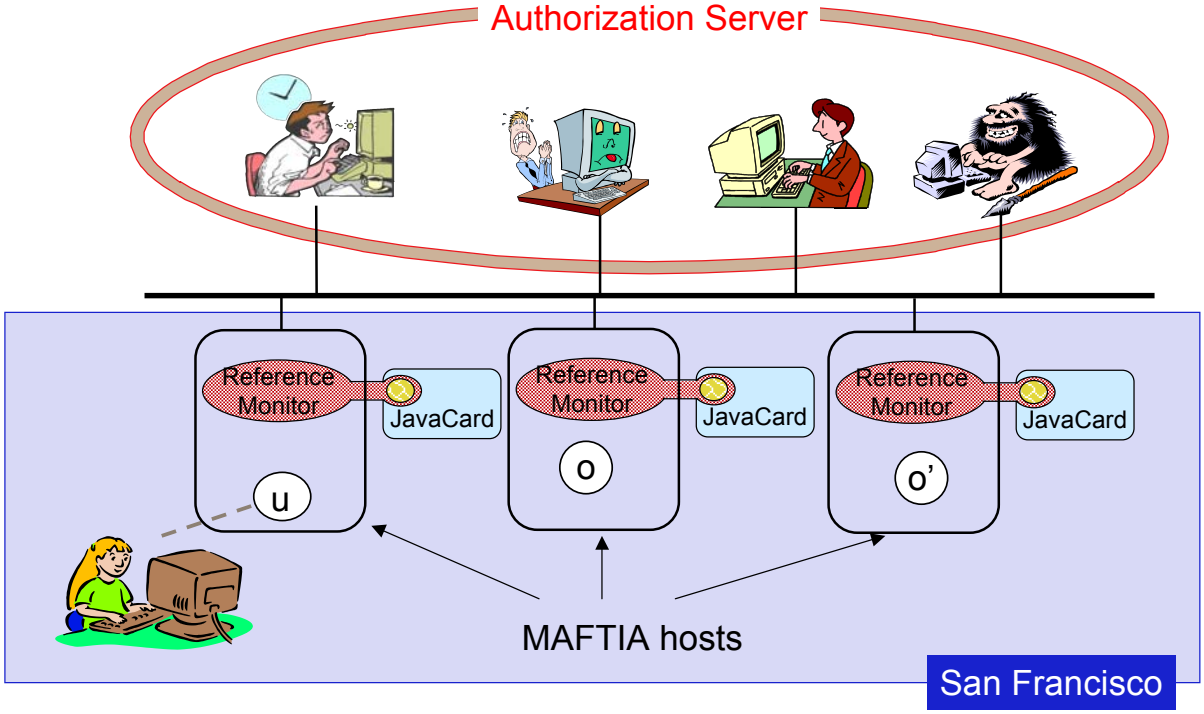
Security properties

- Authorization server:
  - AS1: The AS generates only valid permissions (capabilities, tokens)
  - AS2: It is not possible to prevent AS from generating valid permissions
  - ✓ Byzantine agreement, threshold signatures (WP2)
- Local Reference Monitors
  - RM1: Only permitted operations can be executed on a non-faulty host
  - RM2: It is not possible to prevent the execution of a permitted operation on a non-faulty host
  - ✓ Verification of capabilities by RM, signed acknowledgements & time-outs

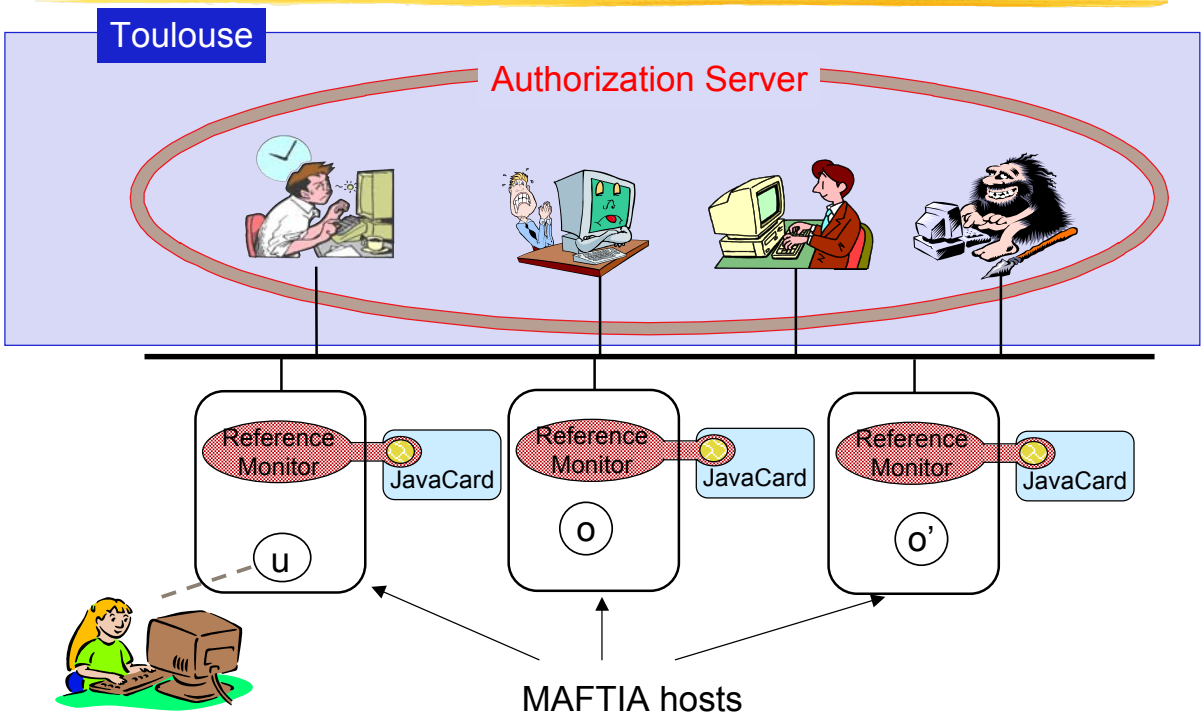
Demonstration



Demonstration



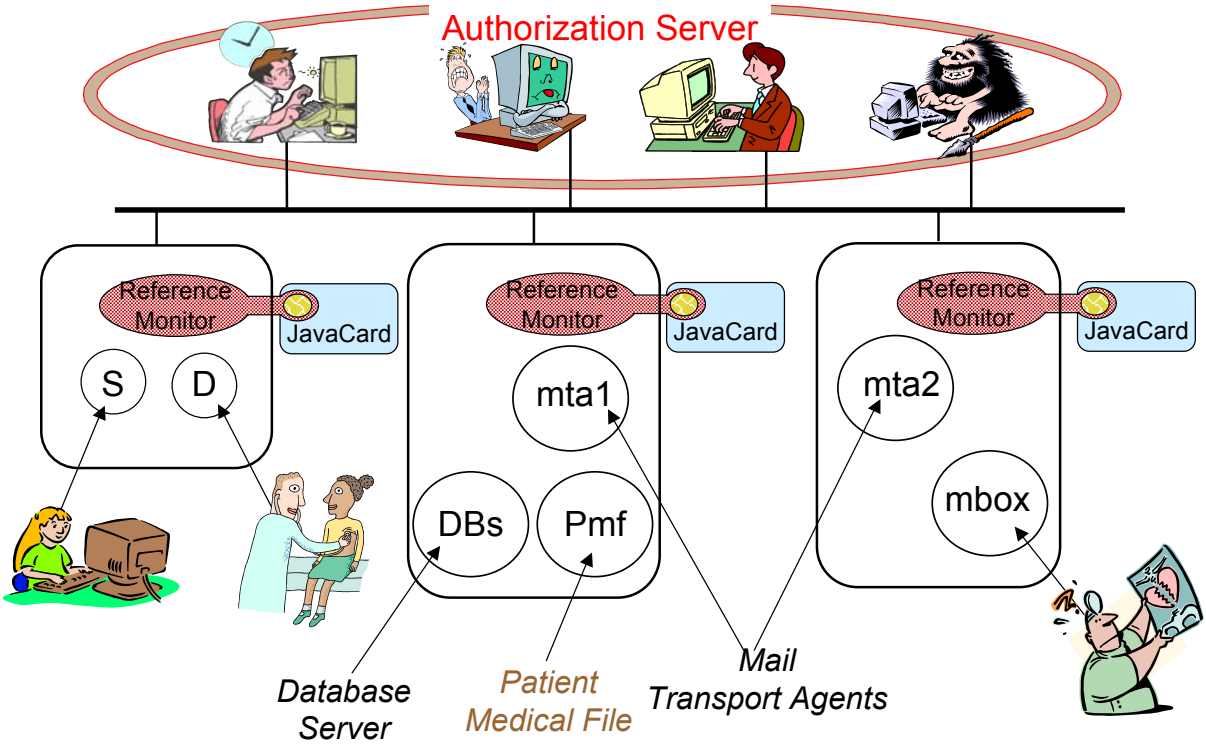
Demonstration



Scenario

- Hosts
  - Medical practice: a Doctor, a secretary
  - Patient Record Server
  - A Specialist (mailbox)
- Transactions
  - The Dr can read/write his patients' records (the secretary cannot)
  - The Dr and the secretary can send a patient record copy to a specialist by mail
- Objects
  - Dr, Secretary
  - DBS
  - PMF: Patient Medical Files
  - MTA: Mail Transfer Agents (1 per host)
  - Specialist's mailbox

Object distribution



Implementation

- All the code is written in Java (application objects, reference monitor, authorization server)
- Objects interact through Java Remote Method Invocations (RMI)
- RMIs are *automatically and transparently intercepted* by the "dispatcher", capabilities are verified through the JavaCard
- The operating systems are MacOSX and Solaris
- Java cards are GemXpresso cards from Gemplus

Demo steps

1. Users and reference monitors authenticate themselves to the authorization server
2. A transaction is carried out without corruption (secretary, send-patient-file-by-mail, specialist)
3. A transaction is attempted with a forged capability (invalid signature)
4. A transaction is carried out with corruption/crash of one of the security sites
5. Examples of unauthorized transactions