GDB Meeting, 08.11.2006

“ALICE Authorization for Data Access”

Andreas-Joachim Peters CERN

Paper: http://people.web.psi.ch/feichtinger/doc/authz.pdf

Enabling Grids for E-science in Europe

www.eu-egee.org   cern.ch/lcg   http://cern.ch/arda

EGEE is a project funded by the European Union under contract IST-2003-508833

Overview

- Concepts
- Authorization Model & Implementation
- Access Tokens
- xrootd implementation
- Comparison & Summary

Basic Concepts

- The ALICE authorization model is based on 3 concepts:

User Virtualization

1

“authorize according to VO structure & policies”
- user community is dynamic
- policies are dynamic
- currently no virtualization support in deployed kernels
- policies kept by central instance (FC)
- avoid authorization by physical user mapping on site

Common Security Infrastructure

2

“use standard authentication and security ”
- GSI / Globus
- OpenSSL en-/decryption

Single Entry Point for User Data Access

3

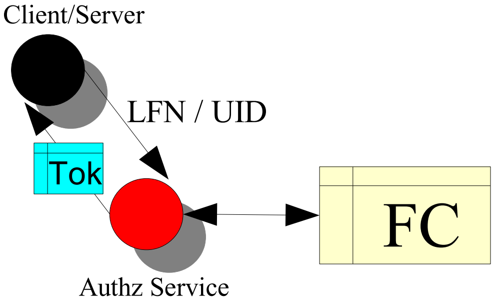
“all user data access uses xrootd”
- allows new authorization scheme
- all files can be owned by the i/o service

Authorization Principle

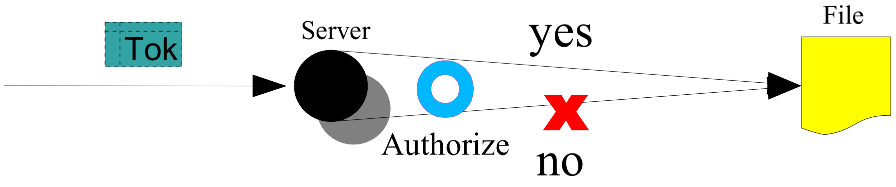
- File Access policies are kept in the central FC

	/alice/raw1.root -rw-r—r-- aliprod z2 /alice/raw2.root -rw-r—r-- aliprod z2 /alice/priv.root -rw------- aliprod z2

- user/server acquires a file access token (envelope) for each file access from a central authorization service

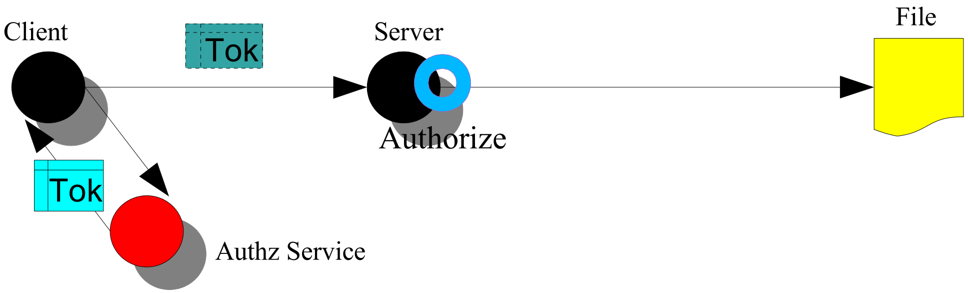


- the i/o server allows file access according to the presented token



Authorization Models

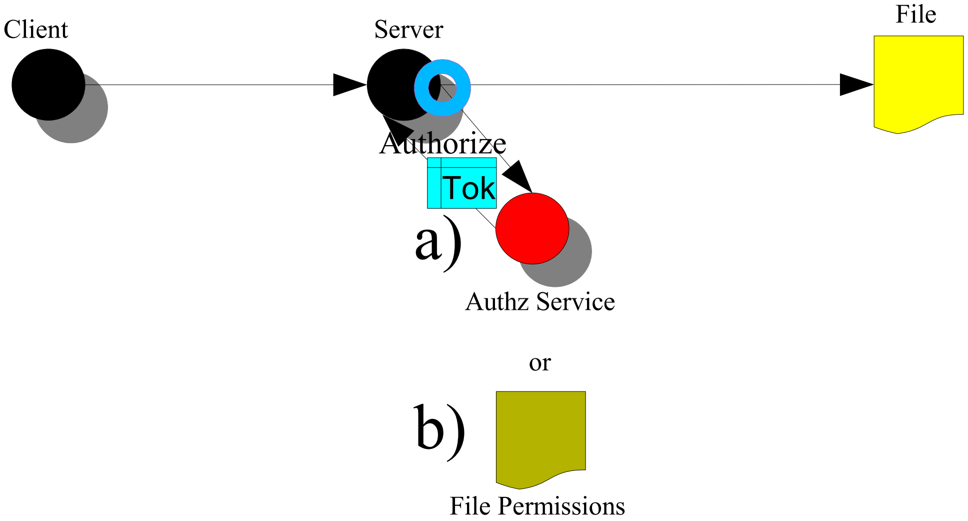
- Clientside Authorization Callback



Pro
Server decoupled from Authz Service
No call back delay on Server

Contra
More effort to move authz decision in a secure way via the client to the server

- Serverside Authorization Callback

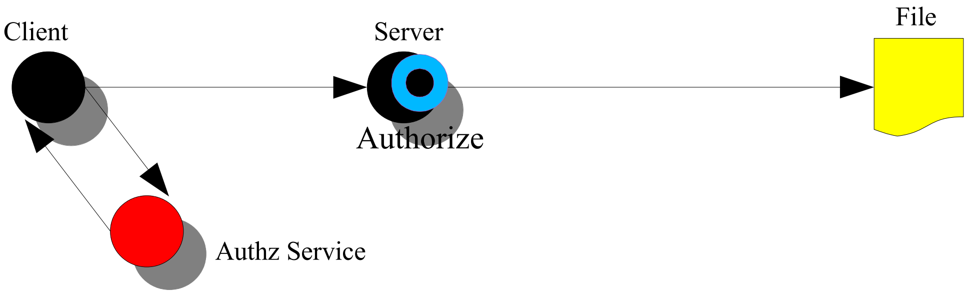


Pro
'Easier' security handling
(no client connection to authz service)

Contra
Server performance depends on Authz time/stability
Server needs to authenticate to Authz
Server needs to link against Authz client libraries

ALICE Authorization Model for Data Access

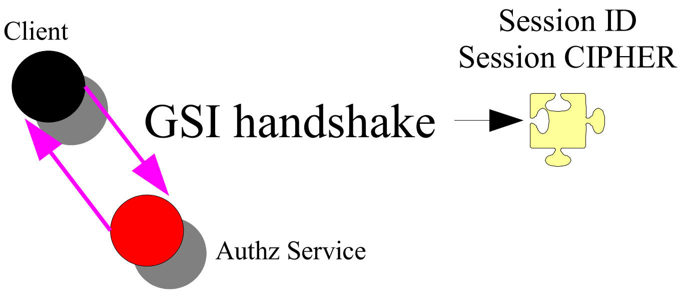
- ALICE: Clientside Authorization Callback



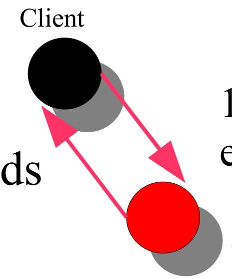
Security Requirements

Client authentication to Authz Service via GSI

once



afterwards



128-bit CIPHER encrypted exchange

ALICE Authorization Model for Data Access

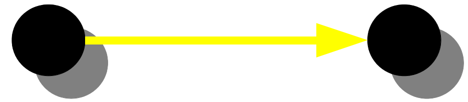
- Client communication with Data Server
  (exchange of authz tokens)
  - option 1 - strong
    - GSI authentication & token verification

Client
Server

  - options 2 - medium
    - SSL handshake + encrypted traffic

Client
Server

  - option 3 - loose
    - unencrypted traffic – no client verification – token can be catched on the network



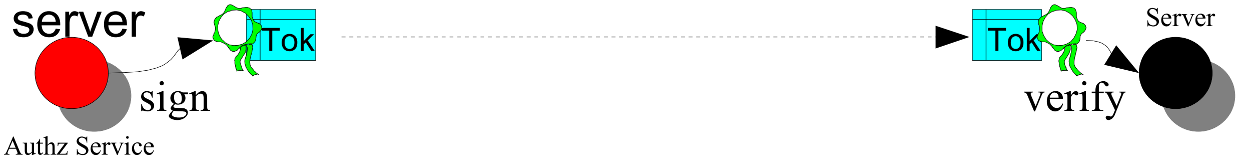
Access Tokens

- Access Tokens contain file meta data in XML format
  - GUID
  - LFN
  - TURL
  - authorized access command
    - write
    - write-once
    - read
    - delete
  - validity
  - CERT subject of the client who can use this token

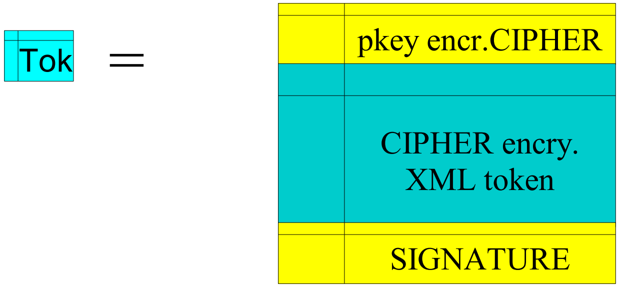
```
<authz>
  <file>
     <lfn> ....
  </file>
</authz>
```

Access Tokens

- Access Tokens are protected using public key infrastructure
  - Step I
    - tokens are signed by the authorization service and verified by the i/o server



  - Step II
    - tokens are encrypted with a random 128 bit CIPHER to hide the information to the client
    - the CIPHER is appended to the token encoded with the public key of the i/o server and decoded by the i/o server with it's private key



Tok =

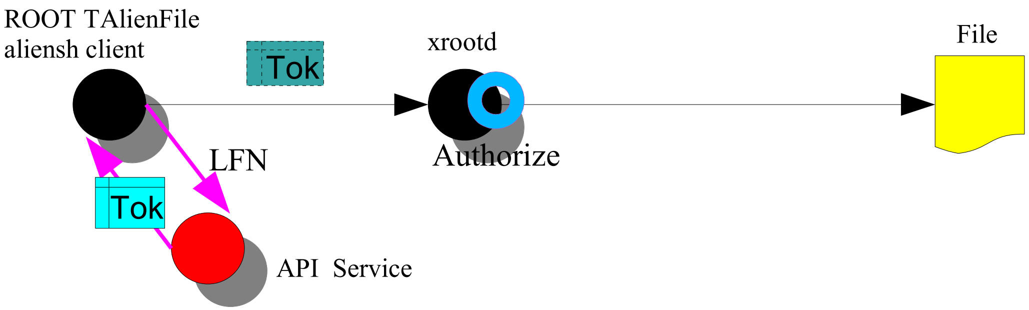
pkey encr.CIPHER

CIPHER encry. XML token

SIGNATURE

Remark: Step II could be skipped for performance gain

The current ALICE Authz operation model



1. client asks for read/write-once/delete access for <LFN> via GSI authenticated session connection
2. API service authorizes request and issues an access token valid 1 day
3. the access token is appended as opaque information to the file URL
   root://lxfsra0606.cern.ch//<LFN>?authz=----BEGIN SEALED CIPHER----......
4. client connects to xrootd with the modified URL
   (currently we don' require GSI authentication to xrootd)
5. xrootd decodes the token and replaces the LFN with the TURL inside the token

Performance: single client 10 file open/s per thread
200 token en-decodings per sec on P4 2.8GHz

[during ALICE PDC 10 file open/s avg. over 1 week]

The xrootd plugin for Authorization

- The authorization mechanism is implemented as a plugin library in xrootd
  - change in xrootd configuration file
    xrootd.fslib /...../libXrdOfs.so
    to
    xrootd.fslib /...../libXrdTokenAuthzOfs.so
  - install the public authz service key for token verification
  - install private i/o service key for token decryption
    (the public partner is given to the authz service)
  - support for read/delete/write-once(no file modification)
  - an additional authorization configuration file specifies
    - the location of the public/private key for a certain VO
    - which physical paths have to be authorized with a token
    - if matching between GSI subject and token subject is required (if GSI authentication has to be used)

xrootd/authorization with various Storage Systems

dCache Castor2 DPM

- dCache has an emulation of the xrootd protocol
- the decoding part of the authorization library and the functionality of the xrootd plugin has been implemented in JAVA and a first successful test against the ALICE authz service (API service) has been done

- a Castor2 and DPM integration of xrootd is existing
  - the same authorization library can be used here, since the native xrootd server is used

Remark: All three have to be tested now in detail by ALICE!

Advantages & Drawbacks & Extensions

- Since every file access is authorized centrally, it is easy to enforce other policies than permissions on file access
  [quota, access volume per day etc ... , priorities]

- Authz/FC service is scaling (100% parallel) within the needs of ALICE
  - currently 3 machines at CERN

- a central service requires access from every workernode
  - communication uses encrypted SOAP messages, which can be routed via standard site HTTP proxies if outbound connectivity disappears at a certain point

- possibility to introduce 'group' tokens, which allow access to a complete data set

Comparison to the VOMS (proxy mapping) model

- Token Authorization
  - based on GSI security
  - permissions on file level
    - every user is private
  - no limit in number of users and groups
  - no synchronization between local accounts and central configuration
  - access tokens are like 'proxy' certificates for individual files [ms scale]
  - Token Authorization can provide everything provided by VOMS/Proxy Mapping

- VOMS Authorization
  - based on GSI security
  - permissions on group level
    - groups are private
  - number of individual users/roles/groups OS-limited
  - synchronization of local accounts with central configuration
  - VOMS proxies authorize big file groups [s scale]

Summary

- ALICE authorization model uses common security standards to provide privacy to experiment and user data
  - GSI authentication
  - public key infrastructure
  - symm. CIPHER encoding for high-performance
- ALICE has integrated their authorization scheme into xrootd
  - get all xrootd advantes (connection multiplexing, file open time O(ms), serve thousand of clients/files in parallel ....)
  - easy configuration: 2 conf. files + 2 keys
    - easy for multi-diskserver setups
  - level of security is configurable
    - GSI authentication in xrootd makes sense, when user jobs are executed with glexec and user proxys can be kept private in batch jobs
- ALICE model provides fine grained and secure acc. control
  - excellent logging and tracking facilities to discover incidents
  - access can be blocked centrally or on site

Thank you!

Any Questions?