Lustre Security

Engineering Requirements

Peter J. Braam

April 2005

Outline

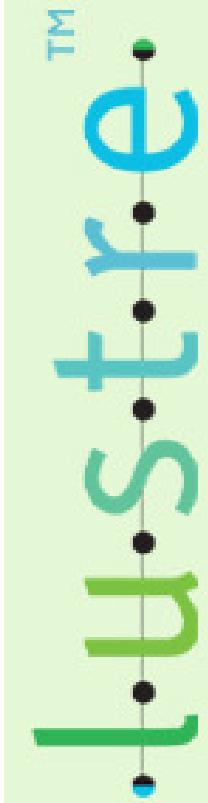
- Problem Statement
- Architectural Strategy
- System Context
- Architectural Views
- How the architecture works

This presentation is crafted to be compliant with the standards explained in:

Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; & Stafford, J. Documenting Software Architectures: Views and Beyond. Boston, MA: Addison-Wesley, 2002. **(SEI series in software engineering)**

Cluster File Systems, Inc. is an SEI TSP(sm) transition partner.

Problem Statement



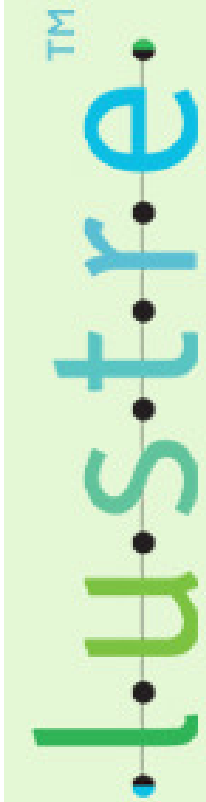
Lustre security requirements

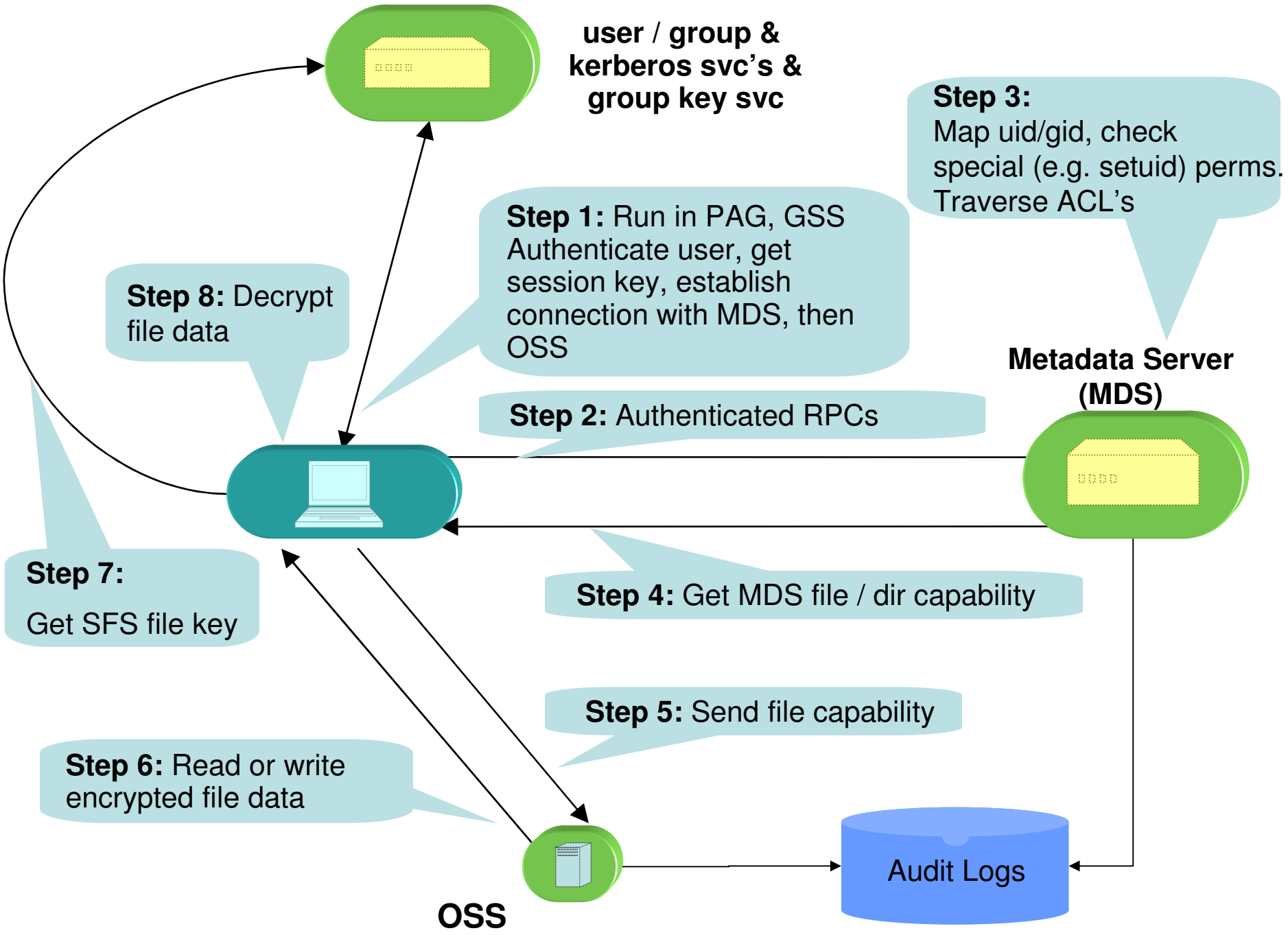
- Input from
  - Trilabs SGPFS contract
  - Marquee customers
  - Experience with other systems

Features requested

- network policy – don't penalize "secure" nets
- GSSAPI authentication – with Kerberos5 initially
- Interoperate with existing user/group database
  - Handle remote user and group data bases
  - Optionally retain strict POSIX and administrative features
- Instant revocation
- PAG's
- ACL
  - POSIX
  - with remote handling
- Capabilities to protect objects and access by FID
- Auditing
- Encrypting file data
- Location dependent authorization
- Secure handling of write-back caches

Architectural Strategy




user / group &
kerberos svc's &
group key svc
Step 3:
Map uid/gid, check
special (e.g. setuid) perms.
Traverse ACL's
Step 1: Run in PAG, GSS
Authenticate user, get
session key, establish
connection with MDS, then
OSS
Step 8: Decrypt
file data
Metadata Server
(MDS)
Step 2: Authenticated RPCs
Step 7:
Get SFS file key
Step 4: Get MDS file / dir capability
Step 5: Send file capability
Step 6: Read or write
encrypted file data
Audit Logs
OSS

System Context

Existing site managed infrastructure

- Site should have a user / group database
- Consistent among all servers
  - preferably consistent among all clients
- Kerberos database
- Lustre uses normal user level system API's for access

Lustre introduced infrastructure

- Client
  - Key to enable root to mount
- Enlarged UID database for use by MDS
  - Grant certain principals special permissions
  - Optional, but most sites will want a few entries
    - E.g. a user allowed to analyze audit logs

Architectural Views

Network policies

- Receiving packets:
  - write data in a buffer
  - deliver an event
- This is done by the NAL & Portals
- NAL can set flag in events for incoming requests for
  - traffic incoming on a certain interface AND/OR
  - from a trusted subnet or list of addresses
- This enables options. For example, servers:
  - let local cluster networks bypass security
  - subject traffic from subnets or routers to less scrutiny

GSS-API based authentication

- Enquiry handshake
  - Phase one: get servers capabilities (un-encrypted)
  - Setup GSS context if server allows
  - Phase two: get targets requirements / capabilities (encrypted)
  - Now proceed with normal request processing
- Setup GSS context
  - Make standard upcall to modified NFS v4 client GSS daemon
  - Modifications ask Lustre for context GSS context, using ptlrpc
    - Normally SUN RPCSEC
  - Almost unmodified (from NFS) GSS handling on the server side
    - Security context only established if Kerberos principal has an account

GSS buffer handling

- Request buffers are prepared by ptlrpc
- Intercept:
  - Just before sending
  - Just after receiving
- Apply integrity & encryption code to buffers
  - Include security context data
- Based on code from NFS v4
  - Many fairly trivial modifications
- The total GSS patch is large (~8% of Lustre)

User and Group Handling

- GSS sends a name to the server
- Lookup in normal user / group databases on MDS get:
  - uid
  - gid
  - group membership

  for the principal
- Recently found entries are cached in kernel
  - context cache can be flushed
- Server threads
  - Use this context
  - Chroot
  - Occasionally use root permissions (e.g. to write a log file)

Remote access

- Client user / group database != MDS database
- For authenticated principals MDS maps:
  - uid and
  - primary group
  - does not map secondary groups
- All other uid's / gid's are mapped to
  - unknown Lustre user id
  - unknown Lustre group

POSIX features & special permissions

- Lustre servers need privileged access
- A backup program may need raw FID access
- An audit log analyzer may need raw FID access
- Most servers need Lustre to respect setuid / setgid
- The samba server needs Lustre to respect setgrps
  - Reducing group membership can always be respected
  - Root can set groups arbitrarily – allow only trusted software
- Most clients should face root_squash

- To handle this:
  - principal can obtain special treatment
  - MDS queries a secondary user database upon connect
  - clients always send all groups and fsuid / fsgid to MDS

Revocation

- Remove user from user / group database
- Force MDS servers to drop security contexts
- Re-authentication will fail
  - client may even get evicted, but will re-connect
- Capabilities can also be revoked – see below

PAG's

- Process authentication groups
  - a group of processes enjoying one GSS credential
  - typically smaller than all processes running as a uid
- principally there to defend against
  - root does **su "user"**
    - and obtains permission for user
  - multiple "logins" to a Lustre client
    - one login should not borrow GSS credentials from another
- But:
  - If root can write kernel memory
  - If root can tinker with ticket files
  - this all becomes almost pointless – need better infrastructure
- If better NFS v4 solution arrives in-time will use it
  - Otherwise CFS will provide a basic AFS style patch

ACL's

- Lustre implements POSIX ACLs
- If client has same uid / gid database
  - use existing ACL API's in Linux
  - use extended attribute manipulation
  - cache ACLs
  - high performance
- For remote clients (different uid & gid's)
  - ask MDS to perform ACL updates and queries
  - cache (& revoke) recently granted permissions
  - user can issue standard ACL commands
    - hope to get slightly modified ACL utilities adopted

Capabilities

- MDS gives out
  - object id's for file data objects
    - used for read / write, truncate operations by clients
  - FID's for MDS inodes
    - used for lookup, getattr, setattr and other operations by clients
- A well behaved client will
  - only do I/O with objects after a successful open
  - handle lock revocations for ACL's, mode bits, owners
- but .. this needs to be enforced reasonably
- Use capabilities

Capabilities

- tamper-proof – only for specific ops on specific objects
- time limited
- transmitted encrypted
- can contain owning kerberos principal
  - service using capability needs GSS to check principal
  - kind of need client-OST GSS for data integrity checks
- can contain owning client identity (not planned now)
  - service using capability needs gss to check client
  - such capabilities cannot be snooped
  - prevents job wide capabilities
- Revocable – distribute revocation list to OSS's
  - by fid, by principal, by client
  - or invalidate all capabilities and manage revocation from the MDS

Renewing capabilities

- I/O can last a long time
  - Once a file is open, client can renew a capability
- Files may need to be re-opened on recovery
  - Requires a capability
  - Permissions on the file may have changed
    - re-opening must remain possible
- Some directories need similar treatment
  - clients chdir into the directory
  - mount points (?)
- Client renews capabilities well before expiry

File data integrity

- Can be done with GSS
- Can be done out of band by prepare / commit RPC's

Auditing

- MDS's, OSS's, clients can audit
  - failed operations
  - some subset of operations
- Audit logs contain enough info to know
  - which user
  - (tried to) access what pathname
  - for what operation
- Audit logs
  - are stored in Lustre, are transactionally correct
  - Log is post processed into syslog
  - Old data in log can be discarded after handoff
  - Presented as files
- Analysis
  - involves combinations of logs
  - requires objects to know owning inodes, inodes to know parent dirs

Encryption of file data

- Use protocol similar to STK SFS
- Client can only receive/send encrypted file data
  - Encrypted file key and special ACL entries present in EA
  - A group key service can provide the un-encrypted key
- Lustre will provide hooks for a group key service API
- Lustre will implement a simple group key service
  - inside the MDS
  - using POSIX ACL's
- This applies to:
  - whole FS
  - flagged directories or flagged files
  - files in sub-trees (in future?)

Encryption of directory tree

- Not part of path-forward
- Server holds encrypted directories
- Similar key mechanisms
- Clients have access to un-encrypted directories

Location dependent security

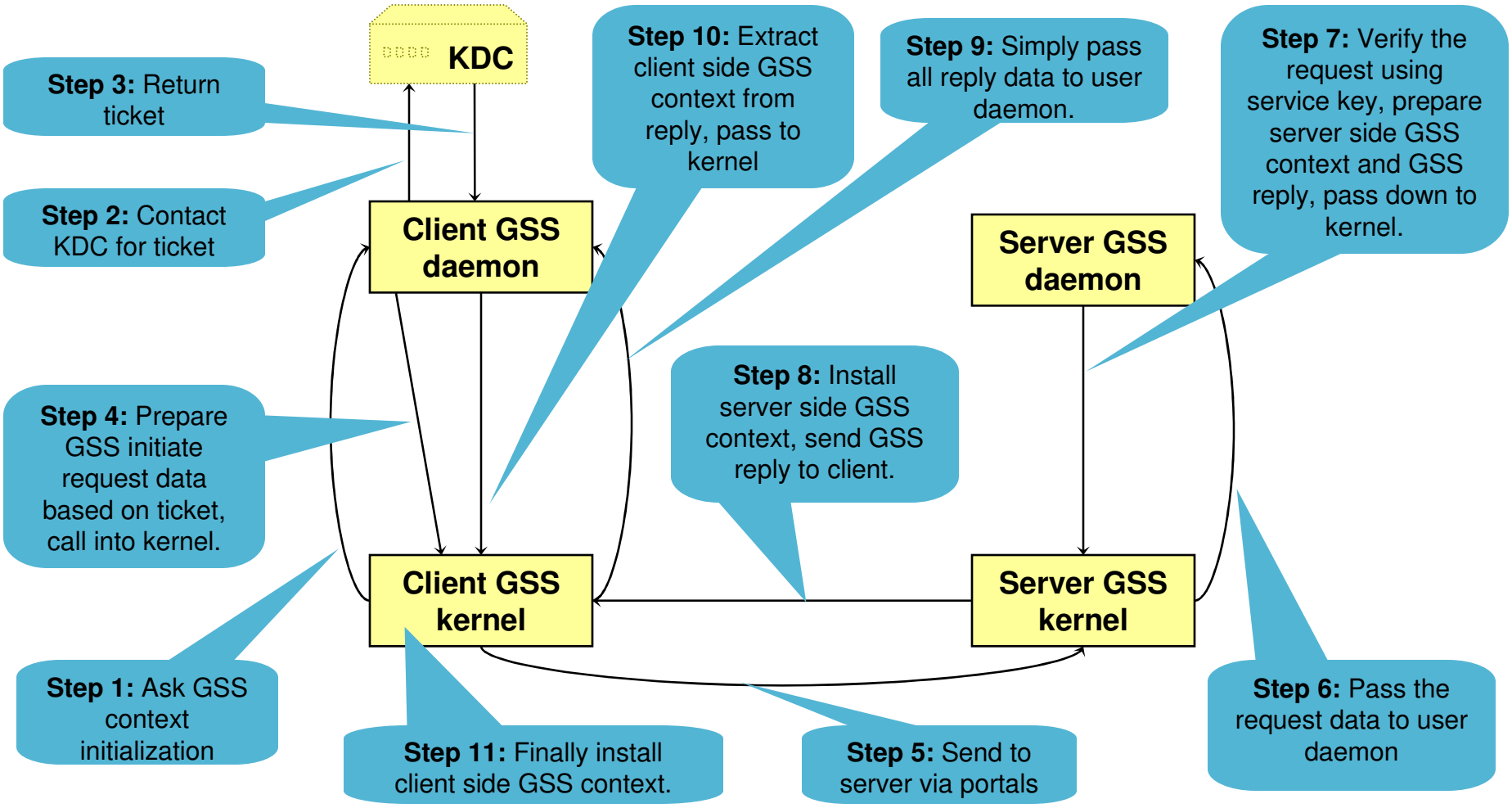
- Not part of path-forward
- OST & MDT pools
  - Local clients can often be determined reliably
    - See network policy slide
  - With strong network control Lustre source net can be reliable
  - Often deserve preferential treatment
    - e.g. RW access instead of RO
    - permissions to create instead of read
- Use this
- lookup and open implement different policies
  - e.g. directories appear RX or RWX
  - file open returns RO file handles

Secure handling of WB caches & proxies

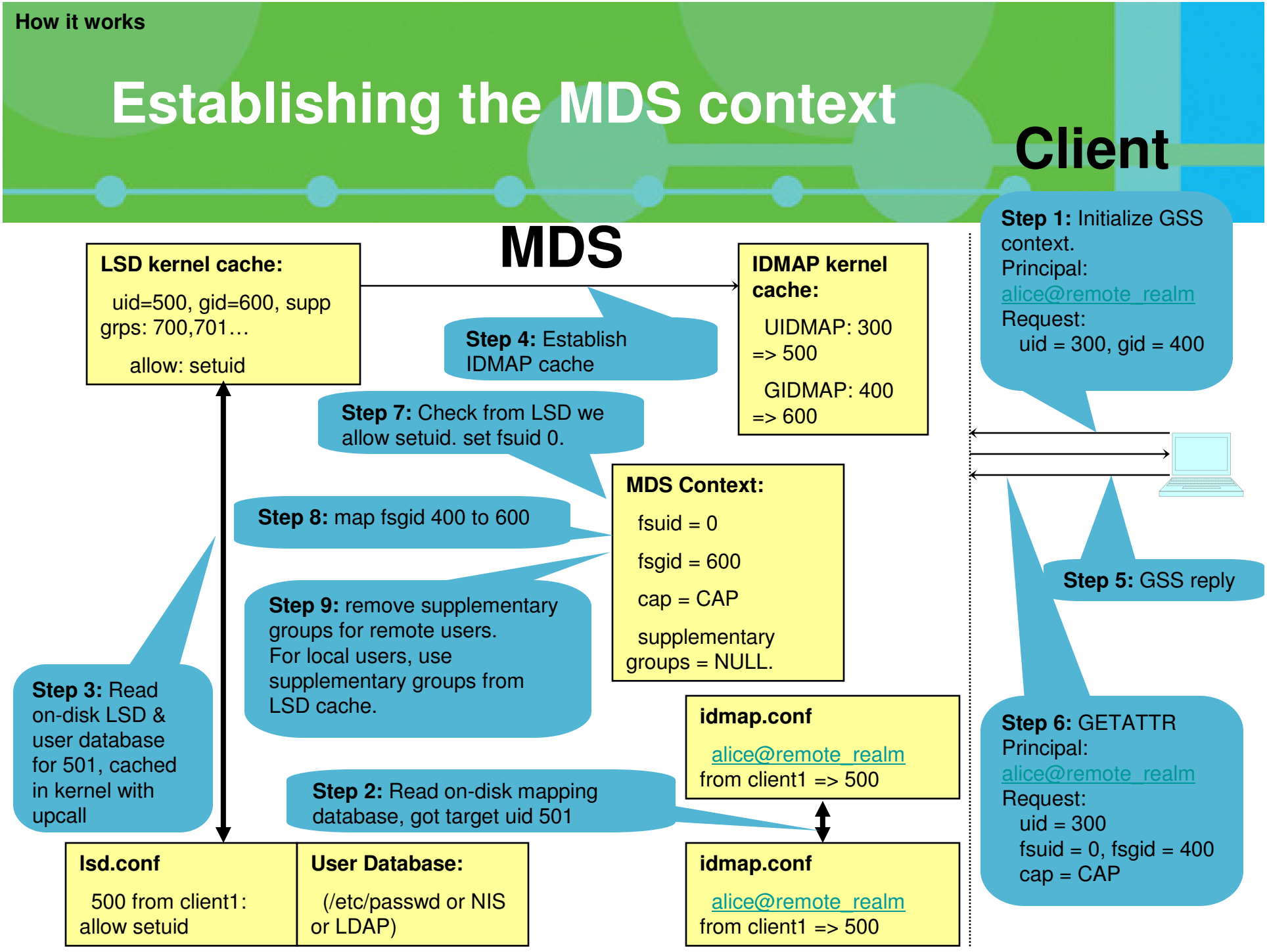
- Authorize operations on replay
  - Normal GSS & ACL handling
  - For client cache this forces single user WB caching
    - Absence of second user could block re-sync
- Proxies
  - Authenticate to up-stream servers
  - Get get special server privileges to replay

Operational Scenarios

GSS context handshake



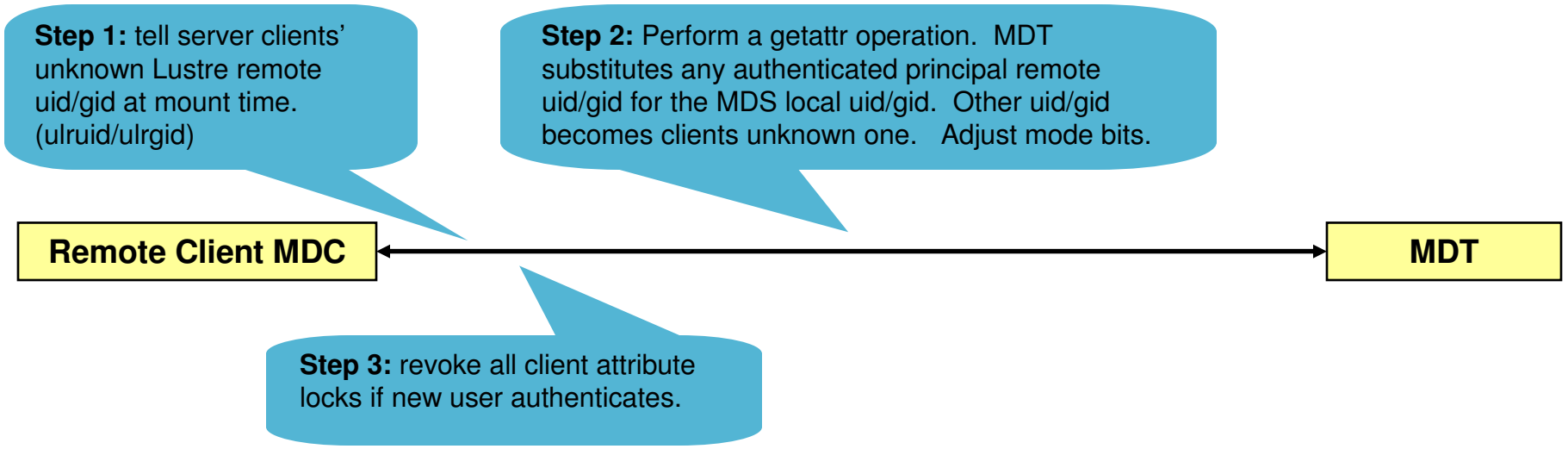
Establishing the MDS context



How it works

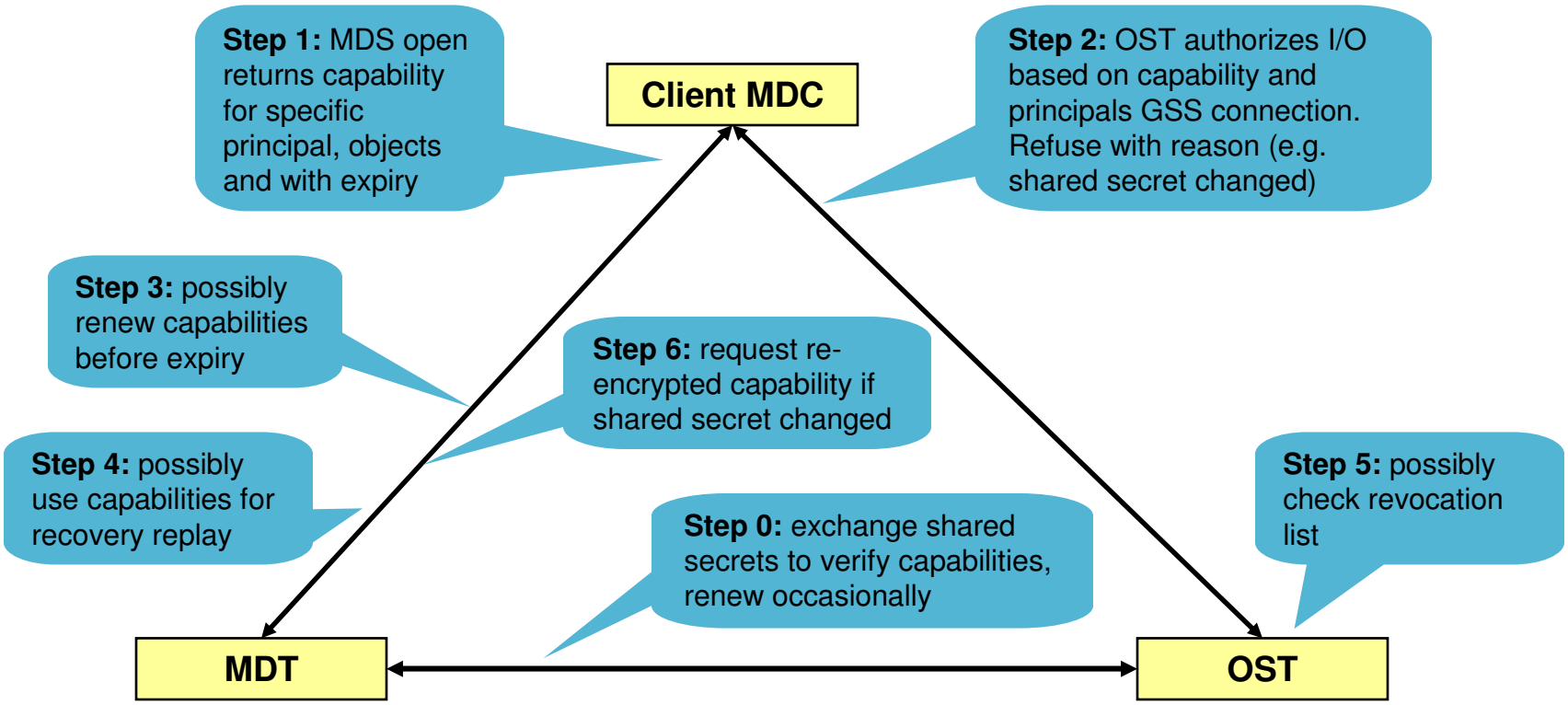
Listing a remote directory

```
gary-lanl lanl-lustre        12218 Jul 27  2004   design.txt*
lee       sandia-lustre     412778 Dec 26 15:48  colinux.tgz
```




```
gary   lustre        12218 Jul 27  2004   design.txt*
ulruid ulrgid       412778 Dec 26 15:48  colinux.tgz
```

Protecting objects from file I/O



File encryption – read / write of file