Plutus: scalable secure file sharing on untrusted storage

Mahesh Kallahalla
HP Labs

Joint work with
Erik Riedel (Seagate Research), Ram Swaminathan (HP Labs),
Qian Wang (Penn State), Kevin Fu (MIT)

March 31 2003

Why care about storage security?

How secure is your storage?

- Laptop ➡ Network ➡ Disk ➡ Tape
- Trust galore:
  - laptop / hard-disks can be stolen
  - network is shared
  - storage is outsourced
- Securing the network alone is not enough
- How much do you trust your administrator / ssp?

What's out there?

- Network security
  - doesn't have storage semantics
- Encrypt-on-wire
  - layered system
  - typically trusts server
  - NASD, iSCSI w/ IPSec, NFS w/ secure RPC
- Encrypt-on-disk
  - encrypt before it leaves clients
  - stored encrypted, typically used only for local storage
  - "sharing" problem not yet addressed
  - CFS, CryptFS, Truffles, Cepheus, Win EFS

Where are we going?

- Introduction
- Plutus
- Handling Revocations
- Implementation
- Summary

Our mantra

The fundamental design principles

- Decentralize key management and distribution
  - clients all the key related work
  - better scalability
  - better security
  - flexibility in policy
- Minimize server trust requirement
  - server can be broken into
  - anyway assuming untrusted network

Translation…

Encrypt-on-disk system

- Data on disk encrypted
  - files and (optionally) directory entries
- Data on network integrity protected
  - no need to re-encrypt bulk data

Client-centric system

- Client does most of the work
  - crypto, key management and distribution
- Server: store and retrieve bits from disk
  - validate writes

Plutus techniques summary

- Store all data encrypted
  - asymmetric read/write keys
- Owners distribute keys to share data
- Protect network integrity
- Server verified writes
- File-groups
- Lazy revocation
  - key rotation

Plutus techniques summary

- Store all data encrypted
  - asymmetric read/write keys
- Owners distribute keys to share data
- Protect network integrity
- Server verified writes
- File-groups
- Lazy revocation
  - key rotation

Plutus

- Store all data encrypted
  - say we use something like DES
- Owners distribute keys to share data
- Protect network integrity
- Server verified writes

Attack:

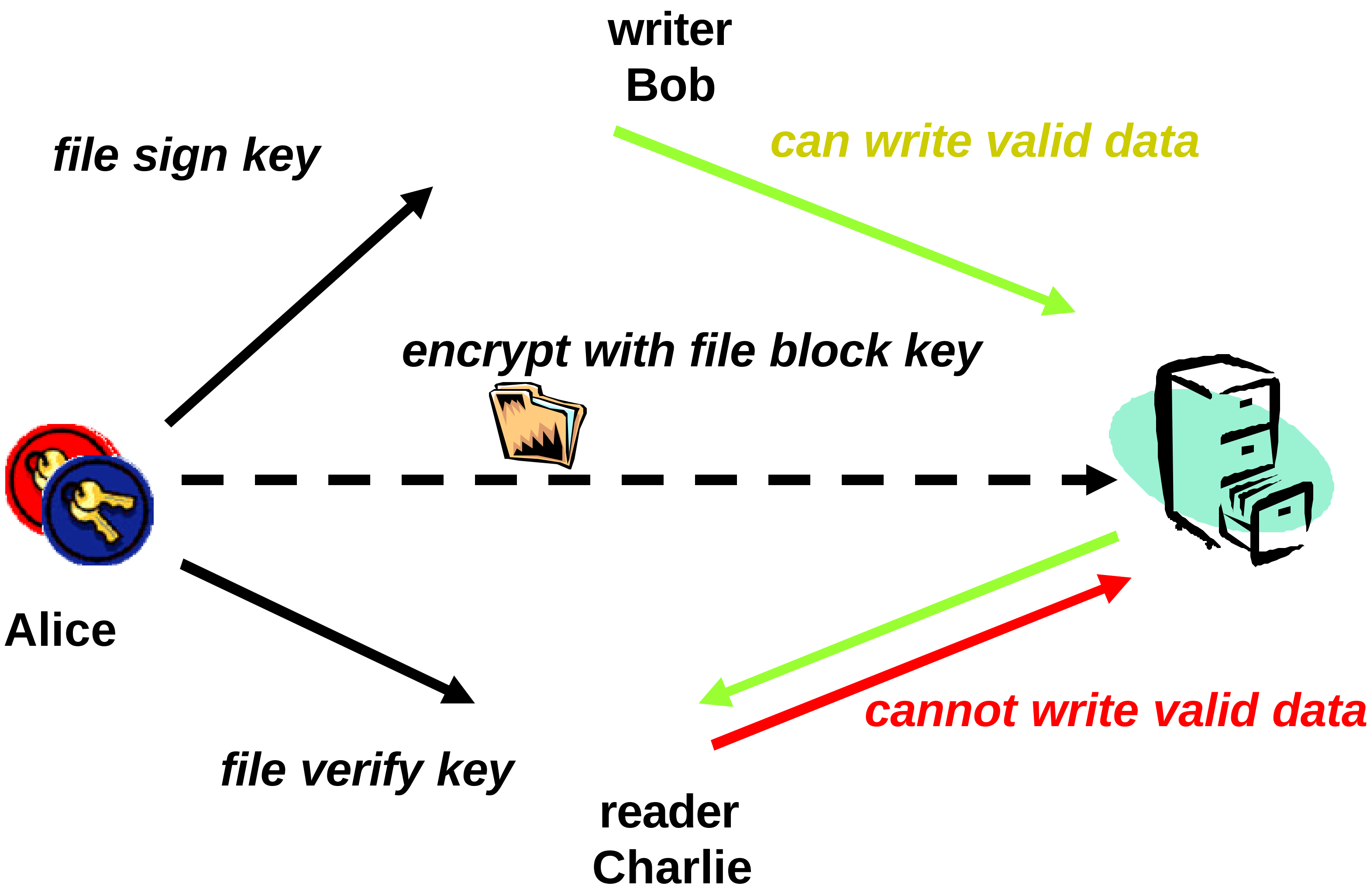
- Reader and server collude to update data
  - a data modify attack

Preventing readers from writing: asymmetric encryption

- Observations
  - Readers and writers need to be given different keys
  - Readers need to see what writers have written
  - Try converting into data destroy attack
    - should only be able to detect it

- Solution: asymmetric keys
  - Encrypt file with “file block key”
  - Protect integrity with “file-sign / file-verify” keys

Differentiating read/write access



Overhead: Too many keys!

- Problem
  - Key distribution overhead
  - Clients not always online
  - Key generation overhead

- Observation: use same key on multiple files

- Solution: file-groups
  - Use same key for files with similar sharing
  - Translating to unix
    - file-group = files with same owner, group, mode bits
    - study: appx 30 keys per user

You're outa here!
revocation

- How to revoke a user's access to a file

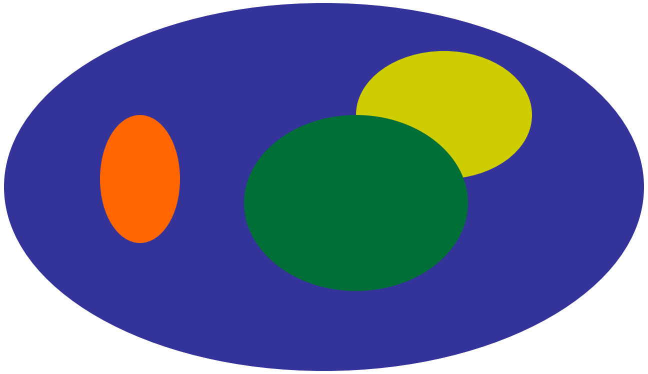
- Change keys and hand them out again
  - re-encryption effort
  - key re-distribution effort

You’re … yawn … outa … zzz …here lazy revocation

- Observations
  - might be ok to leave unchanged data open
- Solution: lazy re-encryption
  - On revocation
    - change keys
    - mark files for re-encryption
  - Only re-encrypt when written next

Complication:
lazy revocation + file-groups

- File-groups
  - same key multiple files
- On write following revocation
  - key for re-encrypted file different!
- Don’t want explosion in keys due to revocations

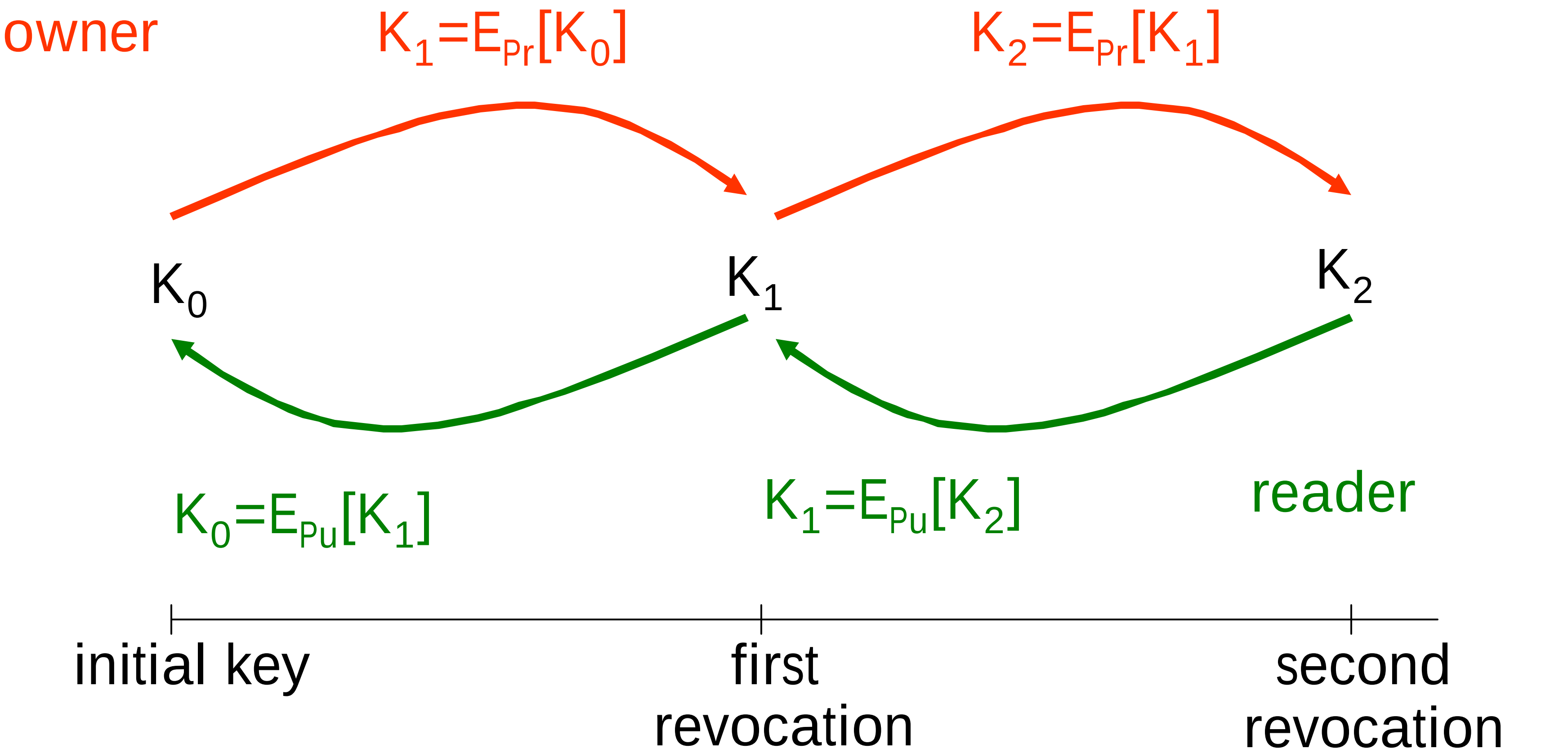


Recursive keys

- Keys for a file-group
  - rotate file lockbox key
  - only owner can generate next in sequence
  - given current key, readers generate previous keys

- On revocation owner generates new keys
- Writers get latest file-sign key
- Readers are given latest file lockbox key
  - rotate back to get older keys on seeing older files
  - derive file-verify keys from corresponding lockbox key

Key-rotation



Key-rotation: the math

Owner (and reader)

- generate random prime e greater than sqrt(N)
  - use file lockbox key as seed

Owner

- generate corresponding “d” (file-sign key)

Writer

- given “d”

Note

- Reader cannot get file-sign key
- Writer cannot get file-verify key

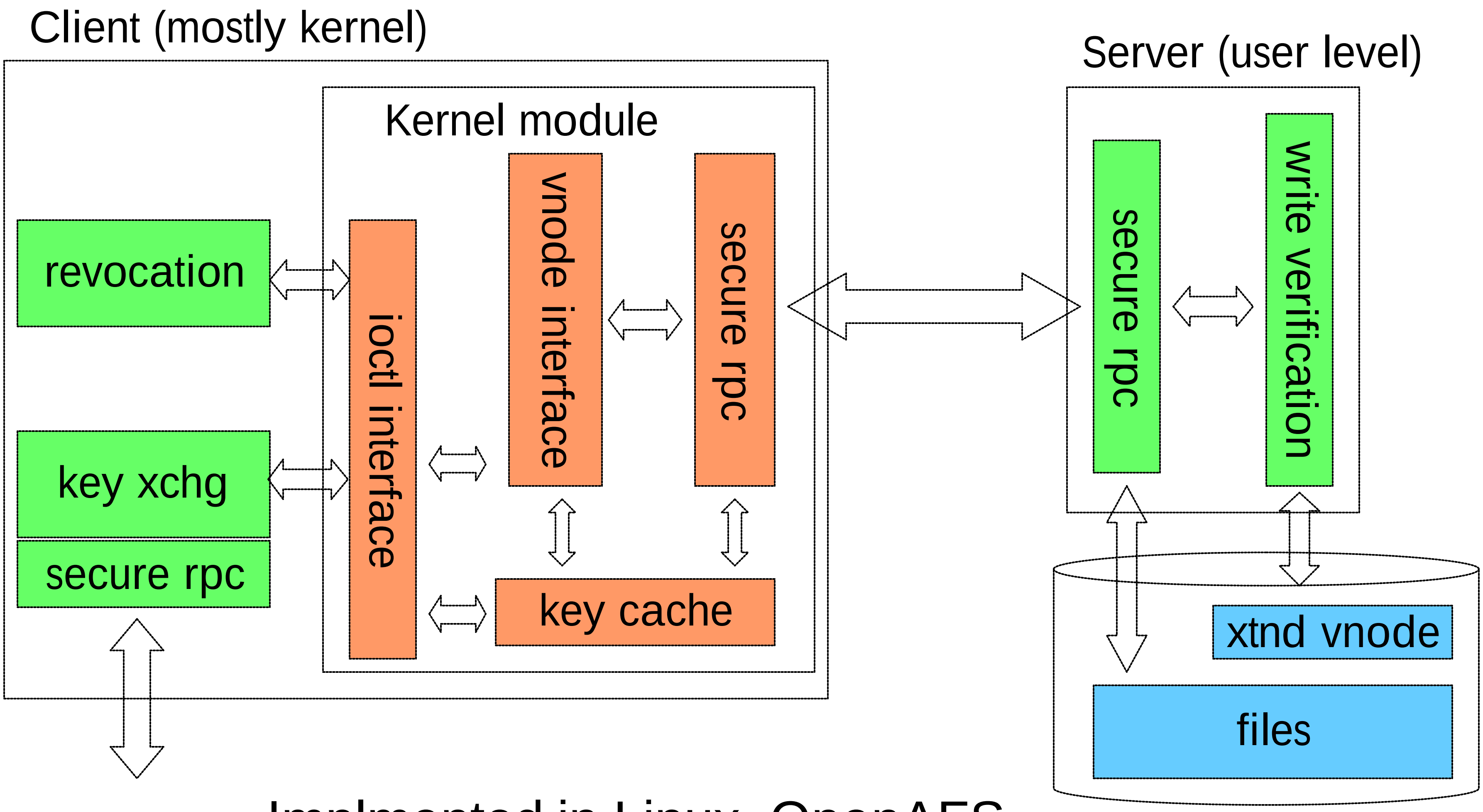
Design summary

- Store all data encrypted
  - asymmetric read/write keys
- Owners distribute keys to share data
- Protect network integrity
- Server verified writes
- File-groups
- Lazy revocation
  - key rotation

Where are we going?

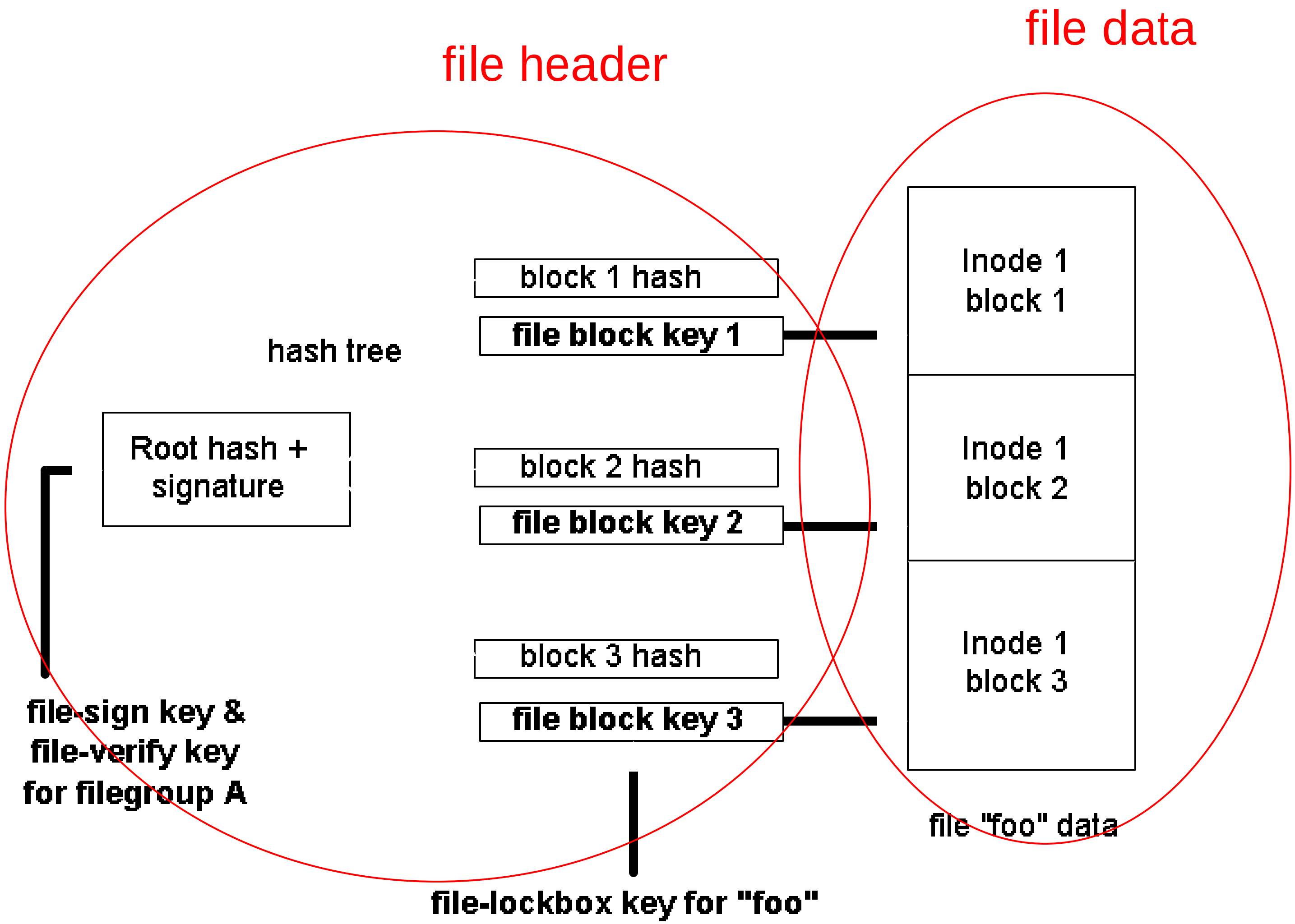
- Introduction
- Plutus
- Handling Revocations
- Implementation
- Summary

AFS incarnation



- Implmented in Linux, OpenAFS
- Crypto: SHA1, RSA1024, 3DES

Key summary – the file side



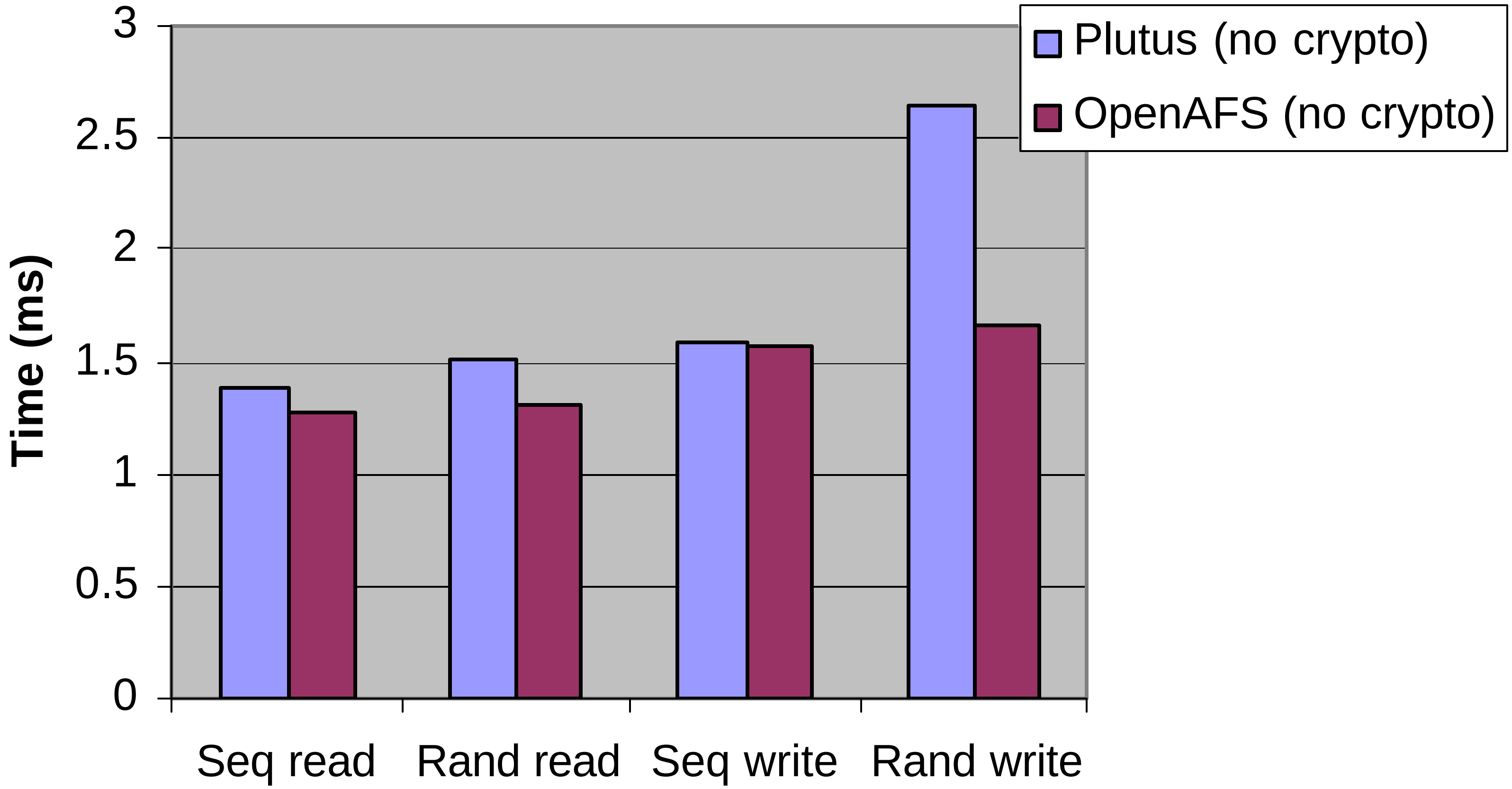
How long does it take

- RSA key generation
  - 2500 ms paid by owner on every filegroup creation
- File read/write
  - per 4KB block: 0.7 ms of crypto
  - per file: 28.5 ms by writer, 8.5 ms by reader

  - write verification: 0.01ms per file

1.26GHz P3 with 512MB ram: SHA1, RSA1024, 3DES

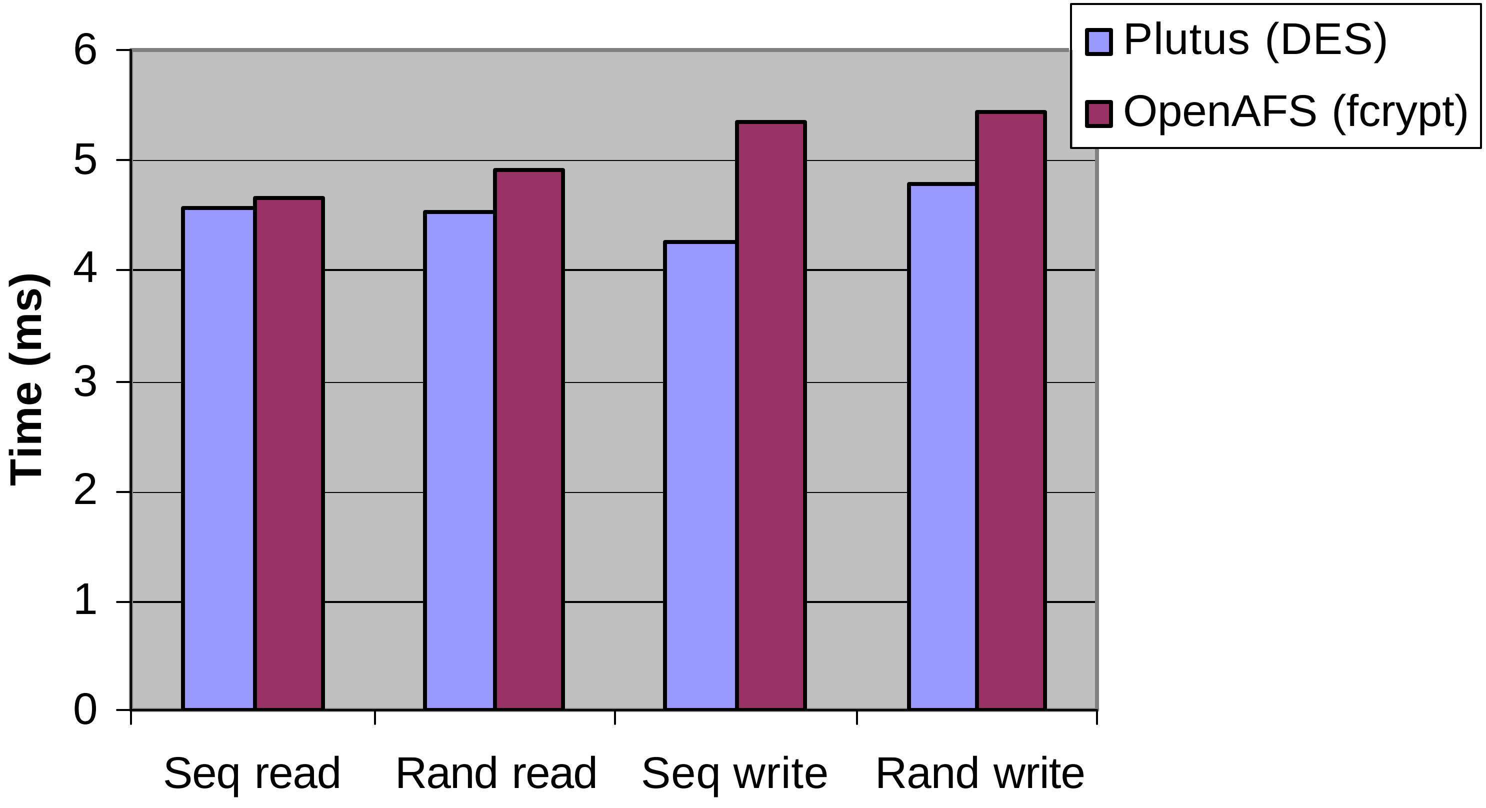
Data structures overhead



Accessing a remote 40 MB file

- Hash tree structure doesn’t take much time

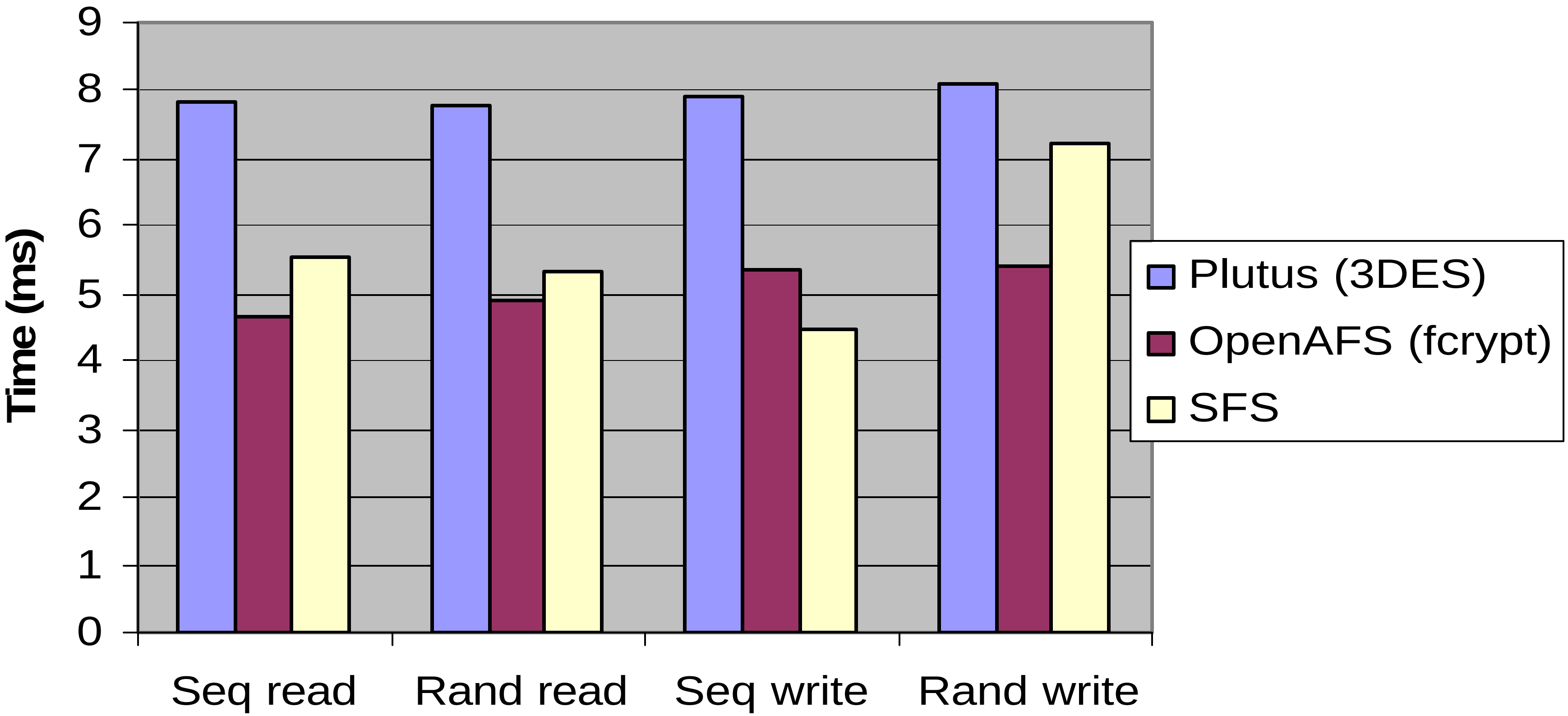
Encrypt-on-disk vs. wire-encryption



Accessing a remote 40 MB file

- Comparable performance

Plutus performance (overall)



Accessing a remote 40 MB file

OpenAFS's fcrypt comparable to DES
SFS's ARC4 14X faster than 3DES

- Plutus compares favorably with SFS

The key takeaways

1. Use keys to pass “access rights”
   + don’t need to maintain lists
   – complicates revocation
2. File groups: use same keys for similarly shared objects
   + simpler key management
3. Key rotation: keys can be related but secure
   + useful for evolving keys
4. Lazy re-encryption
   - don’t do the crypto till absolutely necessary