



Zurich Research Laboratory

Security in Networked Storage Systems

Christian Cachin <cca@zurich.ibm.com>

Overview

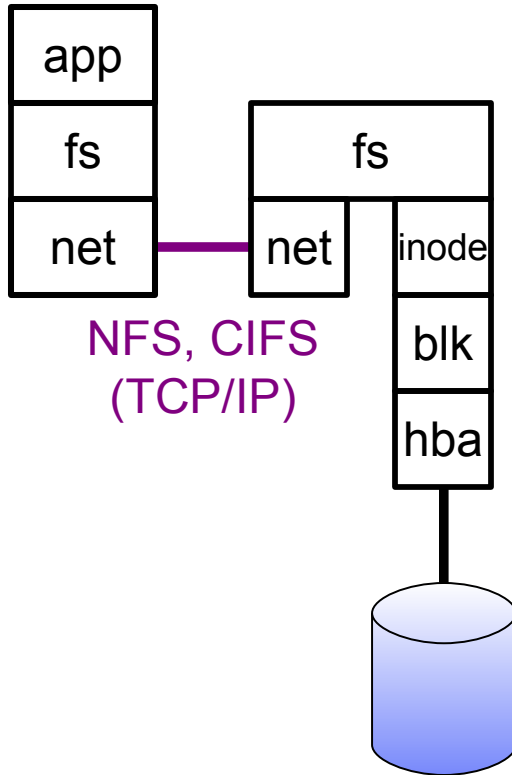
- Networked storage systems
- Design options for security
- Encryption and key management
- Integrity protection

Traditional Storage Systems

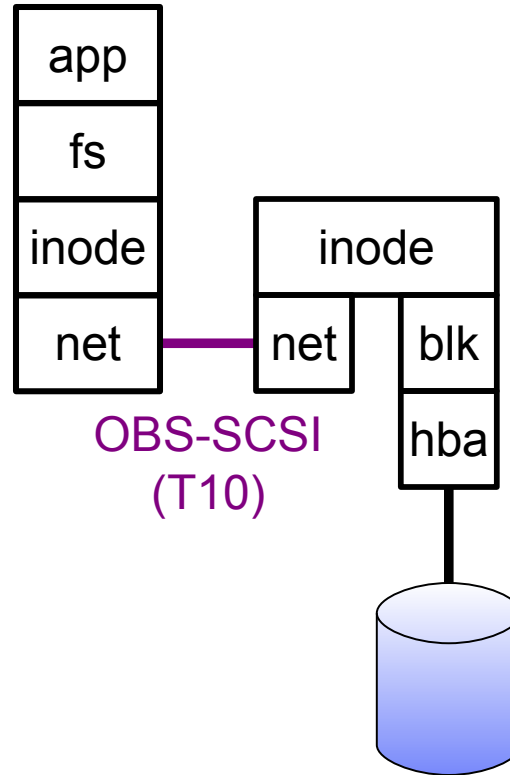


Direct-attached Storage

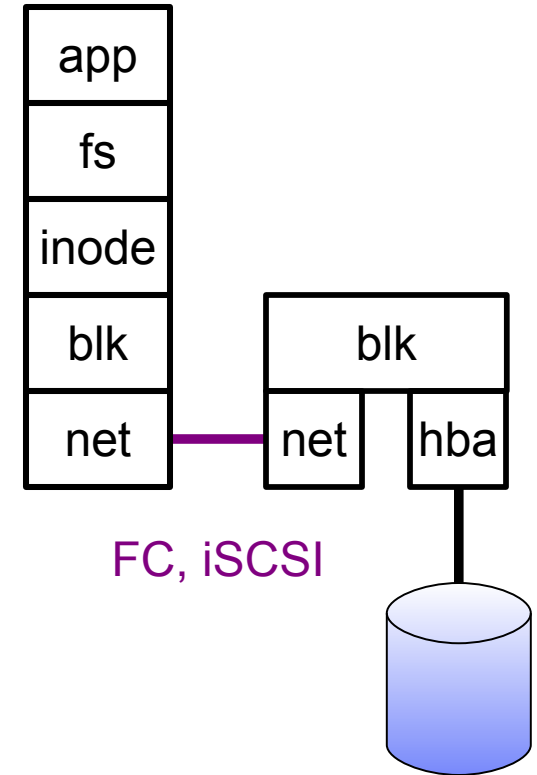
Networked Storage Systems: NAS, OBS, SAN



NAS
(Network-attached Storage)

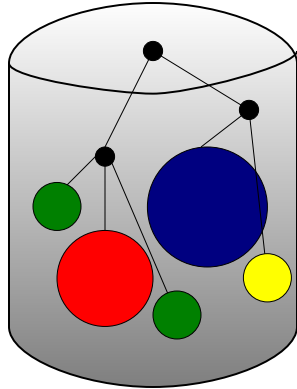


OBS
(Object Storage)



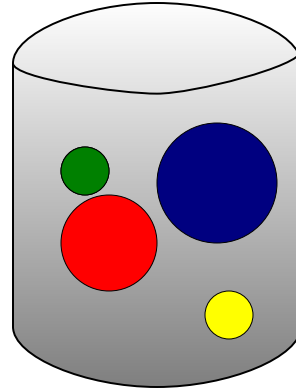
SAN
(Storage-area Network)

Network-based Storage Devices



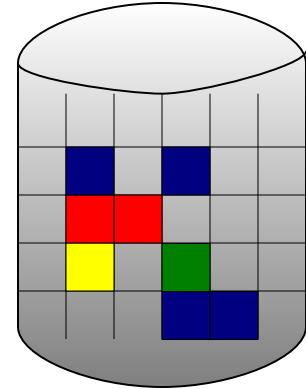
File server

- read & write data in file
- create & destroy file
- directory operations
- file/dir-based access control
- space allocation
- backup ops



Object storage dev.

- read & write bytes in object
- create & destroy object
-
- object-level access control
- space allocation
- backup ops



Block device

- read & write blocks
-
-
- device-level access control
-
-

Security in Networked Storage Systems

- Existing technology offers little protection
 - Server room only
 - Trusted storage providers, networks, and clients
 - Coarse-grained access control
- Security is needed
 - Storage as a commodity
 - Networked storage to desktop (iSCSI)
- Threats
 - physical access to disks
 - access to network
 - authorized machines
 - unauthorized machines
 - ...

Design Options for Security

Security Toolbox

■ Goals

Confidentiality (no unauthorized access)

Integrity (no unauthorized modification)

Availability

■ Security mechanisms

Encryption

→ Confidentiality based on shared key k

Message-authentication code (MAC)

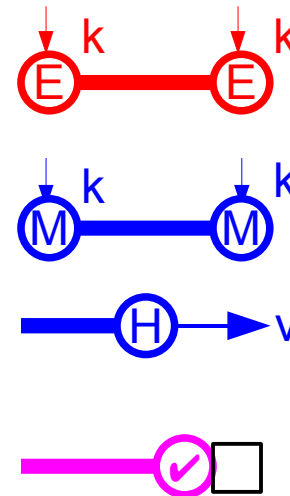
→ Integrity based on shared key k

Hashing and digital signatures

→ Integrity, w.r.t. reference value v

Access control

→ Confidentiality, integrity, availability



■ Any mechanism may be applied on any layer

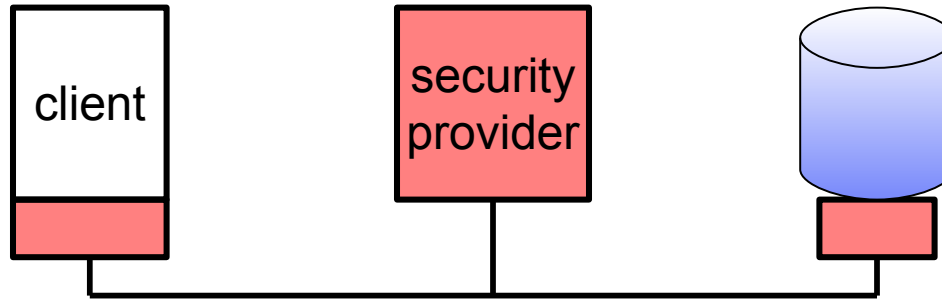
Any Security Mechanism May Be Applied on Any Layer

- Storage systems have these layers for good reason
 - Not all security mechanisms are useful and efficient on all layers
- Challenge is to select the “right” combination

Generic Model of a Secure Storage System

■ Option 1: Protect data in flight

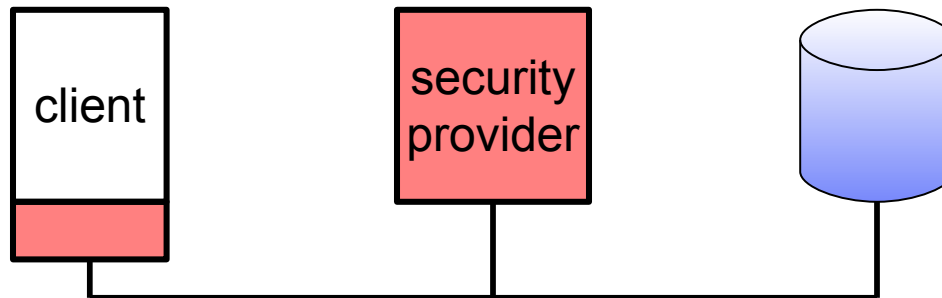
→ Trusted client, trusted storage (untrusted network)



■ Option 2: Protect data at rest

→ Trusted client (untrusted storage and untrusted network)

→ Allows DoS attack, data may be lost

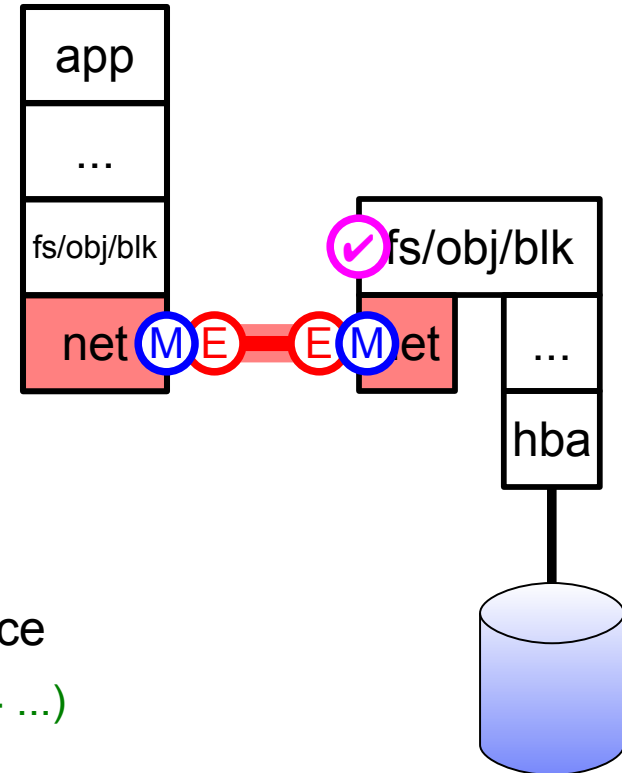


Security for Networked Storage Systems (1)

Option 1: Protect the data in flight

- ✓ Access control
- Ⓜ Integrity protection
- ⓔ Encryption

- Encrypt the communication
 - Session, transport or packet layer
 - Secure RPC, SSL, IPsec, FC-SP ...
- Layer-specific access control on storage device
 - NAS at filesystem layer (exists in AFS, NFSv4 ...)
 - ObjectStore at object layer (in standard)
 - SAN at block layer (proposed)

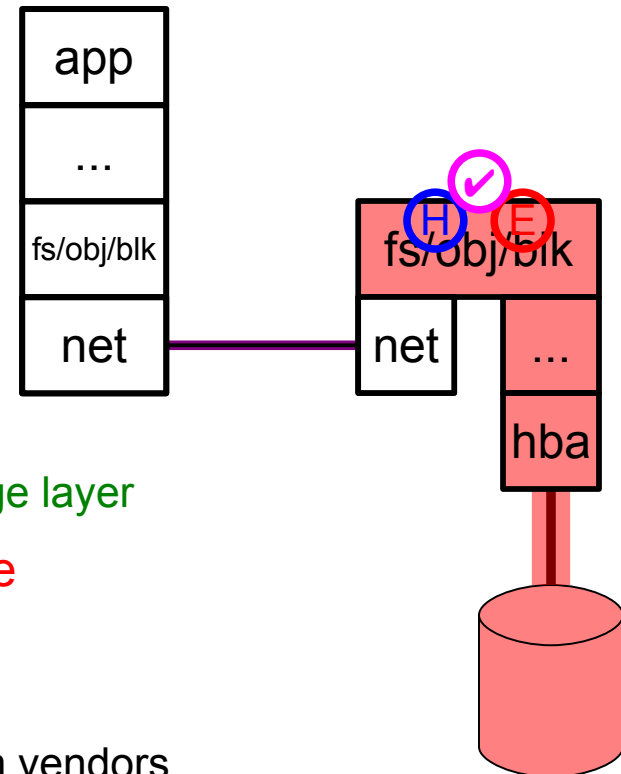


Security for Networked Storage Systems (2)

Option 2: Protect the data at rest

- ✓ Access control
- Ⓗ Integrity protection
- Ⓔ Encryption

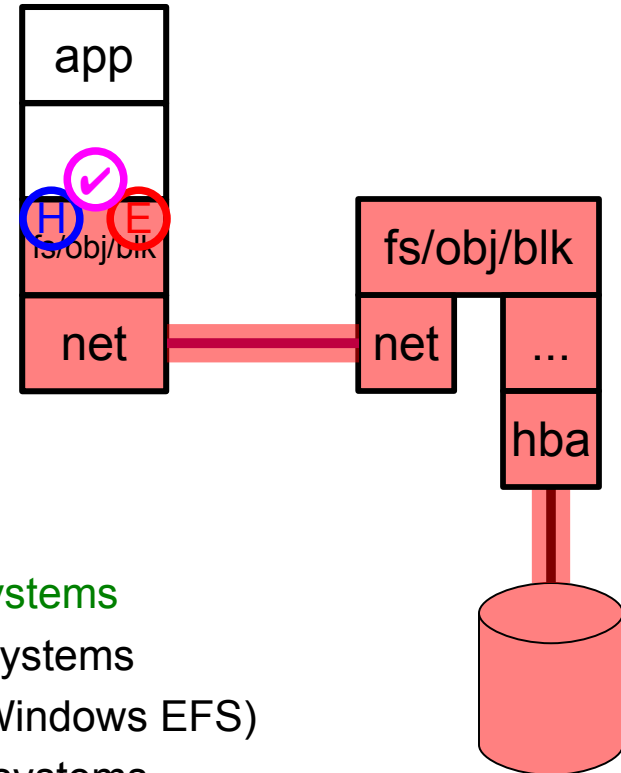
- Encrypt the storage space
 - Encryption and integrity protection for a storage layer
- Layer-specific cryptography **on storage device**
 - Typically on low layers: block encryption
 - Upcoming disk storage systems
 - Available today as security appliance from vendors Decru/NetApp or NeoScale



Security for Networked Storage Systems (3)

Combining Options 1 & 2: Protecting data in flight & at rest

- Encrypt the storage space
 - But don't trust the network
and don't trust the storage device
- Layer-specific cryptography **on client**
 - Typically on higher layers: cryptographic filesystems
 - Available today in local cryptographic filesystems
(CFS, SFS, Linux loopback encryption, Windows EFS)
 - Not yet widely available for distributed filesystems



Design Dimensions

- **Encryption: keys?**

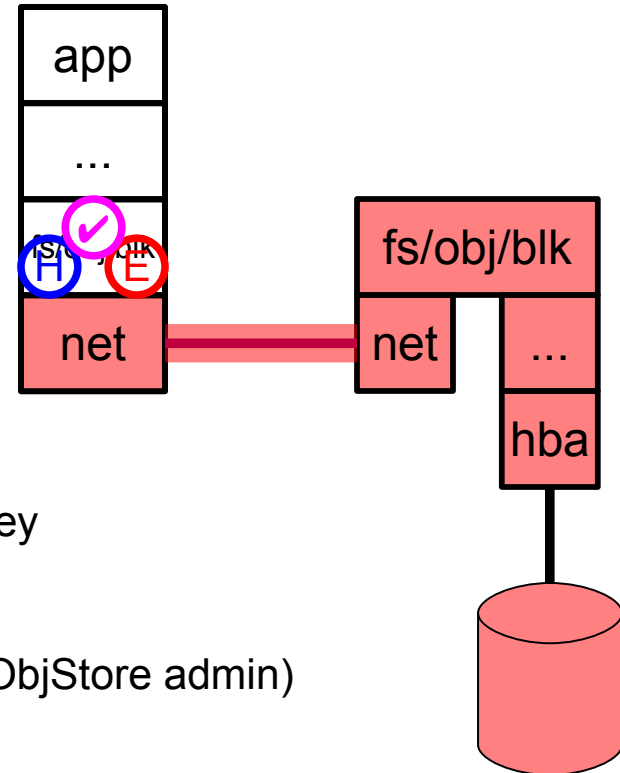
- Separate security admin server
 - Encrypted with user/group public key
 - Held by hardware module

- **Integrity verification: reference values?**

- Integrated in directory
 - Inode tree is hash tree
 - Digital signatures under user/group public-key

- **Access control: credentials?**

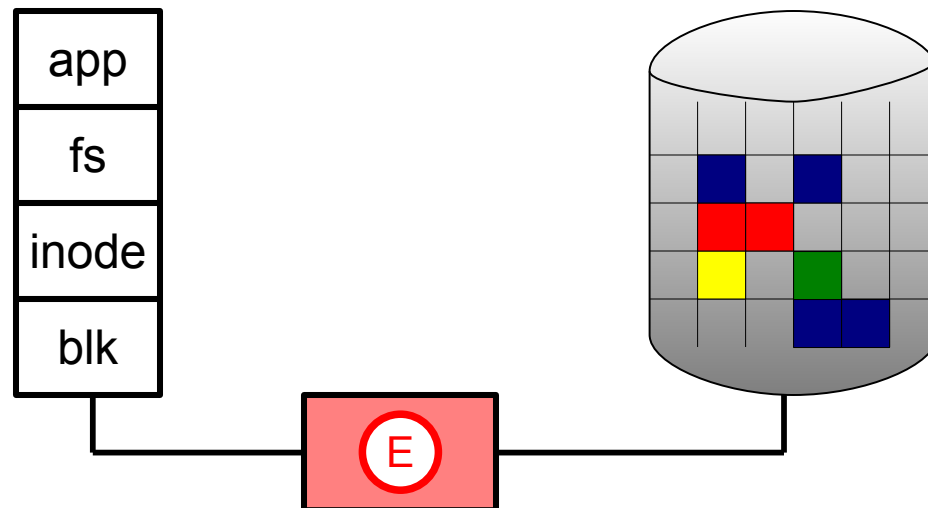
- Separate security admin server (Kerberos, ObjStore admin)



Encryption and Key Management

Encryption at the Block Layer

- “Sector” encryption, 512-byte blocks
- Transparent to storage system → no extra space available



- IEEE SISW standardization effort: P1619, P1619.1, ...

Key Management in Cryptographic File Systems

■ Two approaches

On-line and centralized

- Only symmetric-key crypto
- Simple and efficient
- Limited scope and scalability
- Ex. Cryptographic SAN.FS [PC06]

Off-line and de-centralized

- Requires public-key crypto
- Complex, computationally expensive
- Scalable
- Ex. SFS [FKM02], Windows EFS, Plutus [KRS+03], Sirius [GSMB03] ...

De-centralized Key Management

- Users have SK/PK pair
- Groups have SK/PK pair; every member of group knows SK
- Files encrypted using FEK with block cipher
- Confidentiality: Store FEK encrypted in meta-data
 - Encrypted under every PK of every user/group that has access

Example: File X, encrypted with FEK_X

```
owner: A, rwx,  $E_{PK_A}(FEK_X)$ ,  
group: G, rw-,  $E_{PK_G}(FEK_X)$ ,  
world: ---
```

- Integrity: Add FSK_X / FVK_X , key pair for digital signatures, to X
 - Store FSK like this in every encrypted file
- Drawback: key revocation is tedious

Key Revocation

- User revoked → change all keys that were known to user
 - Re-encrypt all data with fresh keys
- Very expensive and disruptive operation
- Idea: **Lazy Revocation** [F99]
 - Re-encrypt data only when it changes after revocation, keep old keys around.
- All versions of a key must remain accessible.

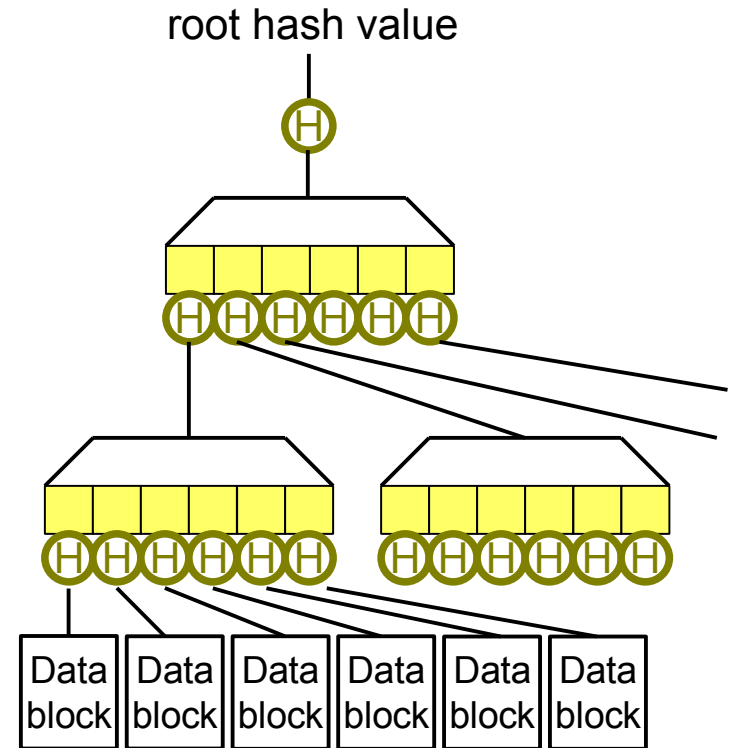
Integrity Protection

Integrity for Data-at-Rest

- Storage server not trusted
- Associate short reference value v with long file
 - Store v on trusted server, with file meta-data
 - Sign v with digital signature
- Hash function?
 - $v = H(\text{file})$
 - Infeasible for long files
 - No random access
- Solution:
 - Hash tree [Merkle]

Integrity Protection Using Hash Trees

- Merkle hash trees
 - Root hash value represents all data blocks of the file
 - Root hash value in trusted storage
 - Tree stored on untrusted storage
- Reads and updates take $O(\log n)$ extra operations
- With local buffering, sequential read or update of all blocks has **constant overhead**



Implementing Hash Trees

- Much more complex than encryption in file system
 - Dual and mutually dependent data paths
- Degree may vary (2 ... 128), determine experimentally (≈ 16)
- Serialize nodes using pre-order enumeration
 - Sparse allocation of maximum-size tree
 - Takes care of file holes

