Zurich Research Laboratory

Cryptographic Protection for Networked Storage Systems

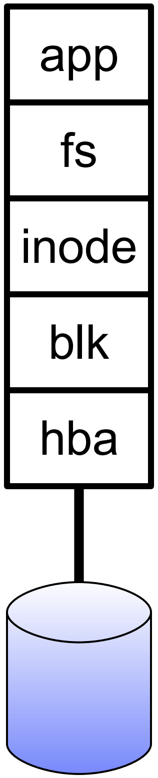
Christian Cachin <cca@zurich.ibm.com>

2 November 2006

Overview

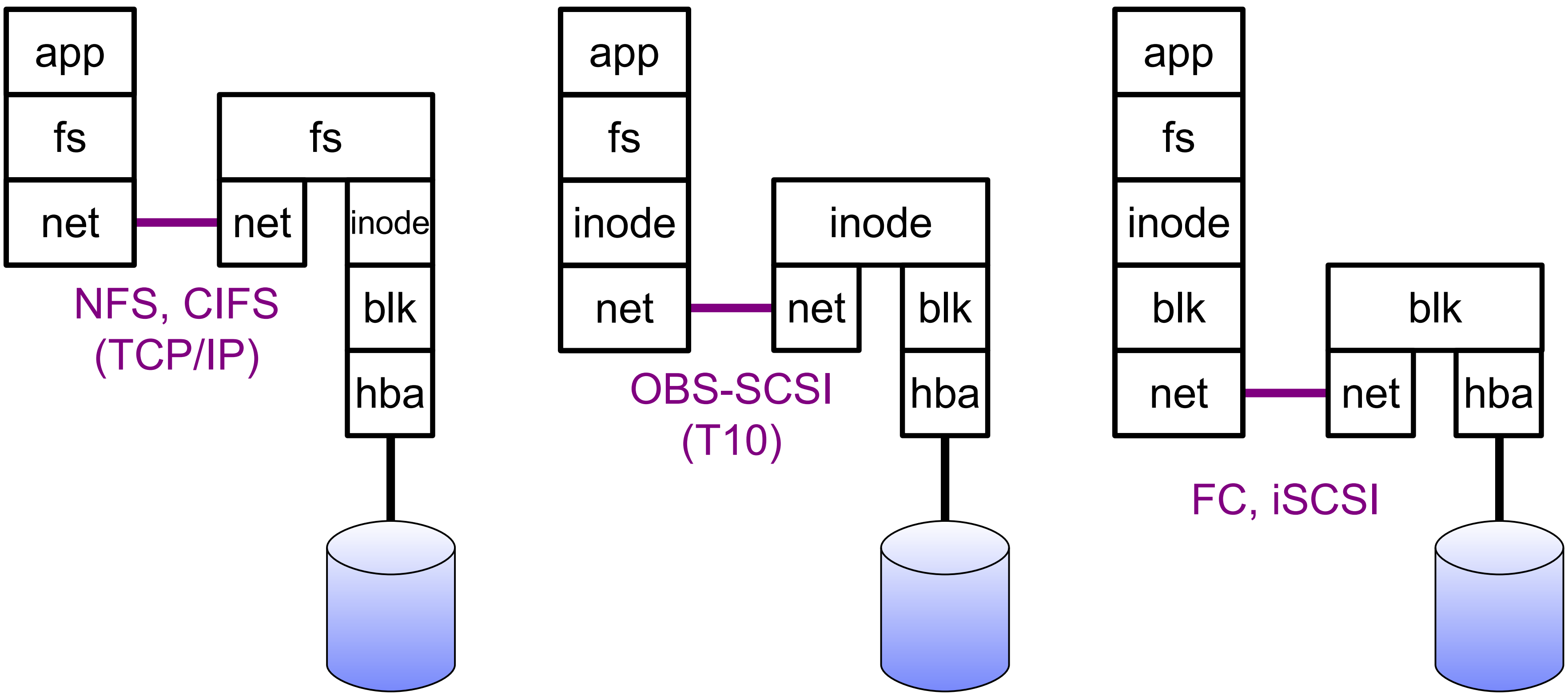
- Networked storage systems
  - → NAS, SAN, OBS
- Design options for security
  - → Data in flight & data at rest
- Block layer
  - → Tweakable encryption modes
  - → Integrity protection using tweakable encryption
- Object layer
  - → Capabilities in Object Store
- Filesystem
  - → Designs for key management
  - → Encryption using lazy revocation and key updating
  - → Integrity protection using hash trees
- Example: a cryptographic SAN file system

Traditional Storage Systems



Direct-attached Storage

Networked Storage Systems: NAS, OBS, SAN

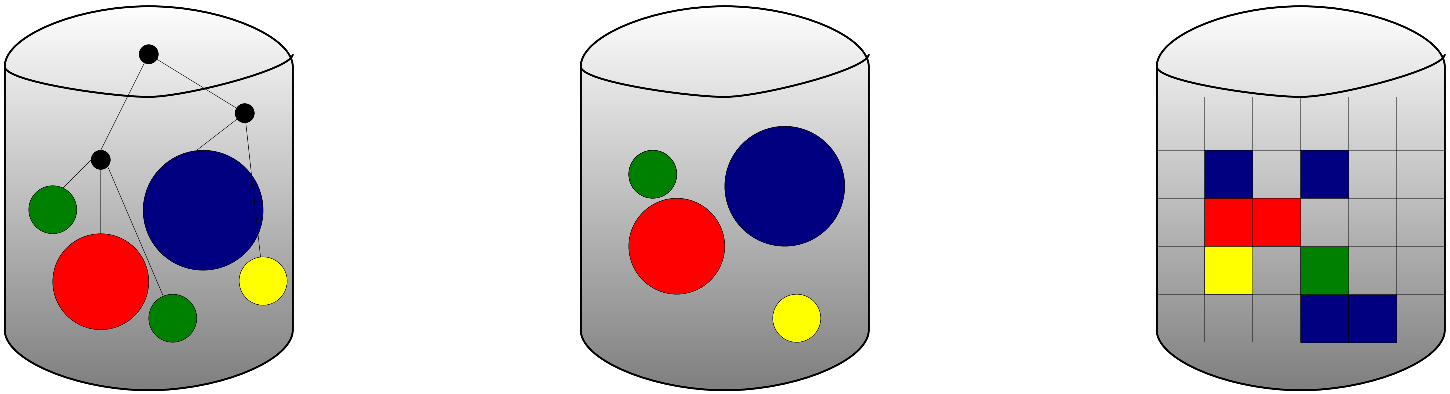


NAS
(Network-attached Storage)

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

--

--

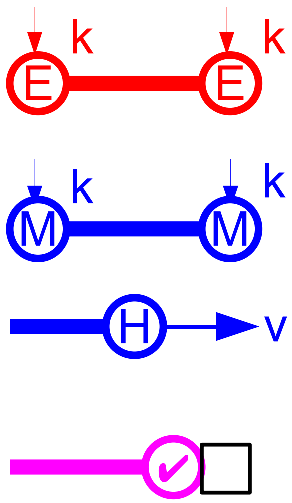
Security in Networked Storage Systems

- 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

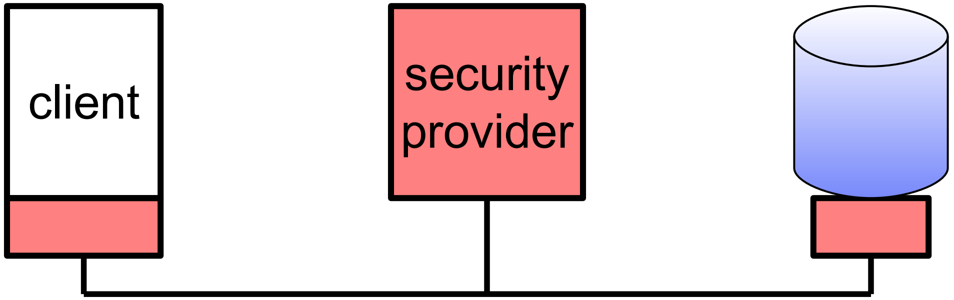
Any Security Mechanism May Be Applied on Any Layer

- 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
- Talk outline:

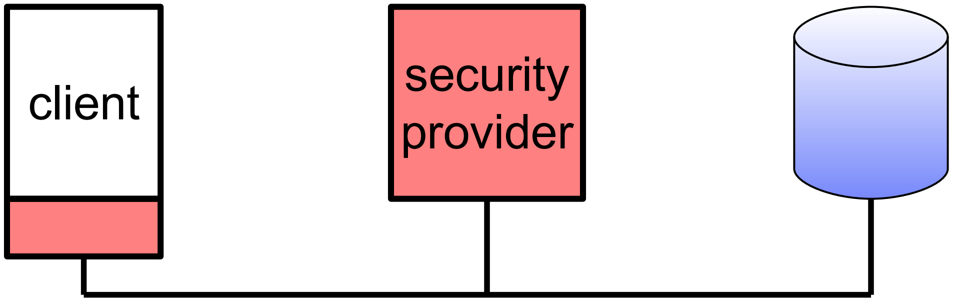
	Ⓔ	Ⓜ	✔
file	key mgmt. & lazy revocation	hash trees	
object			OBS security protocol
block	tweakable block encryption	hybrid block-integrity protection	

Generic Model of a Secure Storage System

- Option 1: Protect data in flight
  - → Trusted client, trusted storage (untrusted network)



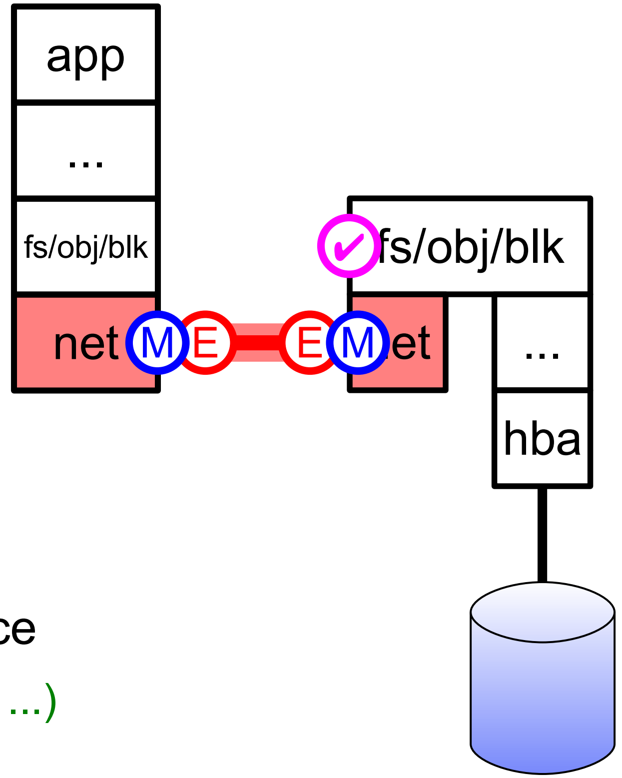
- Option 2: Protect data at rest
  - → Trusted client (untrusted storage and untrusted network)
  - → Allows DoS attack, data may be lost



Security for Networked Storage Systems (1)

Option 1: Protect the data in flight

- ✔ Access control
- M Integrity protection
- E Encryption

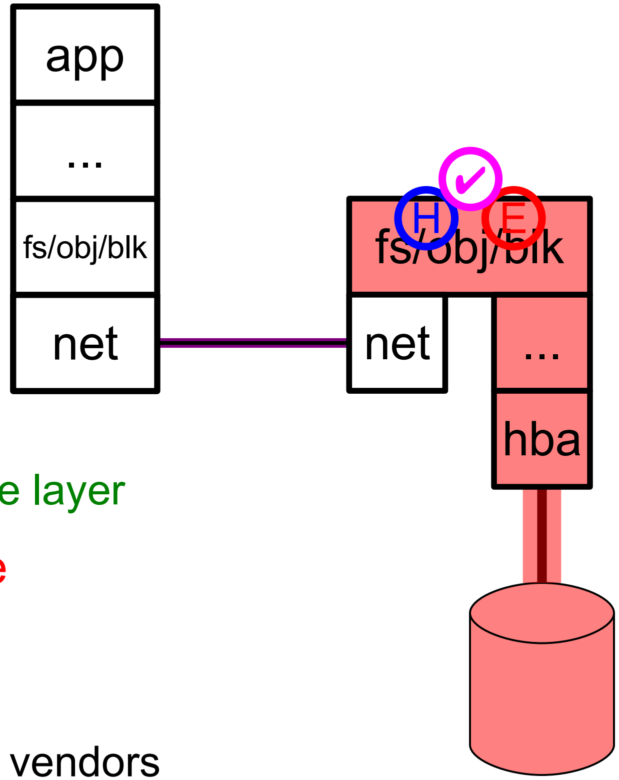


- Encrypt the communication
  - → Session, transport or packet layer
  - → Secure RPC, SSL, IPsec, FC-SP ...
- Layer-specific access control on storage device
  - → NAS at filesystem layer (exists in AFS, NFSv4 ...)
  - → ObjectStore at object layer (in standard)
  - → SAN at block layer (proposed)

Security for Networked Storage Systems (2)

Option 2: Protect the data at rest

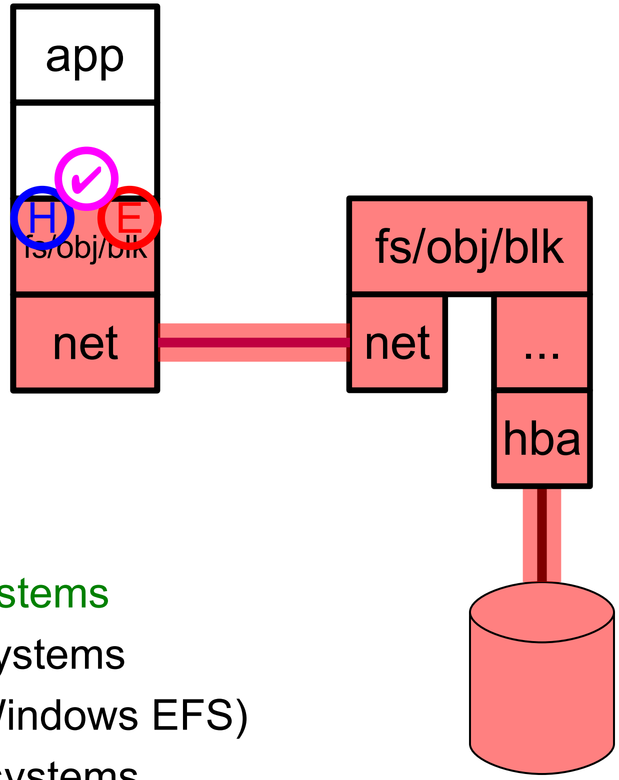
✔ Access control
H Integrity protection
E Encryption



- 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)

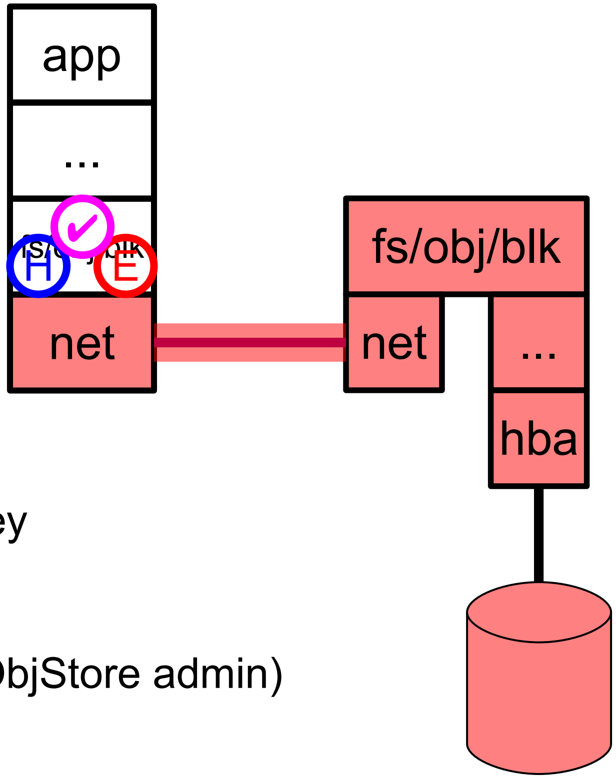
Combining Options 1 & 2: Protecting data in flight & at rest



- 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)



Talk Outline

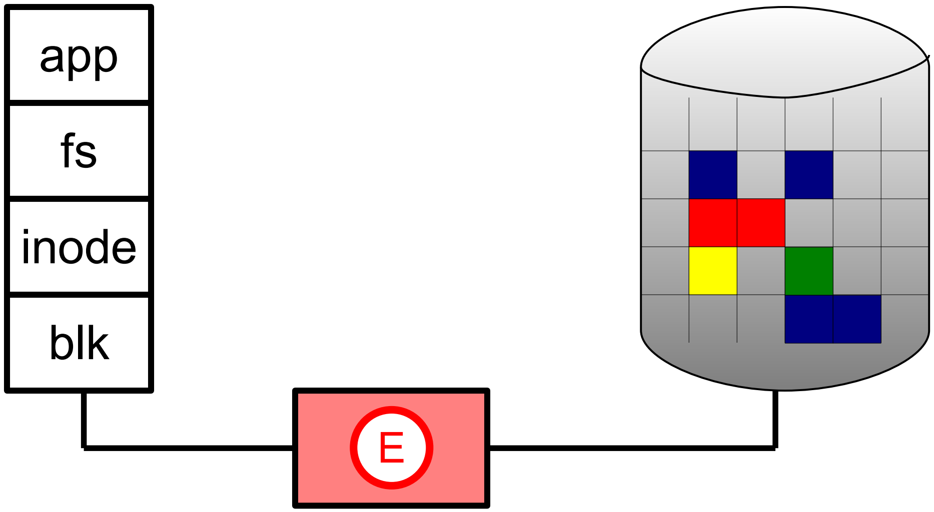
- Storage systems have these layers for good reason
  - → Not all security mechanisms are useful and efficient on all layers
- Challenge is to select the "right" combination

	Ⓔ	Ⓜ	✔
file	key mgmt. & lazy revocation	hash trees	
object			OBS security protocol
block	tweakable block encryption	hybrid block-integrity protection	

Block Layer

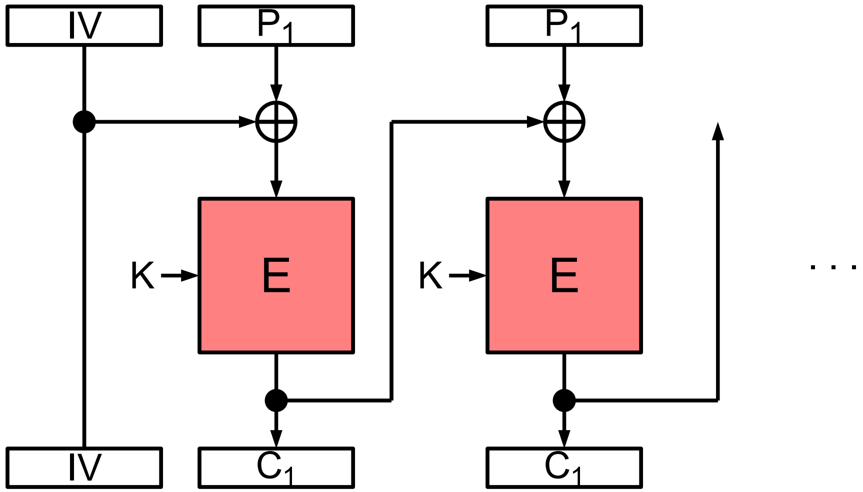
Encryption at the Block Layer

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



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

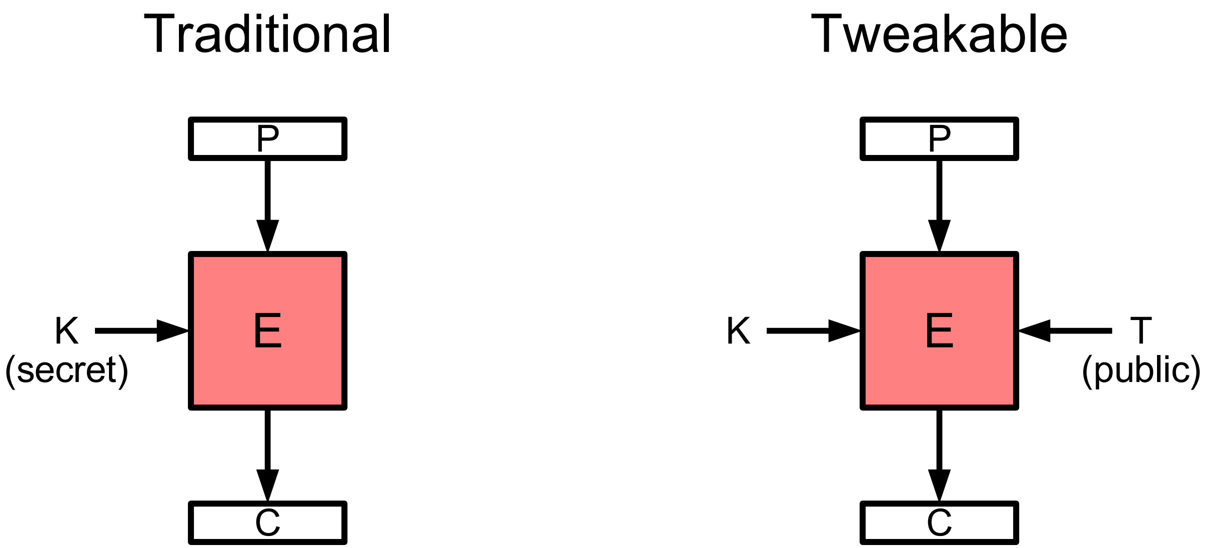
Using CBC Mode



- IV chosen at random → must be stored, doesn't work
- Derive IV from offset of sector on disk

  IV = E_K(sector offset | disk LUN)

Tweakable Block Encryption [LRW02]



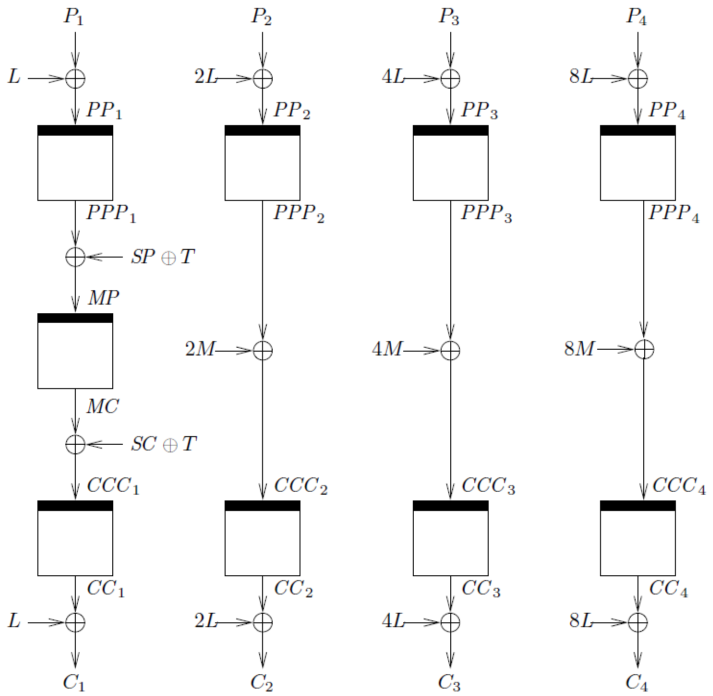
E_K() is PRP

$E_{K,T}$() is a PRP for every T

- E_K() is a deterministic permutation (after picking K)
- Tweakable $E_{K,T}$() is a family of independent such permutations
  - → T = LUN | offset of sector on LUN
- Change of even one bit → decrypted P' completely independent of C

Using Tweakable Encryption Mode

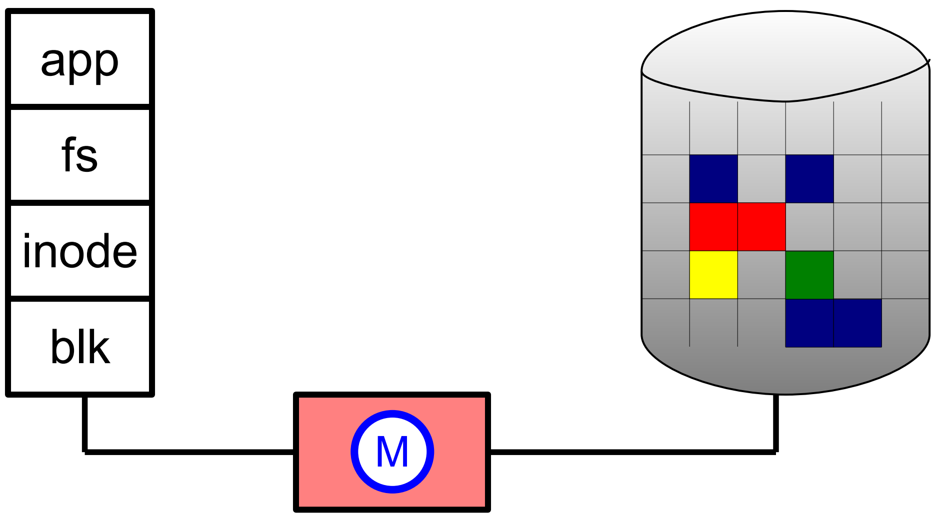
- Turns an ordinary narrow-block cipher E (16-byte blocks) into a tweakable, wide-block cipher (512-byte blocks).
- EME [HR04], calls to E are parallelizable:



- EME requires ≈2 block cipher calls per plaintext block (better is ≈1)
- Mode by [LRW02] is more efficient, but less secure

Integrity Protection at the Block Layer

- No extra space available → really problematic for integrity
- All integrity protection and data authentication methods require extra space for a tag or a hash value



- If there was space, use a MAC or a hash tree ...

Hybrid Integrity Protection at the Block Layer [ORY05]

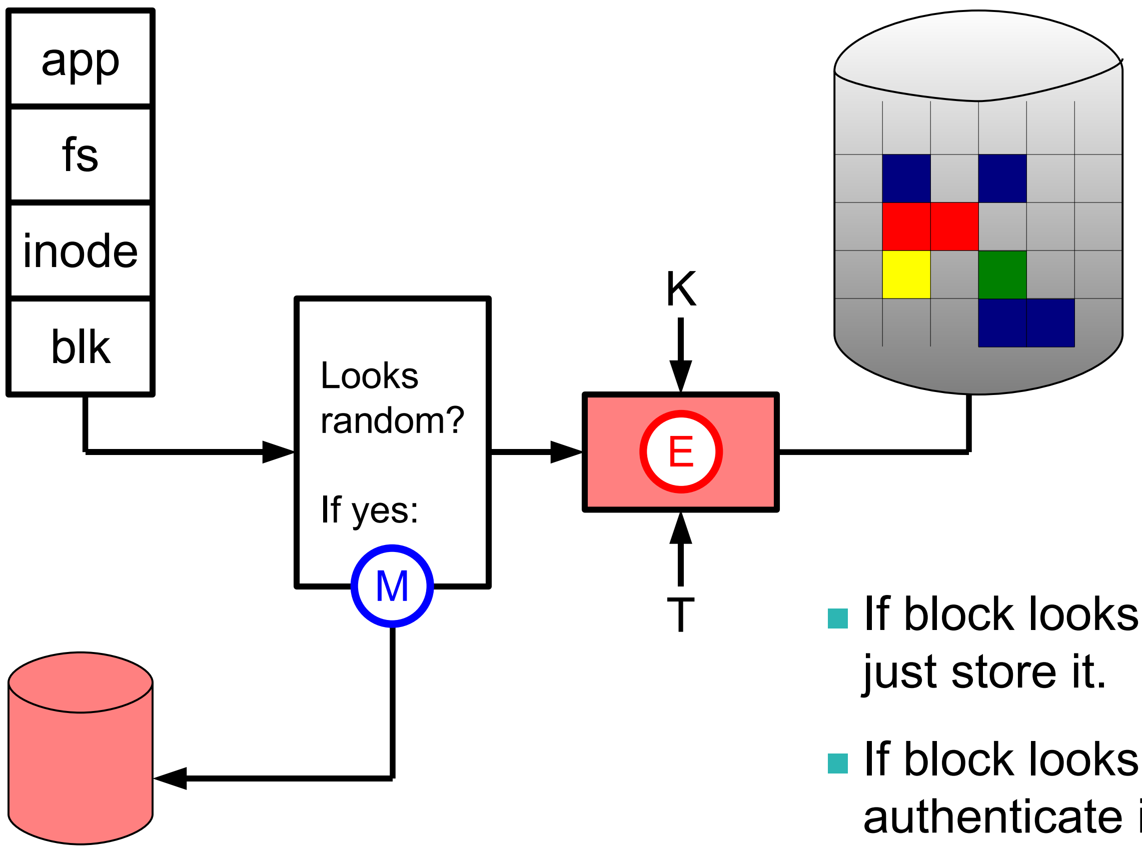
- Data is encrypted
- Use tweakable encryption mode on wide block (sector size, 512B)
- Idea

  If data contains redundancy, then any modification of ciphertext is detectable because decrypted plaintext will look random.

  → "Redundant" blocks are not extra protected for modification detection
  → "Random" blocks are protected in traditional way

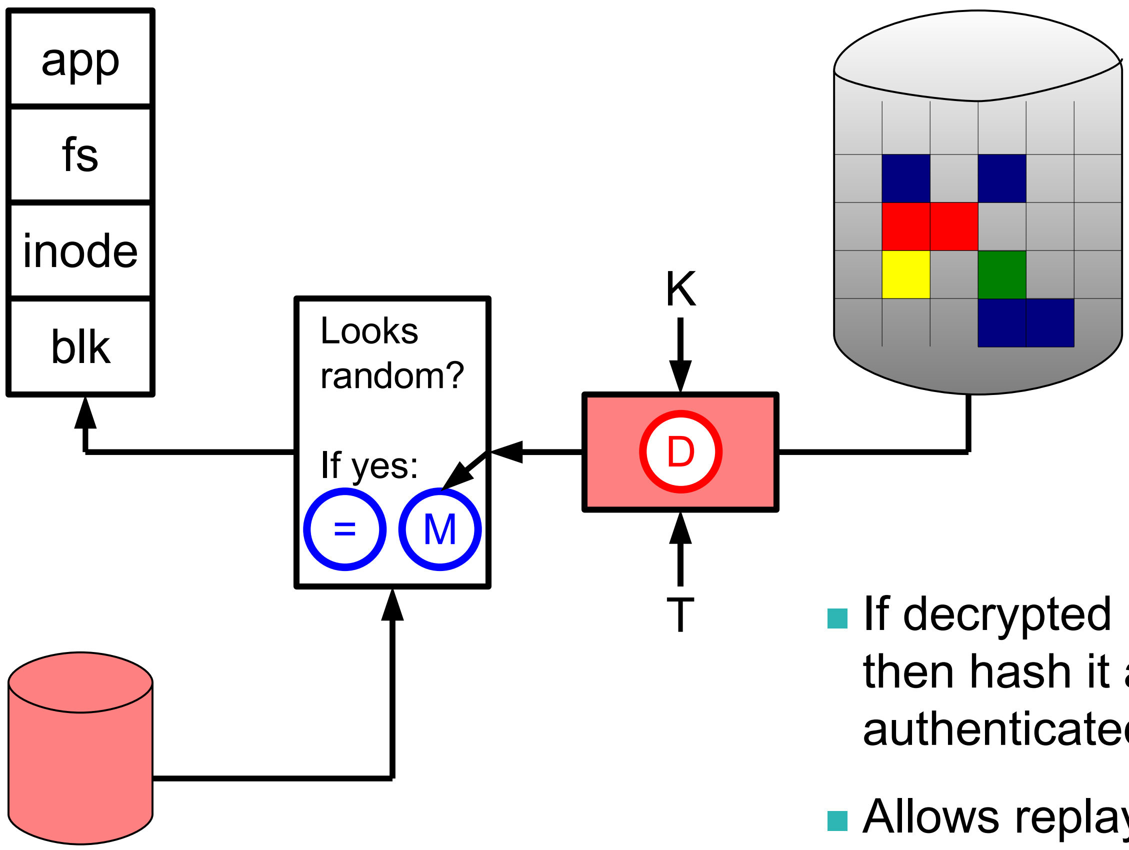
- Needs a heuristic test for "redundancy"

Writing Data



- If block looks redundant, just store it.
- If block looks random, authenticate it using extra trusted storage.

Reading Data



- If decrypted block looks random, then hash it and compare it with authenticated value.
- Allows replay attack with previous content of sector.

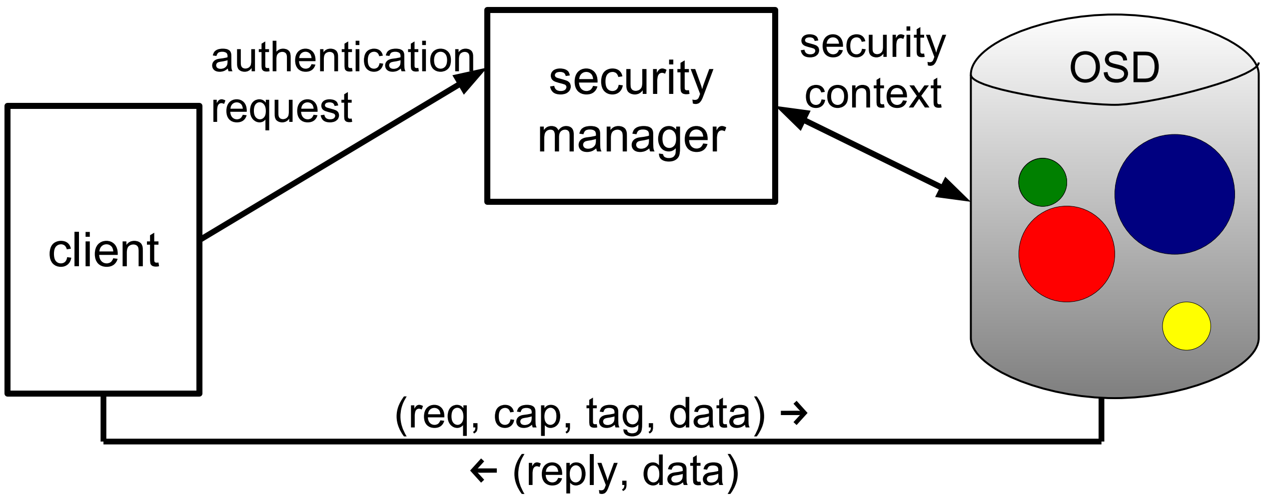
Discussion of Hybrid Scheme

- Performance depends on payload data
- Suffers from replay attacks
- Depends on estimator for redundancy
  → Simple 1-st order entropy test on 8-bit blocks in 1024-byte sector
    - Threshold set to 7.7 bits
    - 98% of blocks from file system trace have observed entropy < 7.7

  → Saves 98% storage space compared to hashing every block
  (Or: protects integrity of 98% of observed data.)
- Cannot achieve ideal security for arbitrary payload

Object Layer

Object Store Security Protocol [ACF+02]



- Capability-based protocol to authenticate requests and traffic between client and object-storage device (OSD)
- Key establishment protocol between OSD and security mgr.
- Protocol between client and security mgr. specific to file system

Protocol Features

- Security methods

  NONE: --

  CAPKEY: authenticate requests at OSD level, no transport security

  → tag computed only over cap

  CMDRSP: above plus transport integrity for request and reply

  → tag computed over capability and request

  ALLDATA: above plus transport integrity for payload data

  → tag computed over capability, request, and data

- May replace IPsec for iSCSI or FC-SEC for Fibre Channel (also duplicates some of their functionality)

OSD Data Types

- Object hierarchy

  OBS → Partition → Object

- Key hierarchy

  Master key: to initialize OSD and create root key

  Root key: to manage partitions and their keys

  Partition key: only to create per-partition working key

  Working key: per partition, changed frequently, useful for revocation (among other uses), protects all objects in partition

OBS Security Protocol Details (CAPKEY)

- PRF F
- Capabilities

  (obj, exptime, permissions, nonce)
- Client requests credential from security manager and receives

  cred = (cap, Kcap)

  where Kcap = F_K(cap) under appropriate partition working key K
- Client sends

  (req, cap, tag)

  to OSD, where

  tag = F_{Kcap}(cap | client | OSD)
- OSD verifies that

  1. req is allowed by cap in partition

  2. validates tag from its own id, using key K' = F_K(cap) with working key K of current partition

File Layer

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_PKA(FEK_X),
  group: G, r-w, E_PKG( 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

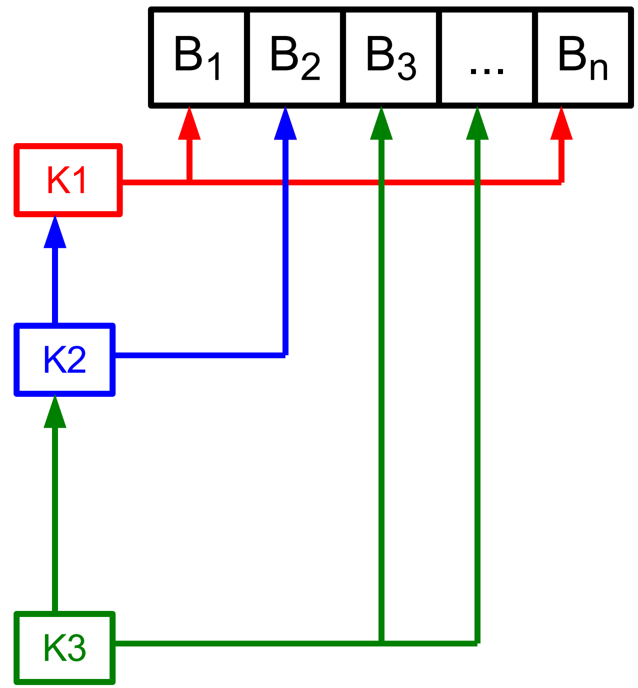
Key Revocation

- 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.

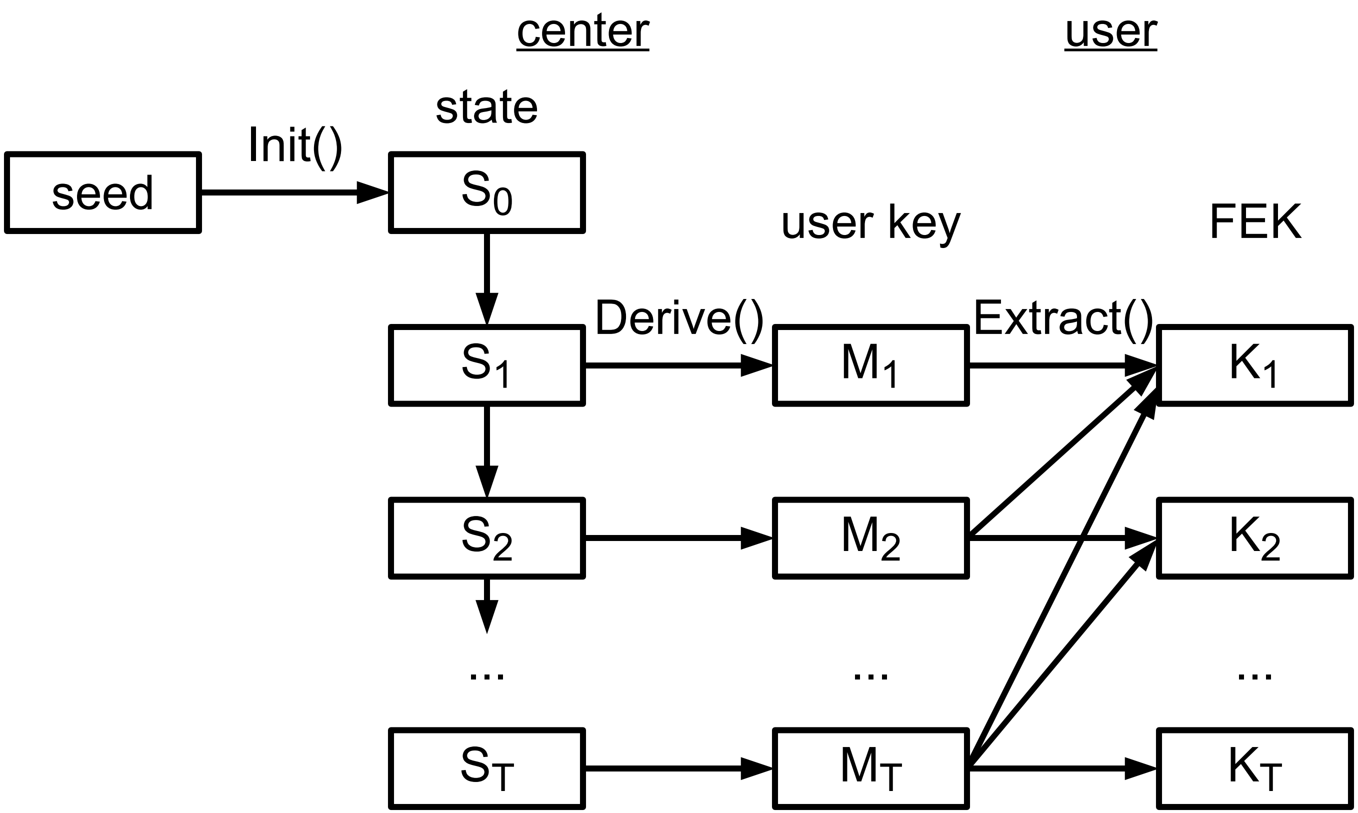
Lazy Revocation [KRS+03]

time	center	users: u	u'	u''
	S1(K1)	K1	K1	K1
u' is revoked				
	S1(K1,K2)	K2	--	K2
u'' writes B_2				
u'' is revoked				
	S1(K1,K2,K3)	K3	--	--
u writes B_3 ...				

storage



Key Updating Schemes for Lazy Revocation



- Requirements
  - → User can obtain K_1 ... K_t from M_t
  - → Adversary with M_t cannot distinguish K_{t+1} from uniformly random string

Formalization [BCO05, BCO06, FKK06]

- Key updating scheme for T periods

  KU_T = (Init, Update, Derive, Extract)

- Metrics of interest
  - → Time of Update(), Derive(), and Extract()
  - → Size of center state S_t
  - → Size of user key M_t

Composition of Key Updating Schemes [BCO06]

- Addition

  $KU^1_{T1} \oplus KU^2_{T2} \;=\; KU^{\oplus}_{T1+T2}$

  Construction

  → First T1 intervals use KU^1

  → Subsequent T2 intervals use KU^2 and include M_{T1} in user key

- Multiplication

  $KU^1_{T1} \otimes KU^1_{T2} \;=\; KU^{\otimes}_{T1 \cdot T2}$

  Construction

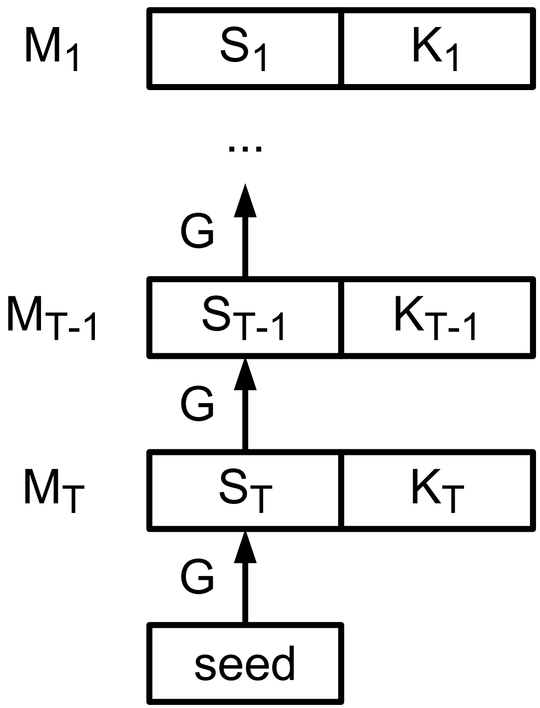
  → Every key generated with KU^1 is used to seed an instance of KU^2

Constructions

- Chaining construction
- Trapdoor permutation-based
- Tree construction

Chaining Construction ("Hash Chain")

- Using pseudo-random generator G

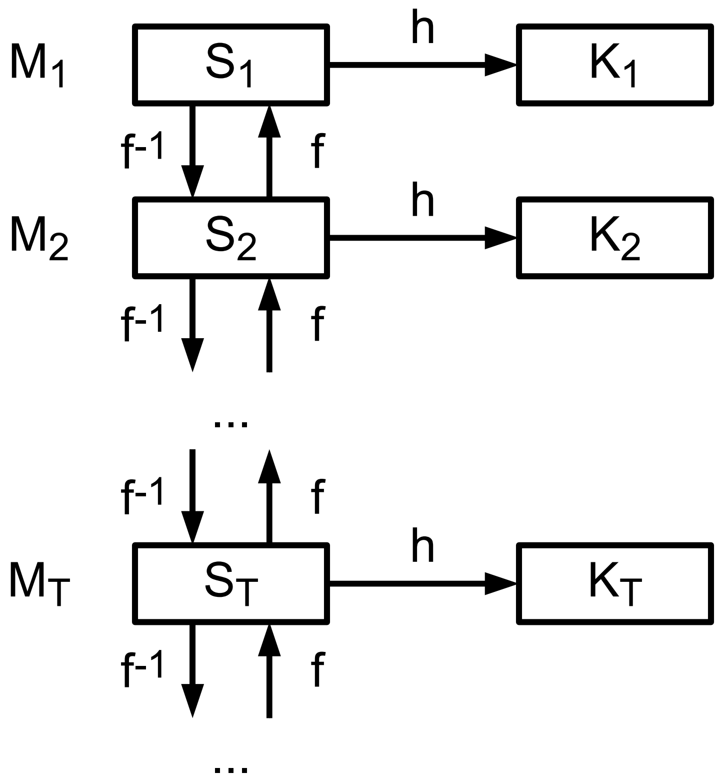


State	Update	Derive	Extract
seed	0	O(T) PRG	O(T) PRG

- Drawback: Fixed T

Trapdoor Permutation Construction [KRS+03]

- Using trap-door permutation TDP (f, f^{-1}), where f is easy and f^{-1} is hard without secret key, hash function h() in ROM

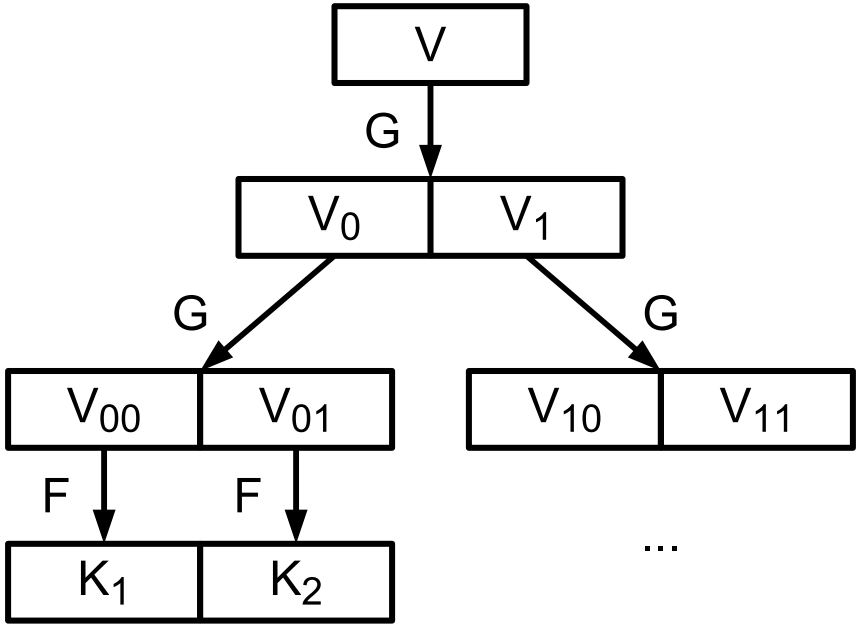


State	Update	Derive	Extract
seed	TDP	const.	O(T) TDP

Advantage: Flxible T

Tree Construction [BCO06]

- Using PRG G and PRF F



State	Update	Derive	Extract
O(log T)	O(log T) PRG	0	O(log T) PRG

- User key M_t is smallest set of nodes needed to derive K_1 ... K_t
- Fixed T, but state parameters only logarithmic in T

Comparison of Key Updating Schemes

- Trapdoor scheme using RSA-1024
- PRF/PRG using AES-128
- Average times [ms] measured on Intel 2.4 GHz Xeon

Scheme	T	Derive + Update	Extract
Chaining	1024	1.28	1.24
Trapdoor	1024	15.4	15.2
Tree	1024	0.015	0.006
Tree	2^{16}	0.015	0.008
Tree	2^{25}	0.015	0.01

Integrity Protection

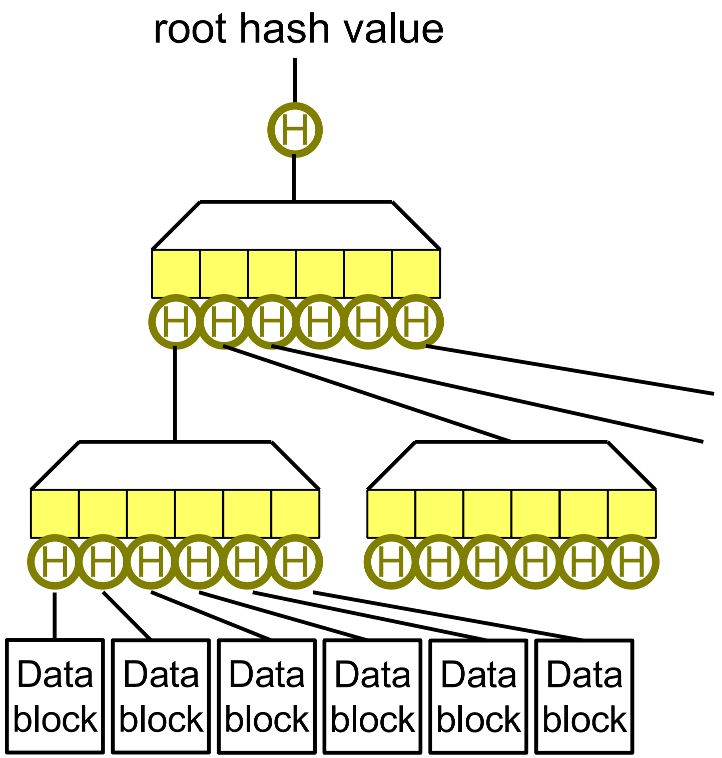
- Storage server not trusted
- Associate short reference value v with long file
  - → Store v on trusted server, with file meta-data
  - → Sign v with digital signature
- Hash function?

  v = H(file)

  - → Infeasible for long files
  - → No random access
- Solution:
  - → Hash tree [Merkle]

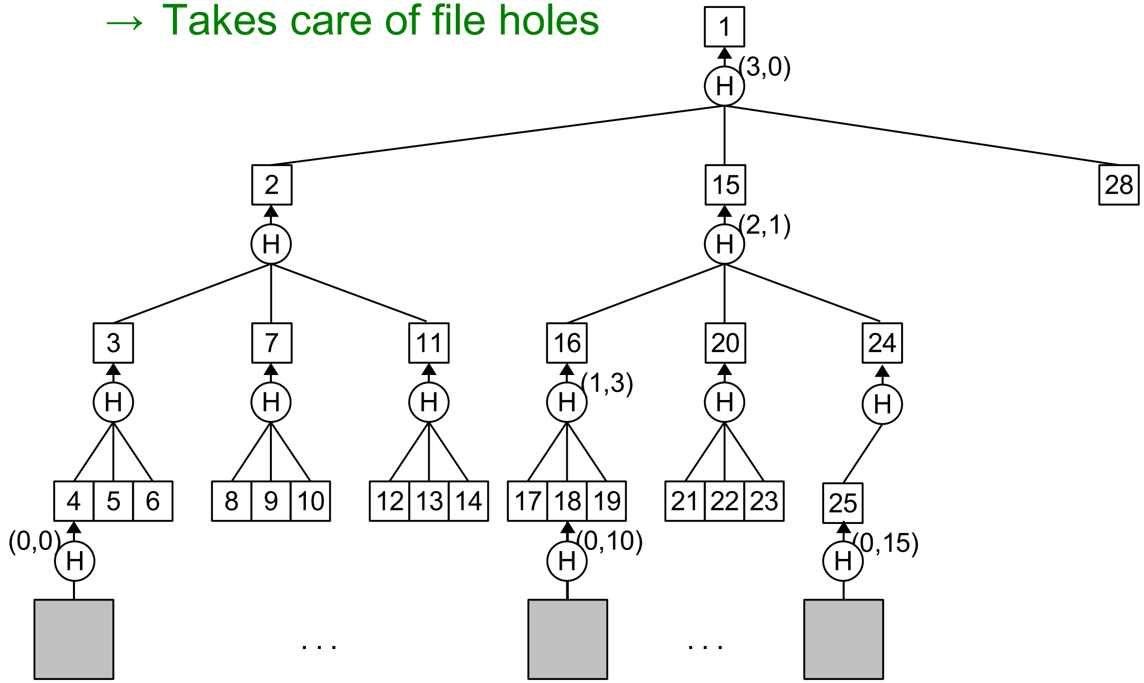
Integrity Protection Using Hash Trees

- 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

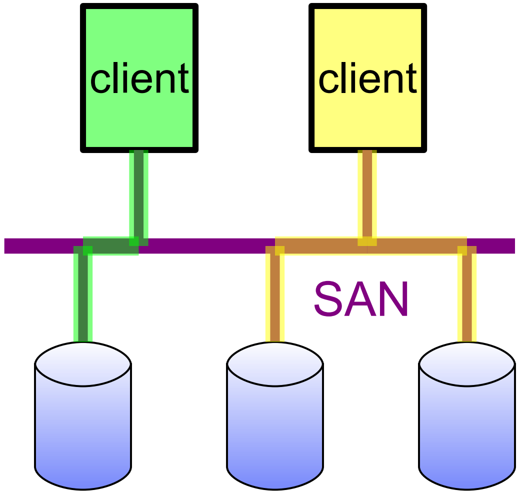
- Much more complex than encryption in file system
  - → Dual and mutually dependent data paths
- Degree may vary (2 ... 128), determine experimentally (≈16)
- Serialize nodes using pre-order enumeration
  - → Sparse allocation of maximum-size tree
  - → Takes care of file holes



Example: A Cryptographic SAN File System [PC06]

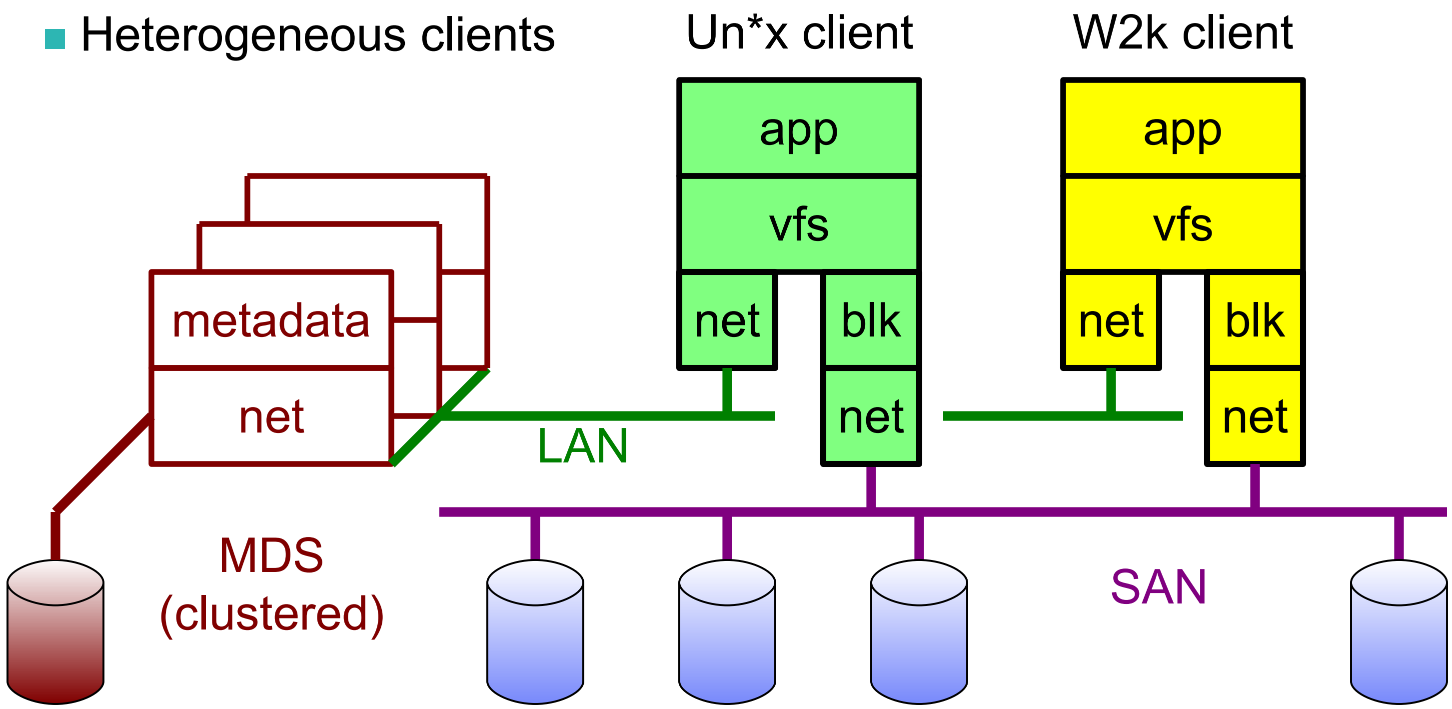
SANs and SAN File Systems

- SAN today:
  - Clients access block storage devices directly
  - → Fibre Channel (SCSI)
  - Static configuration
  - → OS sees a local block storage device
  - Static access control
  - → zoning & fencing in FC switch
  - Inside server room only



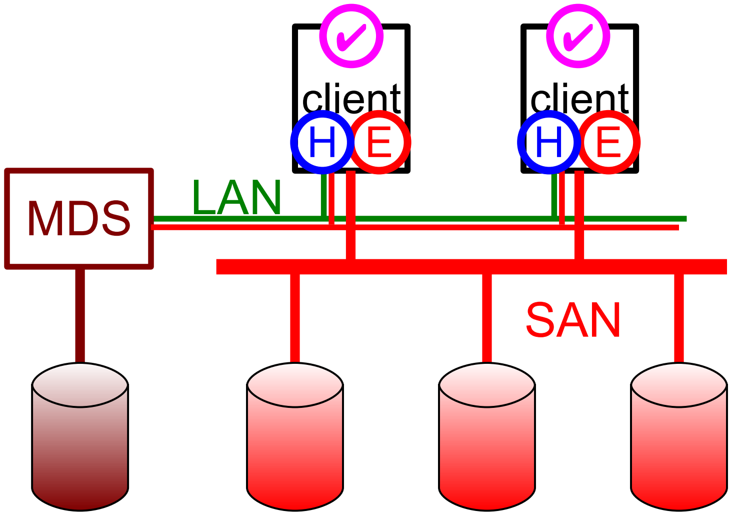
SAN Filesystems (e.g. IBM's StorageTank)

- Virtualized block storage space
- Block access managed by metadata server (MDS)
- Single filesystem name space
- Heterogeneous clients



Design of a Cryptographic SAN Filesystem

- Integrity verification & encryption in client
  - → Scalable
  - → End-to-end security
- MDS is trusted, provides encryption keys & reference data
  - → Integrate key management with metadata
  - → No modification of storage interface
- Needs
  - secure LAN connection (IPsec)
  - trusted client kernels

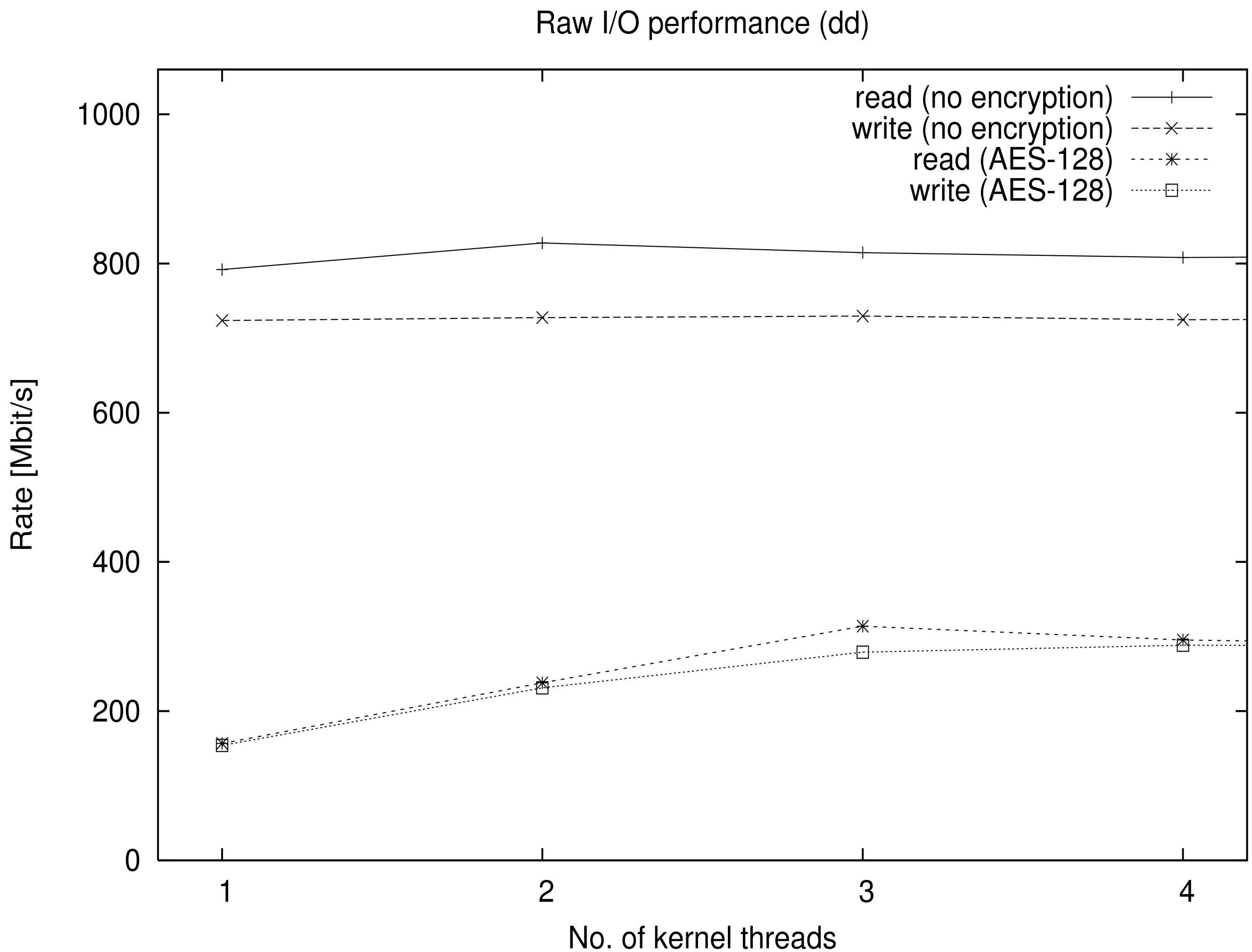


✓ Access control

H Integrity protection

E Encryption

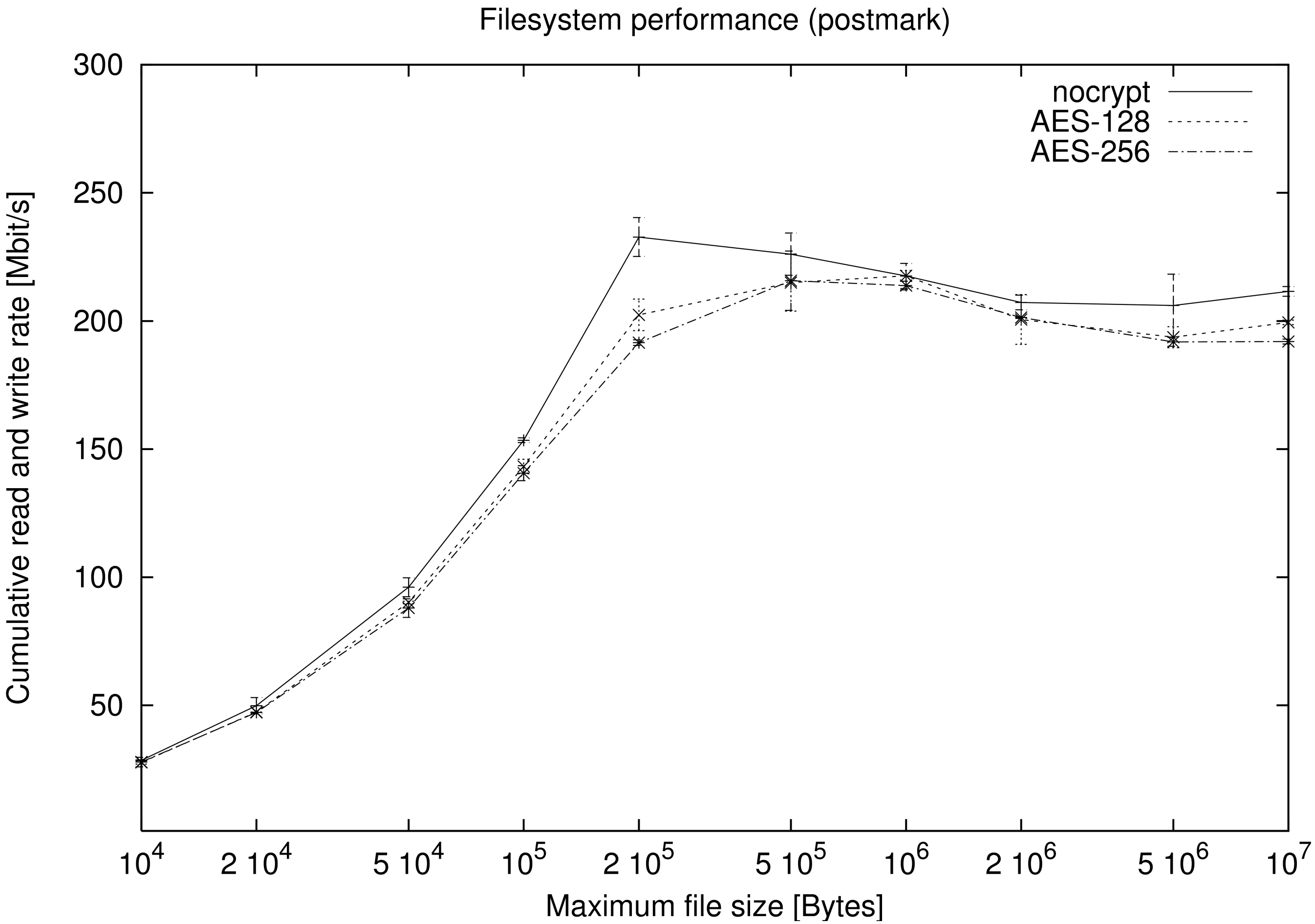
Raw Peak Performance with Encryption (dd)



- SAN.FS client 8.9 on
  - IBM x336
  - dual Xeon 3.2GHz (4 CPUs in Linux)
  - 3 GB RAM
  - Linux 2.6.6
- iSCSI 4.0.1.7 over
  - 1Gbit/s Ethernet
- Storage target on
  - IBM x346
  - dual Xeon 3.6GHz

Kernel threads used by STFS pager

Application Performance with Encryption (postmark)

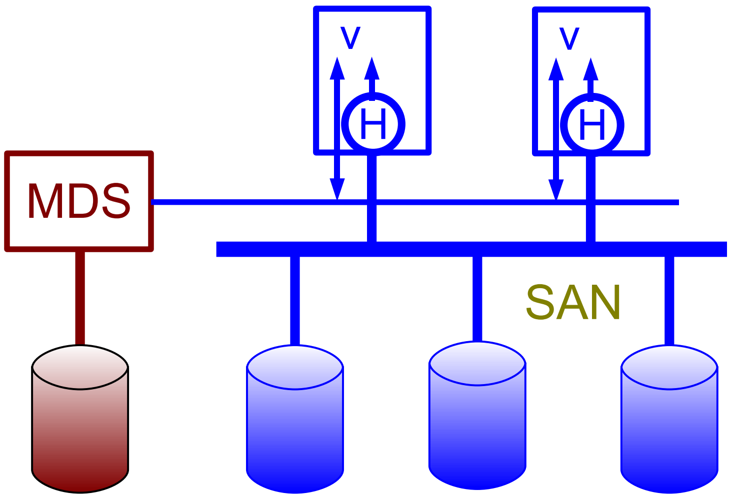


- SAN.FS client 8.9 on
  - IBM x336
  - dual Xeon 3.2GHz (4 CPUs in Linux)
  - 3 GB RAM
  - Linux 2.6.6
- iSCSI 4.0.1.7 over
  - 1Gbit/s Ethernet
- Storage target on
  - IBM x346
  - dual Xeon 3.6GHz

postmark performed 5000 transactions on 2000 randomly sized files between 1k and max. file size

Integrity Protection

- Data is hashed on client to digest values
  - → Digest values stored at MDS
  - → Secure transfer of digests
  - → Integrity protected in flight *and* at rest, modifications are detected
- Storage interface unmodified
  - → Impossible to prevent overwrites, but violations are detected
- SHA-1, SHA-256 or others
  - → NIST standards, fast & secure
  - → ~ 260 MByte/s in software (Xeon 3GHz)
- Using hash tree



Summary

- Any security mechanism can be applied on all layers
- Challenge is to select the “right” combination

	E	M	✓
file	key mgmt. & lazy revocation	hash trees	
object			OBS security protocol
block	tweakable block encryption	hybrid block-integrity protection	

Thank you!

- More information?

  http://www.zurich.ibm.com/~cca

  <cca@zurich.ibm.com>

References (1)

- [ACF+02] Alain Azagury, Ran Canetti, Michael Factor, Shai Halevi, Ealan Henis, Dalit Naor, Noam Rinetzky, Ohad Rodeh, and Julian Satran. A two layered approach for securing an object store network. In Proc. 1st International IEEE Security in Storage Workshop (SISW 2002), 2002.
- [BCO05] Michael Backes, Christian Cachin, and Alina Oprea. Lazy revocation in cryptographic file systems. In Proc. 3rd Intl. IEEE Security in Storage Workshop, pages 1-11, December 2005.
- [BCO06] Michael Backes, Christian Cachin, and Alina Oprea. Secure key-updating for lazy revocation. In D. Gollmann, J. Meier, and A. Sabelfeld, editors, Proc. 11th European Symposium On Research In Computer Security (ESORICS), number 4189 in Lecture Notes in Computer Science, pages 327-346. Springer, 2006.
- [FKK06] Kevin Fu, Seny Kamaram, and Tadayoshi Kohno. Key regression: Enabling efficient key distribution for secure distributed storage. In Proc. Network and Distributed Systems Security Symposium (NDSS 2006), 2006.
- [FKM02] Kevin Fu, Michael Kaminsky, and David Mazières. Using SFS for a secure network file system. ;login: --- The Magazine of the USENIX Association, 27(6), December 2002.

References (2)

- [Fu99] Kevin Fu. Group sharing and random access in cryptographic storage file systems. Master Thesis, MIT LCS, 1999.
- [GSMB03] Eu-Jin Goh, Hovav Shacham, Nagendra Modadugu, and Dan Boneh. SiRiUS: Securing remote untrusted storage. In Proc. Network and Distributed Systems Security Symposium (NDSS 2003), 2003.[HR04] Shai Halevi and Phillip Rogaway. A parallelizable enciphering mode. In Tatsuaki Okamoto, editor, Topics in Cryptology --- CT-RSA 2004, volume 2964 of Lecture Notes in Computer Science, pages 292-304. Springer, 2004.
- [KRS+03] Mahesh Kallahalla, Erik Riedel, Ram Swaminathan, Qian Wang, and Kevin Fu. Plutus: Scalable secure file sharing on untrusted storage. In Proc. 2nd USENIX Conference on File and Storage Technologies (FAST 2003), 2003.
- [LRW02] Moses Liskov, Ronald R. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, Advances in Cryptology: CRYPTO 2002, volume 2442 of Lecture Notes in Computer Science, pages 31-46. Springer, 2002.

References (3)

- [ORY05] Alina Oprea, Michael K. Reiter, and Ke Yang. Space-efficient block storage integrity. In Proc. Network and Distributed Systems Security Symposium (NDSS 2005), 2005.
- [PC06] Roman Pletka and Christian Cachin. Cryptographic security for a high-performance distributed file system. Research Report RZ 3661, IBM Research, September 2006.

Further reading

- [KK05] Vishal Kher and Yongdae Kim. Securing distributed storage: Challenges, techniques, and systems. In Proc. Workshop on Storage Security and Survivability (StorageSS), 2005.
- [RKS02] Erik Riedel, Mahesh Kallahalla, and Ram Swaminathan. A framework for evaluating storage system security. In Proc. USENIX Conference on File and Storage Technologies (FAST 2002), 2002.
- [WDZ03] Charles P. Wright, Jay Dave, and Erez Zadok. Cryptographic file systems performance: What you don't know can hurt you. In Proc. 2nd International IEEE Security in Storage Workshop (SISW 2003), 2003.