On Protecting Integrity and Confidentiality of Cryptographic File System for Outsourced Storage

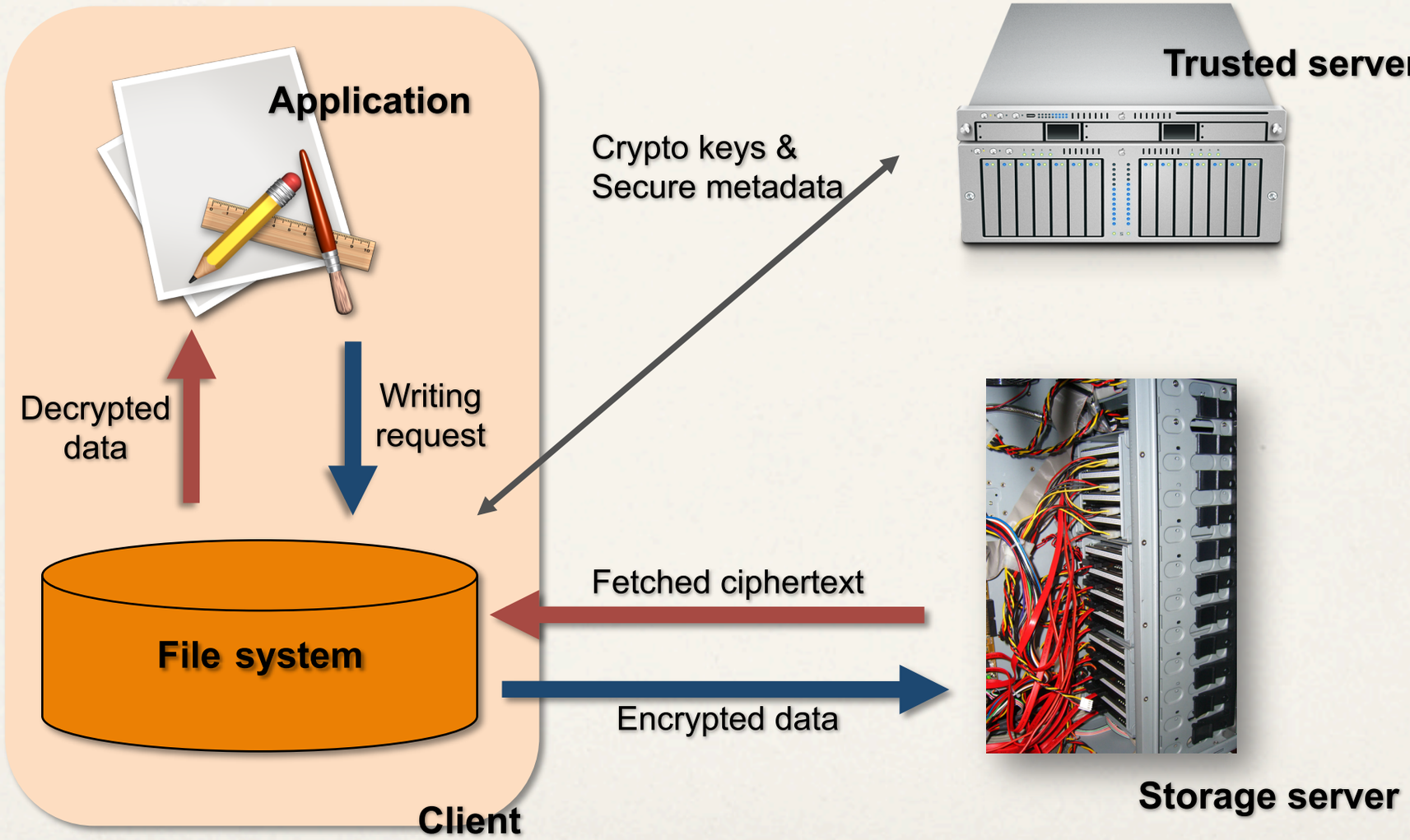
Aaram Yun, Chunhui Shi, Yongdae Kim
University of Minnesota

CCSW 2009, 13 Nov 2009

Cryptographic network file system

- How to achieve
- a network file system
- where data storage can be outsourced
- securely and efficiently?

Cryptographic network file system



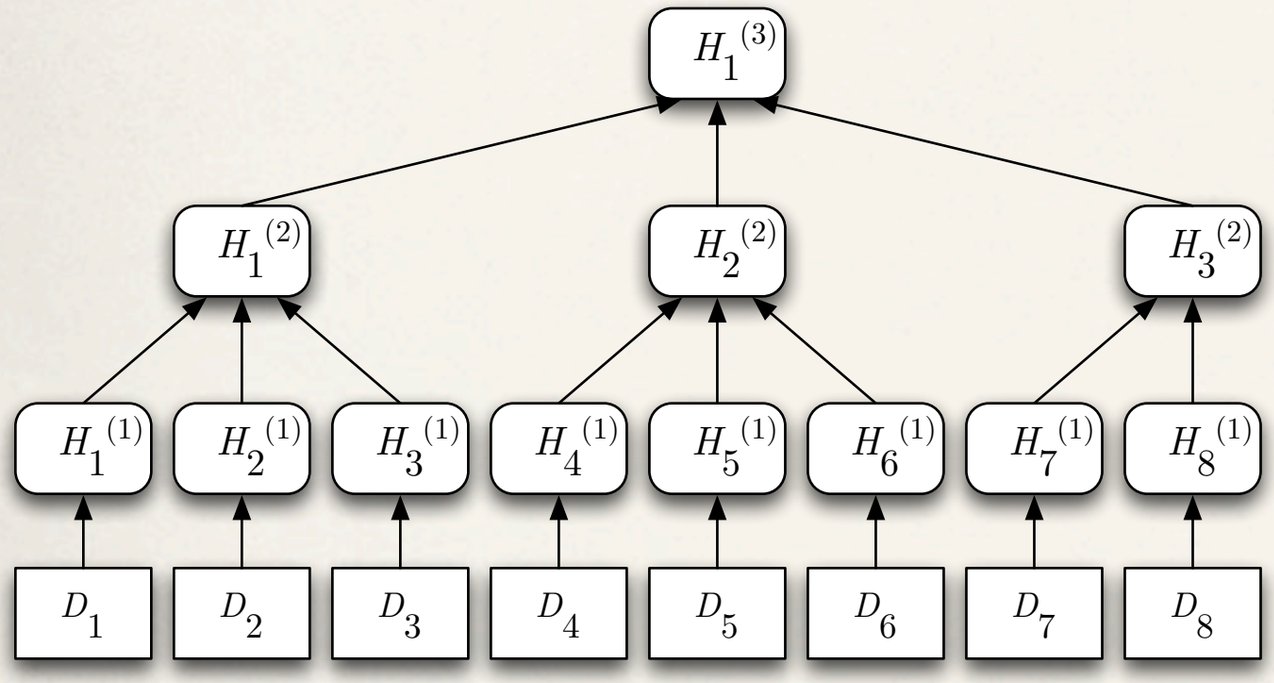
Goals

- Formal security definition for cryptographic file system
  - confidentiality & integrity against attacker which controls data storage
- Efficient construction
  - better computational overhead for crypto operations

Requirements

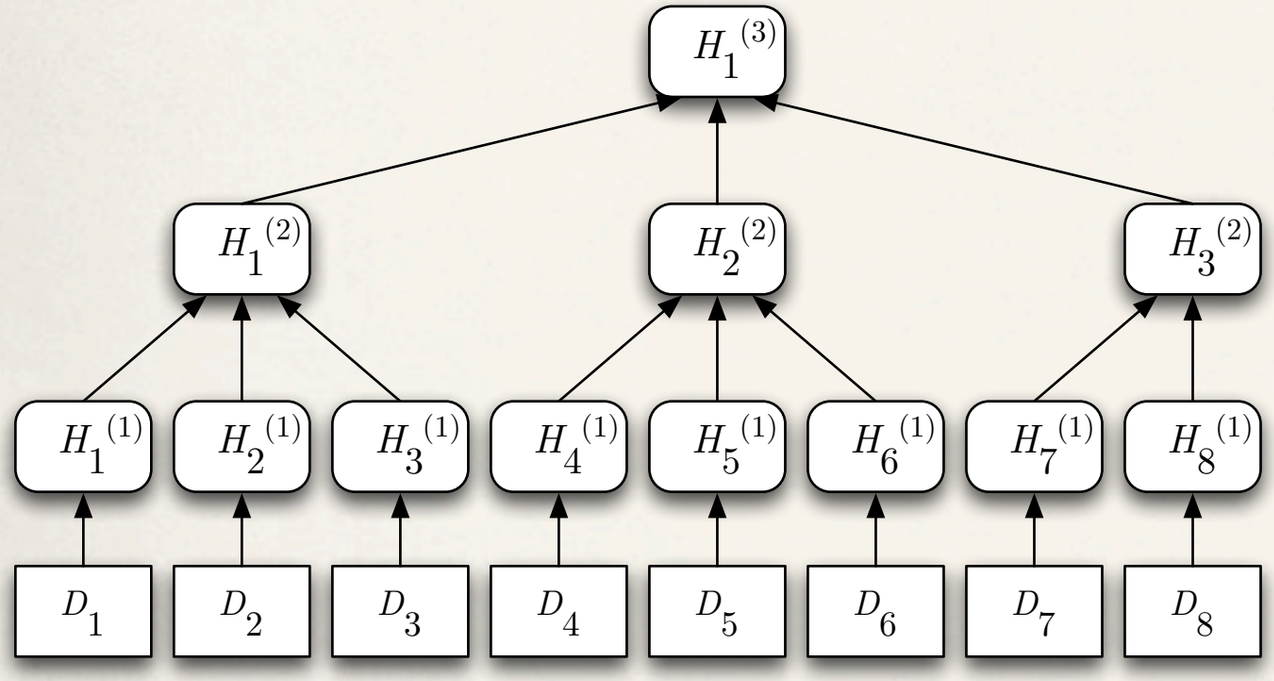
- Confidentiality & integrity of stored data
- Random access
- Only constant amount of trusted storage per file
- Small computational overhead

Merkle hash tree



- Popular solution for protecting data integrity
- Data blocks at leaf nodes
- Tree of hash values
- Root should be stored securely
- O(log n) cost for update

Merkle hash tree + encryption



- Put encrypted data blocks at leaf nodes
  - Blockwise encryption using CTR, for example
- Protects confidentiality and integrity

How to enhance Merkle tree?

- Efficiency
  - Hash function is fast, but not too fast
    - Speed of SHA-1 only about 1.5 times faster than AES-128, in most software environments
    - SHA-2 slower than AES-128 in general
- Security
  - Secure, but could leak information if not used carefully

Formalism

- A file represents a sequence of file blocks $D_1D_2...D_n$
  - Allowed operations (file encryption key is implicit)
    - Read(k), Length(), Update(k, D), Append(D), Delete()
- T: trusted storage, S: data storage
  - $(t, s) \in T \times S$: state of a file, starting from a fixed initial state, updated by file operations
  - Failed operation cannot change t, but it may change s

Security definitions

- Integrity: infeasibility of alteration of file content

- Attacker is allowed to interact with the file, making file operation queries

- Attacker can feed arbitrary state s′ before any file operation

- Attacker wins if he requests read(k) and obtain $D' \neq D_k$

  - D_k: k^{th} block of the correct file content

Security definitions

- Confidentiality
  - infeasibility to learn anything about a file block, other than by reading the block
    - Even when the attacker somehow coerces a valid user to read a block of plaintext or eavesdrops it, still unread blocks do not give any information

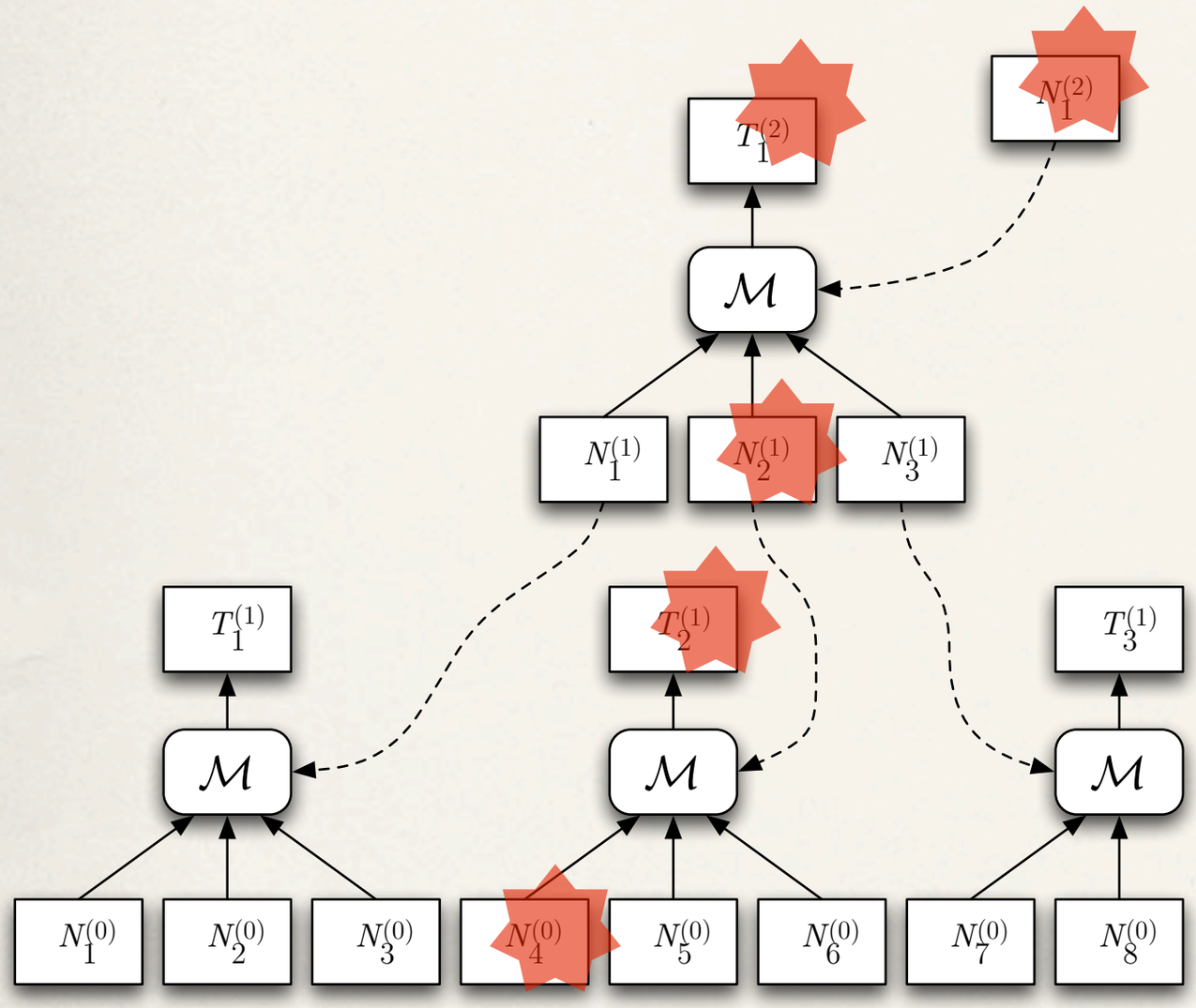
Universal hash-based MACs

- Universal hash function : Prob[$H_k(x)=H_k(y)$]<ε for any $x \neq y$
- Structure of $H_k(x)$ is very simple
- Long data block is ‘compressed’ by cheap universal hashing, then ‘encrypted’ by XORing to an enciphered nonce

  $\tau = M_{k,\,k'}(N, M) = H_k(M) \oplus E_{k'}(N)$

- Attacker cannot produce a forgery: (N, M, τ) satisfying $\tau = H_k(M) \oplus E_{k'}(N)$ with new (N, M)
- We use Poly1305-AES, but other UH-based MACs are also usable

Nonce-based MAC tree construction



- If nonce is untampered, validity of data & MAC can be checked
- Root nonce is securely stored
- Trust is transferred down the tree
- Leaf nonces are used to encrypt data blocks
- Needs only to protect nonces & nonces can be shorter than hashes!

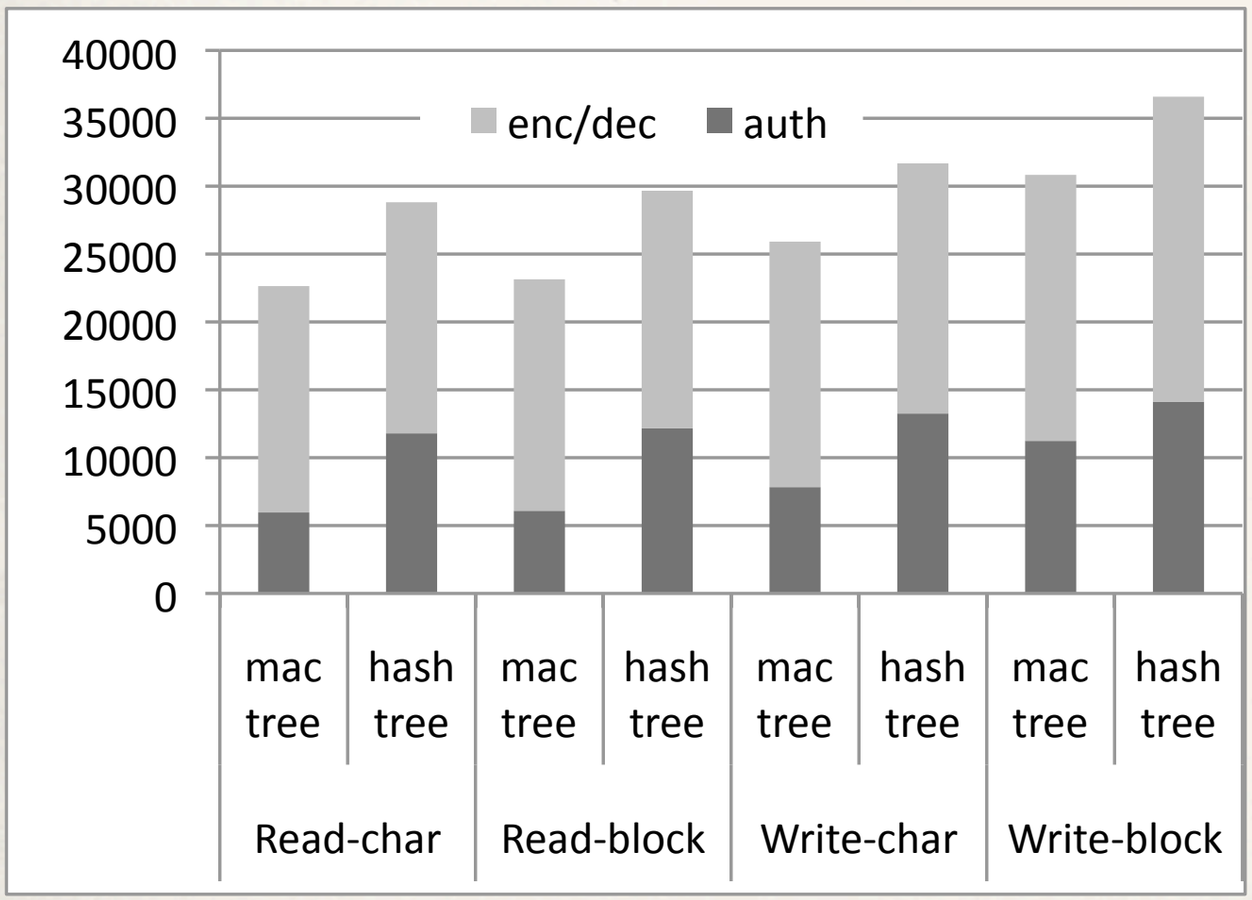
How to encrypt using nonces

- Nonces at the leaf nodes, $N_k^{(0)}$ are used for encrypting each file blocks in CTR mode, and also for authenticating file blocks

- If, $N_k^{(0)}$ are kept in a trusted storage & incremented properly whenever update of a block happens, this encryption & authentication can be proven to be secure

- But, since $N_k^{(0)}$ are protected by the MAC tree, still this is secure

Implementation & performance



- Implemented the file system on a FUSE based network file system
  - One for our MAC tree, one for Merkle hash tree
- Cost of authentication is about 50% of the Merkle tree construction in general

Thank You!