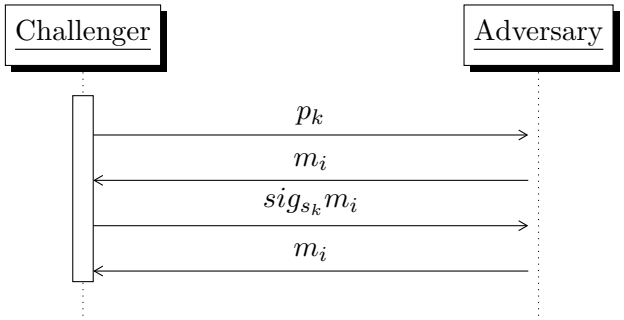
9/22 Sundr

Scribe: Michael Chen

September 27, 2015

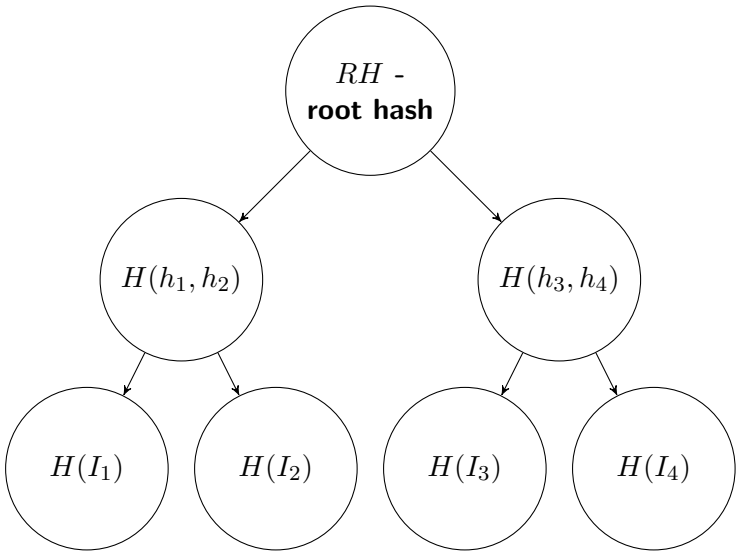
Crypto Refresher

- Digital Signatures
  - Functions
    - Keygen: $K(1^k) \rightarrow s_k, p_k$, where the input is the security parameter to get k bits
    - $Sign(s_k, m) \rightarrow sig_{s_k}(m)$
    - $Verify(m, p_k, signature) \rightarrow$ yes/no
  - Correctness: a correct signature will verify as yes: $\forall m, Verify(m, p_k, sig_{s_k}(m)) \rightarrow$ yes
  - Security: existentially unforgeable



    Adversary wins if he can return a message m_j such that $Verify(m_j, sig_{s_k}(m_i)) \rightarrow$ yes and m_j has not yet been seen ($m_i \neq m_j$)

- Collision resistant hash functions
  - $hash : \{0,1\}^* \rightarrow \{0,1\}^d$
  - Collision resistance: adversary can't find $x, x' \neq x$ such that $H(x) = H(x')$
- Merkle Hash Trees: authenticate a set of items



  - Clients only store root hash
  - Server sends path and siblings of path up to root hash to prove validity
  - Good properties
    * Proof is $O(\log n)$
    * Client only needs to store 1 hash (root hash)
    * Given that the hash functions are collision resistant, attacker can't provide an alternative answer

Sundr Basics

- Ensure data integrity with compromised server
- Threat model: attacker can fully compromise server
- Fetch-modify consistency
  - Fetch reflects the latest modification by an authorized user
  - One total order between fetch and modify
- Fork Consistency
  - If clients meet, they can find out that they were forked
  - If Alice sees an update from Bob, Alice sees all previous updates from Bob
- Setup
  - Each user has s_k, p_k
  - Assume each user knows public key of another user
  - Each client know access control list for all files
  - Small client storage

Strawman 1: sign each file

- + client checks modification by authorized user
- - data can be old

Strawman 2: each client stores root hash of Merkle hash tree

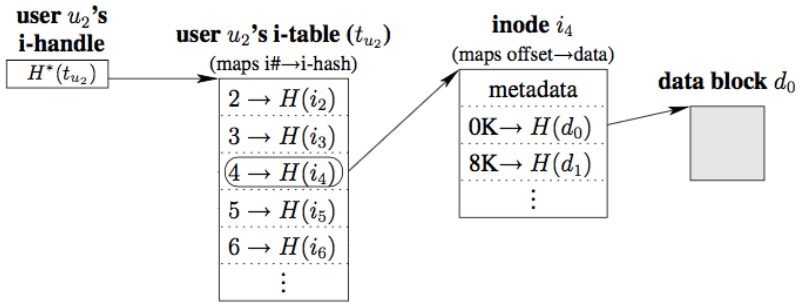
- + works for one client
- - does not work for many clients

Strawman 3

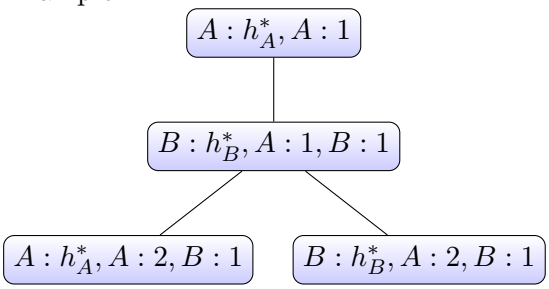
- Global lock on entire filesystem
- Server keeps entire history of operations for all files
- User signs each operation
- Client operation
  1. Acquire lock
  2. Download History
  3. Check signature
  4. Recreate filesystem
- Fetches remain in history so that we can notice forks: when Alice sees Bob's update, she will see all of his previous updates and whether her updates and fetches were in Bob's history
- Negatives: expensive
  - High bandwidth
  - No concurrency
  - High storage requirements

Sundr, without concurrency

- Inode stores list of hashes of blocks
- Id of block authenticates block's content
- User has I-table, which represents
- User's i-handle: $H^*(i-table) \rightarrow i\# \rightarrow$ hash of inode
- User's version structure



- keeps track of version numbers for each user, signed by user
- Each client stores own version structures list (VSL)
- VSLs are incompatible if there are some version numbers that are greater than another user and some version numbers that are less
  - Example:



    This is an incompatible VSL state because neither A nor B has a VSL that is a prefix of the other.
- No fork $\leftrightarrow \forall u1, u2$: history of one user is a prefix of the other

Client fetch/modify

- Acquire global lock
- Get VSL of all users
- Check to make sure VSLs are compatible
- Update user local VSL to reflect new version
- Fetch data based on Merkle root hash from person with last version (this works because VSLs contain their respective Merkle root hashes)
- Put new VSL at server
- Takes $O(U^2) << O(\#operations)$, where U is the number of users and $\#operations$ is the number of operations in the history of the file system

Concurrent implementation: allow concurrent requests, but changes to VSL are still serial.