Secure Untrusted Data Repository (SUNDR)
Li, Krohn, Mazieres, and Shasha
OSDI 2004

goal:
  store your files on a server
  usually we "trust" the file server
  but what if it's outsourced over the internet
    maybe owning company doesn't apply Microsoft patches
  server may be completely corrupt
  users can verify that they got the correct data back
  so "secure" is about authentication, not privacy

can we design our own?
  i sign what i write to the server
    put(filename, signed contents)
    get(filename) -> signed contents
  users know each others' public keys
  check signatures when you retrieve
  now the server cannot forge data!
  are we done?
  this might be OK for a single user

what might go wrong?
  server shows me stale data: signed != fresh
    do i have to keep track of latest write TS for every file i own?
  server shows me stale data from other users
    i can't even know when other users wrote
  server omits or re-orders operations
  in general: server shows different users different contents
  what about directory contents if many users create files?
    or file contents if many users write parts of the same file?
    how can anyone sign the data?

idea:
  ordinary FS protocol:
    operations to server
    server replies with contents
    you can secure the channel
    you have to trust the server to interpret the operations
  SUNDR approach:
    signed operations to server
    server replies with the same signed operations!
    the clients interpret the operations
    the server only stores operations

the paper's straw man
  [explain log scheme]
  client asks server for a copy of the current log
    validates log (is my latest operation there? does last signature
check?)
    appends new operation, signs over the whole new log
    writes log back to server
    server must implement a lock
  it's a log of operations, not really of contents
    so clients have to interpret (play) the log
  prevents server from showing me my own stale data

```
  prevents server from changing the order of operations
    since each operation signs its place in the log

what *can* a malicious server do in straw man?
  it doesn't have to show me anything after my last operation
    i.e. it can conceal other users' recent operations

suppose server conceals U2's last operation from U1
  now U1 appends an operation to the log it got from the server
  can the server ever show U1's operation to U2?
  can the server ever show U1 any more of U2's operations?
  so: from now on the server can only show U1 its own new operations
    and can only show U2 its new operations
  this is a forking attack

fork consistency:
  only attack is a fork attack: conceal operations
  all users see the same log before the first concealed op
  no user sees another user's ops after the first concealed op

why is fork consistency good enough?
  the server *can* perform a forking attack!
  it's good that it's a violent attack
    after a while it will be obvious that we've been attacked
    easy to detect if users compare notes
  much better than allowing a concealed op, but showing subsequent ops

why is the straw man not enough?
  need to xfer the whole log to check signatures
  and you need to play/interpret the log entries
  caching optimizations possible, but expensive if you are away for a
weekend

can we get rid of log, just keep current directory tree?
  each i-node/directory block contains crypto hashes of children?
  i tell server just blocks changed by my operation?
  then i sign the root block?
  root block contains content-hash of previous root block?

why is a signed directory tree not quite right?
  how can i check that server didn't drop one of my operations?
    would need a log of root blocks?
  how can U1 prevent U2 from writing U1's files/directories?
    e.g. /u owned by root, /u/rtm owned by rtm

first, a user should sign only its own files/directories
  so we can't have a directory tree representation
    which requires change all the way up to the root for any file
change
  i-handle points to current i-table
  i-table maps i-number to hash of i-node
  directory block maps name to user#/i-number
    thus my directories can hold your files
    i-number lets you modify those files w/o changing my directory

now the file system is the collection of users' i-handles
  we really have a sequence of new i-handles
```

```
  arrange as a time-line per user

second, use version vectors to verify that operations are uniquely
ordered
  each user numbers successive i-handles, puts vers number into i-
handle
  each i-handle also contains versions of all users i-handles at time
of op

how do version vectors evolve in correct operation?
  U1: 1,0           2,2
  U2:      1,1  1,2     2,3
  validation:
    server shows me my my recent i-handle
    i-handle version vectors can be totally ordered
    i.e. 2,2 < 2,3

what would version vectors look like if server hid an i-handle update?
  U1: 1,0       [2,1]
  U2:      1,1        1,2
  server hides U1:2

do the version vectors give us fork consistency?
  can the server show future U1 i-handles to U2?
    e.g. 3,1
    no: 3,1 and 1,2 cannot be ordered!
  can the server show future U2 i-handles to U1?
    e.g. 1,3
    no: 1,3 and 2,1 cannot be ordered
  can the server show 2,1 to U2 at a later time?
    i.e. after U2 finishes 1,2, when it's about to start 1,3?
    no: 2,1 and 1,2 cannot be ordered

why aren't we done at this point?
  the server still needs to serialize operations
  server will be idle between sending current VVs, and waiting for new
one
    i.e. between UPDATE and matching COMMIT
  we cannot just allow concurrent operations:
    not orderable, would look like an attack
```