spin: the personal cloud

Nicholas C. Landolfi

Revision: b7f91fe, December 2022

spin is...

- a remote file system
- a personal cloud
- a digital data store as a utility
- a device-independent, globally available file store
- a global namespace

spin consists of

- a `KeyServer`, holding the access credentials of citizens
- one or more `DirServers`, holding directory trees for citizens
- one or more `BitServers`, holding file blocks for citizens
- together these give an API resembling an access-controlled file system
  - with caching, this file system can be over the network, but (relatively) fast
  - may be slow on first read, speeds up with usual memory access patterns

citizens and keys

- a *citizen* is a user, member, customer; identified by a *name* (unicode string)
- a citizen has *keys*, these records are managed by a `KeyServer`; a key has the following fields
  - *type* (unicode string); e.g., `unknown`, `password`
  - *name* (unicode string); e.g., `lando`, `lando_extra_key`
  - *citizen* (unicode string); e.g., `lando`, `mike`, `cgb`
  - *data* (unicode string); e.g., base64 encoded password hash
  - *meta* (unicode string); e.g., salt
  - *created at* (timestamp); e.g., 12/27/22 12:00 PM
  - *expires at* (timestamp); e.g., 12/28/22 12:00 PM

`KeyServer` API

- the `KeyServer` maintains a *directory* of keys
- it can be accessed by clients via two methods
  - `Which` — lookup a key
    - input: public, private (unicode strings)
    - output: a key, with the data and meta fields empty
  - `Temp` — create a temporary session
    - input: public, private (unicode strings), duration (as string, e.g. 2m5s, 48h)
    - output: a key (with the data and meta fields empty), the private password (unicode string)

directory trees

- a user has a hierarchical *directory tree*; like a file tree in Unix; e.g.,
  - / (user directory root, always a dir)
    - `dir1` (a directory)
      - `hello.txt` (a file)
      - `world.csv` (a file)
    - `data.txt` (a file)
    - `cal.csv` (a *directory*)
      - `cal.csv` (a file)
      - `cal.csv.ops` (a file)
- *paths* identify files and directories; e.g., `/dir1`, `/cal.csv/cal.csv`
  - a `DirServer` manages this tree

directory entries

- a path is said to *exist* if there is a *directory entry* associated with it; entries have fields
  - *citizen* (unicode string); e.g., `lando`, `cgb`
  - *path* (unicode string); e.g., `/dir1`, `/cal.csv/cal.csv.ops`
  - *type* (unicode string); e.g., `file`, `dir`
  - *blocks* (list of `DirBlock`)
- a file entry is composed of *directory blocks* which have fields
  - *addr* (unicode string); e.g., `https://bit.spinsrv.com`
  - *ref* (unicode string); e.g. `743777dfda479b4c34a3221c57a864715c4811e6d002f11232dad898e2d8123f`
  - *offset* (int64); e.g., 20,971,520
  - *size* (int64); e.g., 10,485,760

`DirServer` API

- a `DirServer` maintains a tree of `DirEntry`
- it can be accessed by clients via three methods
  - `Apply` — serially apply a list of directory operations atomically
    - input: public, private (unicode strings); list of ops, which have *operation type* and dir entry
    - output: list of entries, an error
  - `Tree` — retrieve all directory entries underneath a path to some level
    - input: public, private, citizen, path (unicode strings), level (int)
    - output: list of dir entries, an error
  - `Watch` — retrieve all changes underneath a path
    - input: public, private, citizen, path (unicode strings), sequence (int, w/ special values)
    - output: initial response, then a stream of *dir events*

`BitServer` API

- a `BitServer` maintains a store of *named* byte sequences (also called blobs, chunks)
  - the name of a sequence is called its *ref*
- it can be accessed by clients via one method
  - `Apply` — batch apply (not atomic) a sequence of *bit ops*; have fields
    - *operation type*; one of "put","get", "del"
    - *ref*; unicode string
    - *bytes*; byte array, only for when type is put