One-way indexing for plausible deniability in censorship resistant storage

Eugene Vasserman, Victor Heorhiadi, Nicholas Hopper, and Yongdae Kim
Censorship resistant storage

- Provides robust permanent storage
- Protects against targeted blocking
  - Resists rubber-hose cryptanalysis – provides publisher deniability
- Easily searchable (e.g., not hashes)
- Removes “dead data”
  - Without necessarily killing unpopular content
- Scales gracefully
I am not a lawyer...

- Is plausible deniability needed?
- Is plausible deniability enough?
- Is “probable ignorance” enough?
“Conflicting” requirements

- Storer plausible deniability
- Keyword search
  - Decryption key must be stored in the network
  - Pointer and storer must not discover the key
- Self-contained network
  - Store keys and content in the same network?
    Are you crazy?!
- “One-way indexing”
DHT P2P storage refresher

ANIMATION

Publisher

Searcher

Pointer (Storer)
Encoding a file

Publisher has

File F

Encrypt with key K

$m$-of-$n$ erasure coding

$E_K(F)$

$n$ chunks

($n \gg m$)
Publishing files

Publisher composes “manifests”:

- **File Manifest**:
  - $h(E_K(F))$
  - $h(\text{keyword}_1), h(\text{keyword}_2), \ldots$
  - $h(\text{index keyword}_i)$
  - $h(F)$
  - $h(c_1), h(c_2), \ldots, h(c_n)$

- **Key Manifest**:
  - $h(E_K(F))$
  - $h(\text{keyword}_1), h(\text{keyword}_2), \ldots$
  - $h(\text{index keyword}_i)$
  - $h(K)$
  - $K$
“One-way” publishing

Publish file manifest to $h(r, \text{keyword}_1), h(r, \text{keyword}_2), \ldots$

Publish key manifest to $h(r', \text{keyword}_1), h(r', \text{keyword}_2), \ldots$
Finding a file

Search for file
Search for file manifest by keyword
Search for key manifest by keyword
Decrypt file, verify against manifest (hash)
Reconstruct file, verify against manifest (hash)
Retrieve random file chunks
Beware of forbidden keywords

- \( h(\text{keyword}_1) \rightarrow \text{salt}, h(\text{salt}, \text{keyword}_1) \)
- Brute-force hash search protection (rainbow tables)
- Robustness improvement (load balancing)
- Different salts in different manifests
- “Forbidden keyword” attacks tend to fail
Continuous robustness

- Pointer storer → manifest “guarantor”

- Guarantor can:
  - Reassemble the encrypted file
  - Check replication level of manifest and file
  - Re-encode the encrypted file (like publisher)

- Guarantor cannot:
  - Decrypt the file (get the plaintext)
  - Obtain the keywords (invert a hash)
  - Remove data from the network (can drop own data)
Maintaining/refreshing a file

- Retrieve x' > 2m manifest replicas, verify them
- Retrieve x ≥ m random chunks
- Reconstruct data, verify against manifest (hash)
- Re-publish data chunks and/or manifest if needed
- Manifest guarantor
Dead data pruning

- Each stored item has a timestamp
  - File manifest, key manifest, content chunk
- Timestamp initialized at publication time, refreshed at access time
- Nodes lazily garbage-collect “idle” items
  - Have not been accessed in some time period $t$
  - A single honest guarantor is enough to retain a file in the network
- Manifests “vouched for” by editors are not subject to dropping
System robustness

Censorship vs. Success

Node failure

1-of-10 (replication)
10-of-100
50-of-250
50-of-500
50-of-750
75-of-750
Performance

Time to perform DHT operations

“User time” to find and download a file

[Graph showing CDF of time for various DHT operations and “User time”]
Summary

Toward robust censorship-resistant permanent storage:

- “One-way” indexing and easy search
- “Probable ignorance” for storers
- Replication and proactive maintenance – targeted are attacks difficult
- Need underlying blocking resistance
- Dead data removal and file curation
- Keeps all files for a time, some forever