# One-way indexing for plausible deniability in censorship resistant storage

Eugene Y. Vasserman
*Kansas State University**

Victor Heorhiadi
*University of North Carolina at Chapel Hill**

Nicholas Hopper
*University of Minnesota*

Yongdae Kim
*University of Minnesota*

## Abstract

The fundamental requirement for censorship resistance is content discoverability — it should be easy for users to find and access documents, but not to discover what they store locally, to preserve plausible deniability. We describe a design for "one-way indexing" to provide plausibly-deniable content search and storage in a censorship resistant network without requiring out-of-band communication, making a file store searchable and yet self-contained. Our design supports publisher-independent replication, content-oblivious replica maintenance, and automated garbage collection.

## 1 Introduction

Censorship resistant systems allow users to find and access content even if an external entity is trying to prevent this, either by attempting to block specific content (e.g. by keyword), classes of content (e.g. video files), classes of websites and services (e.g. social networks), or block the use of the communication system itself (e.g. shutting down the Internet). Prior real-world experience demonstrates that nation-state-level adversaries are willing to engage in all these tactics [5, 18, 31]. Numerous potential solutions have been proposed [4, 7, 27], but the problem of *plausibly-deniable search and robust storage* remains elusive due to its seemingly contradictory set of requirements — how does a system maintain a searchable index of content for users and yet hide it from intermediate/relay nodes and volunteers who store content?

Any useful censorship resistant system must provide plausibly-deniable in-band search and content privacy on the wire. Protection for storers as well as intermediaries is vital, since we expect that any user's computer may be seized and examined by a powerful adversary [22], so the owner must be able to plausibly disavow knowledge of stored content. That same user must be able to search and find content in the network which may already be on his or her computer, but should not discover that it is stored locally. Prior work has partially addressed this by encrypting files and requiring out-of-band discovery of decryption keys, which makes reconstruction of content difficult. We describe a design for plausibly deniable search and robust storage for a censorship resistant network that supports *natural keyword search* while retaining deniability.[1] *Our design is self-contained — no out-of-band communication is required* to find content nor obtain decryption keys to decode files. This promotes usability and reduces users' real-world risks.

**One-way indexing.** To solve the problem we propose "one-way indexing," such that a user can search by keyword, but someone storing parts of the file cannot determine the content of the file or query. To publish file $F$ with keyword $kw$, Alice partitions it into three logical portions — the *content*, consisting of encrypted *blocks* $b_1, \ldots, b_k$ each indexed under ID $hash_1(b_i)$; the *content manifest*, containing a list of all block hashes (allowing retrieval of the file) and indexed as $hash_2(kw)$; and the *key manifest*, containing the file decryption key, indexed as $hash_3(kw)$. To retrieve a file, a user will search for $hash_2(kw)$ and $hash_3(kw)$, but any node not storing both manifests must invert the keyword hash in order to retrieve the other manifest and reconstruct the file, even if all file blocks are stored locally.

**Robust storage.** Censorship resistance requires perpetual and robust storage. We use both erasure coding and replication at publication time to achieve initial robustness, and maintain it without publisher intervention. Once the file has been stored as described above, nodes who store the file's content manifest lazily verify that a file is sufficiently replicated, freeing the original publisher from responsibility and providing added deniability. To prevent mitigate adversaries overwhelming the system with useless data, we incorporate lazy garbage collection, randomly selecting unused contents for dele-

*Part of this work was performed at the University of Minnesota.

[1] Legal precedent regarding the value of plausible deniability for users is beyond the scope of this work.

tion. We also allow content curating — editors[2] can "bless" important but unpopular files by signing their manifests, exempting them from deletion. The manifest holder periodically retrieves the file, so its blocks will avoid garbage collection.

We examine the requirements for censorship resistant and plausably deniable search in Section 2 and describe how our design addresses them in Section 3. Section 4 details both a theoretical and practical evaluation of our system security, performance, and survivability using a modified Azureus/Vuze DHT client [2] in a dynamic network environment (high node churn and frequent transient network faults). Finally, we discuss related work in Section 5 and conclude in Section 6.

## 2 Requirements

The set of challenges for censorship resistant search is distinct from other storage architectures, which usually implement access control and data confidentiality — the antithesis of censorship resistance goals. Censorship resistant systems should maintain plausible deniability for storers while preserving availability of *all content for all users* with overwhelming probability even when an adversary can remove or compromise an large constant fraction of the network [11]. In-band key management[3] is therefore difficult: nodes should be able to deny knowledge of local file content even while all other users can reconstruct and decrypt all files in the network.

The network consists of a large number of dynamic malicious adversaries, along with honest-but-curious *storers*, who volunteer storage space and route messages. We assume adversaries with the resources of nation-states who control the network infrastructure, compromise targeted nodes, and block usage of censorship resistant systems (but preserve Internet connectivity in general). Storers may also take the role of *publisher* to upload content, or *searcher* to find and download content.

**Discoverability.** To make the system easy to use, content should be easy to find, such as through keyword search.

**Deniability.** Users who have their communication monitored or computer confiscated and examined must be protected. Node-local data should not allow storers to infer the nature of the content they store or queries to which they respond, either in real-time or post-hoc, beyond the fact that queries may match a local file. Making recovery of individual queries or local files *difficult but not impossible is sufficient for plausible deniability* as long as bulk recovery is even more resource-intensive. Protecting identities of system users is also crucial, so while censorship resistance requires protocol obfuscation [16, 17] and identity concealment [13, 15, 23, 26], they are beyond the scope of our design — we are agnostic to how they are implemented and incorporated.

**Robustness.** Storage is provided by network nodes, each contributing a portion of the overall network storage capacity in a peer-to-peer (P2P) manner. Content is replicated to ensure availability with high probability even with high storer churn and attempted blocking.

## 3 System Design

To achieve self-contained plausibly deniable storage, we separate file content, metadata, and encryption keys, making this information only known to the publisher and a searcher, but not storers or arbitrary network nodes. For search we use distributed hash tables (DHTs), which are structured overlay networks that allow for efficient lookup and publication of key-value pairs [21, 24]. Each node has a logical DHT identifier; to distinguish DHT keys from cryptographic keys, for the remainder of this paper we will refer to DHT keys as DHT IDs.

We are agnostic to lower-level blocking-resistant protocols (e.g. membership concealment [15, 26] or darknets [13, 23]) that protect against network-layer attacks.

### 3.1 Publishing

Figure 1 shows the publication process. To publish file $F$, a publisher generates an ephemeral[4] asymmetric key pair $PK/SK$ and symmetric key $K$, then encrypts the file $F$ using a symmetric cipher keyed with $K$ to produce $E_K(F)$, and partitions the resulting ciphertext into *x blocks*.[5] To improve availability, each block is $m$-of-$n$ erasure coded [20] to yield a total of *xn encoded chunks* ($EC^n_m(E_K(F))$). The parameters of the erasure code determine the storage overhead — an $m$-of-$n$ code imposes a factor of $n$ increase on the amount of data stored in the system. Selection of $m$ and $n$ values which are "good enough" to store data indefinitely is discussed in Section 4. Each encoded chunk is then be inserted into the DHT, using the hash of the chunk content as storage location ID. The chunks are therefore self-verifying — the hash of the received chunk must match the requested hash and the hash in manifest [19].

**Manifests.** Each data file has two associated *manifests*, or pieces of metadata required to locate and identify content: the key manifest contains the decryption key $K$, and the content manifest lists the network locations of erasure-coded file chunks. The publisher compiles a list of $k$ keywords that describe the file contents, and includes the list of keyword hashes in the manifests, salted to slow down brute-force dictionary attacks. Salts can be arbitrarily long and each instance of a manifest for

[2] Editor selection is outside the scope of this paper.

[3] The network stores everything required to retrieve content

[4] These can be kept and used as long-term pseudonyms at the cost of publisher deniability.

[5] Once everything is uploaded, the publisher can discard all keys — since they become public, there is no benefit in keeping them.

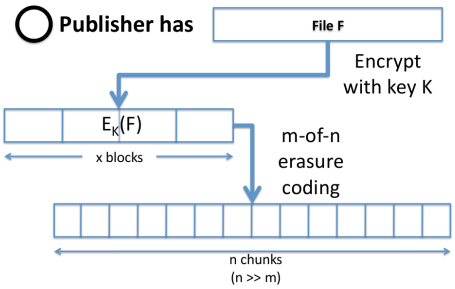


Figure 1: A publisher encrypts a file and then applies an $m$-of-$n$ erasure coding scheme.

the same file can use different salts. For indexing, one keyword per manifest must remain unsalted, otherwise clients must guess salt values in order to find files. Each manifest includes only one unsalted keyword selected from list $k$, and all salts are random, so each manifest is unique. Although it is sufficient to brute-force only the indexing keyword to partially compromise plausible deniability, honest nodes have no incentive to do so unless compelled,[6] and adversaries gain nothing, since they can directly search for any keywords of interest. Content manifests have the following format:

- $s_1, hash_2(s_1, kw_1), \cdots, s_k, hash_2(s_k, kw_k)$, a list of hashed salted keywords
- $hash_2(kw_i)$, a single unsalted hashed keyword from the list above, for indexing
- $C_1, C_2, \cdots, C_n$, a list of chunk storage locations (content hashes) in the DHT
- $hash_1(E_K(F))$, a hash of the file ciphertext before erasure coding, to verify reconstruction
- $h(F)$, a hash of the file plaintext before encryption and erasure coding, to verify decryption
- $PK$, the publisher's ephemeral public key
- $Sig_{SK}(\mathscr{M})$, the publisher's ephemeral signature over the manifest, to preserve integrity

Key manifests are identical except that they hold a plaintext copy of key $K$ instead of the chunk storage locations, and keywords are hashed using $hash_3$ instead of $hash_2$.

In order to block a file it is sufficient to deny access to all copies of either its key manifest or content manifest, so a particularly aggressive replication is needed to ensure availability. Manifests are not erasure-coded, but heavily replicated: each manifest is indexed by an unsalted keyword as well as a replica number, so a manifest with $k$ keywords and replication factor $r$ is stored under $rk$ DHT IDs $h(i, h(keyword_j))$ for $0 \leq i < r$, $0 \leq j < k$.

## 3.2 Search and retrieval

To retrieve a file, a searcher first obtains the file manifest by hashing meaningful search terms and fetching the corresponding values from the DHT. Search terms are hashed using a cryptographically-secure one-way hash function, so everyone forwarding the request and storing the results has plausible deniability as to the search target. Each query will return a number of manifests, and the searcher must determine which of them most closely matches the query keywords.[7] The searcher repeats the process for key manifests, but can stop after retrieving the first instance of a key manifest matching the file manifest (they will have identical $h(E_K(F))$). In parallel with key manifest retrieval, the searching node fetches any $m$-chunk subset of the $n$ content chunks listed in the file manifest to reconstruct the encrypted file. Reconstruction is successful if the results match $h(E_K(F))$. Decrypting using key $K$ from the key manifest and verifying that the plaintext matches $h(F)$ completes the process.

## 3.3 Storage, maintenance, and cleanup

**Content-oblivious replication.** To ensure continuous availability even in cases of large-scale blocking or failure, manifest storers continuously monitor the replication factor of manifests, and are referred to as *manifest "guarantors."* Key manifest holders do not know the locations of content chunks and cannot reconstruct the file, but content manifest holders can, and also maintain the replication factor of all file chunks listed in the manifest. Note that this requires no action on the part of the publisher. Content manifest guarantors can reassemble entire erasure-coded encrypted file, but cannot:

- Recover file plaintext
- Recover file keywords except by brute force
- Alter the manifest without breaking the signature scheme used to sign it
- Remove chunks or manifests from the network

Every time period $\tau$, guarantors examine their stored content manifests and search for a sample of those chunks to can probabilistically determine the *current* replication factor of a file and compare it to the *desired* replication factor. If the difference is significant, the node can download and reconstruct the encrypted file, apply the erasure code (obtaining copies of all chunks), and inserts missing chunks back into the DHT. Both content and key manifest storers do the same with the manifests themselves, checking to see if enough replicas are available, and creating new replicas if necessary. To prevent selective response to only refresh queries (to make the replication factor appear high), these probes must be indistinguishable from "real" file access.

**Garbage collection.** Every storer associates a timestamp of the original storage time with every locally stored

[6] Even if participants are forced to carry out dictionary attacks against their stored content, they can drop content with banned keywords without compromising content discovery, as humans easily avoid such keyword-based attacks in practice [9, 30].

[7] Alternatively, to minimize query time and bandwidth requirements, clients can first search for the least common keyword under which content of interest might be indexed, download those manifests, and then check additional salted keyword hashes locally.

chunk and manifest, and updates the timestamp every time that chunk is accessed by another network user. During idle times, nodes clear storage space by lazily examining their local content and probabilistically discarding anything with timestamps older than a global cutoff (e.g. one month). Manifest guarantors implicitly serve to refresh timestamps by accessing content to check the replication factor. Likewise, manifests that are not being actively accessed or are over-replicated can be probabilistically discarded by their guarantors. As manifests are dropped, file chunks to which they refer become "orphaned," and will be garbage-collected. Thus *two honest manifest holders, one for content and one for the key, are sufficient to maintain replication* of any piece of content with overwhelming probability, as long as the minimum number of chunks required to reconstruct the encrypted file can still be retrieved. This will not protect against junk data, but will increase adversarial workload.

**Curated content.** Our system is similar in concept to a massively distributed version of WikiLeaks [29], and in keeping with its spirit, we *preserve unpopular but important files by employing an editor-facilitated publishing model*. This hybrid model supports storage for data which has not received editor approval (whether for lack of examination or explicitly unapproved), but editor-approved data has special protection. We bootstrap the network with a set of hard-coded *editor public keys*, the private counterparts to which are held by a select group of pseudo-administrators who sign manifests[8] to add a "stamp of approval," excluding them from garbage collection by honest nodes. Since manifest holders refresh content in the network, chunk storers need never know whether they store editor-approved content or not.

Of course, editors introduce their own set of problems: malicious editors may sign "junk," but we must err on the side of content retention if we want to support storage in perpetuity, and *a single honest editor is sufficient* to ensure that important content is retained. Therefore we are not particularly concerned with malicious editors, since they cannot explicitly remove content from the system. While a free-for-all model where all published content is maintained indefinitely is attractive, it suffers from a number of drawbacks such as unregulated content quality, pollution and collision attacks, and storage space concerns, discussed further in Section 4.

## 4 Evaluation

**Plausible deniability.** We provide plausible deniability through separation of encrypted content from metadata and decryption key. The separation is one-way, i.e. access to one or more of the components does not yield the plaintext or keywords. Nodes who store individual chunks have no information regarding the plaintext,[9] and nodes who store content manifests can access the list of chunk locations in the network and a list of salted keyword hashes, but not the content plaintext. These nodes can reconstruct encrypted files, but cannot decrypt without a brute-force search of the decryption keyspace, or a brute-force search for at least one original keyword to fetch the key manifest. Finally, a node storing the key manifest does not have access to the content, the content locations, nor to the keywords. It would likewise need to invert keyword hashes through brute force to find the content manifest to download and reconstruct the file.

There is no advantage in storing chunks in conjunction with a manifest, but a peer storing both the key and content manifests can fetch, reconstruct, and decrypt the file. *However, this does not benefit honest-but-curious storers, who would sacrifice their own plausible deniability* by not following the protocol and storing both. Honest nodes should refuse a manifest store request, without examining manifests' content, if they already have the other manifest ($h(E_K(F))$ will be the same in both). Adversaries do not benefit from storing both manifests either, since the same goal can be accomplished by directly searching for keywords they wish to censor. Nonetheless, we minimize the risk of asking the same node to store both manifests by using different hash functions for each manifest type: $hash_1$ for chunk ID generation, $hash_2$ for content manifests, and $hash_3$ for key manifests.

**Robust storage.** As in [12], we express the *durability* $D$ of a file block when using $m$-of-$n$ erasure coding as

$$D = P(s \geq m) = \sum_{k=m}^{n} \binom{n}{k} (1 - f_{max})^k \cdot f_{max}^{n-k}.$$

Recall that every file block is erasure-coded to produce $n$ chunks, such that at least $m$ chunks are required to successfully decode each block; here, $s$ is the number of successfully-retrieved chunks, and $f_{max}$ is the failure probability of a node, assuming uniformly random failures and that all chunks are stored at different nodes. Therefore, if $x$ blocks are stored in the network, then the probability that *every file is recoverable* is $\rho(x) = D^x$. If $1 - \rho(x)$ is negligible then we say that a system is robust.

Figure 2 shows results of simulations to determine file robustness with erasure code parameters uniformly sampled from $[1,5]$ and $[5,500]$. Storage overhead of nearly a factor of 10 is needed for a robust censorship resistant system to support $2^{60}$ variable-size chunks, with a 50-of-500 code as the best trade-off between overhead and robustness. The resulting network can tolerate more than 70% node failure before losing data.

**System availability.** Protection from outright blocking at IP and protocol levels should be provided by

[8] Both the content and key manifests must be signed.

[9] Timing attacks may be used to identify correlated chunks, but not the nature of the content.

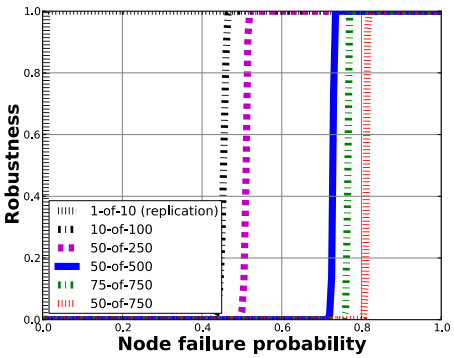



Figure 2: Inverse storage robustness $(1-\rho)$ using various erasure code configurations or replication.

lower-level mechanisms, such as identity concealment and darknets [13, 15, 23, 26], and we focus instead on plausible deniability and robustness. DHTs overlays are particularly vulnerable to certain classes of attack:

- **Storage exhaustion:** publishing as much as possible in order to exhaust all available network storage;
- **Pollution:** publishing irrelevant content indexed with keywords that adversaries wish to block; and
- **Keyword "squatting":** attempting to acquire DHT identifiers that place adversaries in logical network locations such that they control specific keywords, and block searches for them.

Any open-access P2P network is necessarily vulnerable to pollution and storage exhaustion. *Keyword squatting* is a problem if adversaries can influence the choice of their own DHT IDs and content is *only* replicated at locations logically close to the hash of the data, allowing attackers to block access to targeted content by denying access to a small subset of the network (a "neighborhood"). Inter-neighborhood replication and load balancing mitigates squatting attacks, so in addition to erasure coding we replicate each chunk and each manifest: for a given replication factor $r$, chunk $c$ is stored under $r$ DHT IDs $h(r,h(c))$ and a manifest with keyword $kw$ is stored under $h(r,h(kw))$ for $0 \leq i < r$. Assuming a uniform distribution of the resulting hash values, replicas should not exhibit neighborhood locality. Erasure coding and replication yields better results than either strategy alone [12].

A number of storage schemes deal with *pollution and exhaustion* by either requiring content to be refreshed periodically or by purging unpopular files [7, 8]. However, popularity-only maintenance is vulnerable to attack [14] and does not protect potentially important but infrequently accessed data, and periodic refresh places undue burden on the original publisher and greatly reduces deniability of the data source. In fact, requiring refresh *benefits the censor* since publishers now have the power to implicitly remove files from the network and can be compelled to do so. We use a hybrid scheme, composed of periodic refresh, popularity-based garbage collection, and *curated content* vetted by human editors. An additional twist to our refresh strategy is that network nodes *other than the publisher* perform refresh in a *content-oblivious* way, i.e. nodes are not aware of the nature of the content they are refreshing. Unrefreshed data is garbage-collected, so popular data is retained and unpopular non-vetted data is subject to deletion. This decreases, but does not completely eliminate, damage from pollution and exhaustion attacks by requiring adversaries to continually refresh their content in the network by accessing it or uploading it again. Human intervention will be required in any system, since we cannot algorithmically distinguish "useful" and irrelevant or junk content.

## 4.1 Performance

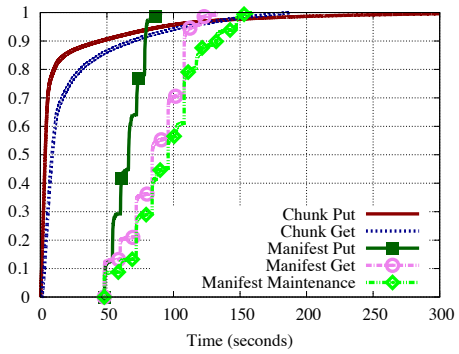
We implemented our system using the Azureus/Vuze DHT [2] [10] and measured the performance of 250 nodes deployed on the PlanetLab distributed testbed [6] and participating in Vuze control traffic, but not sharing files except with our own custom nodes. Vuze is not designed for bulk content transfer, so we added a TCP-based file transfer subsystem to a UDP-only Vuze client.

**Experimental setup.** At the start of our experiments, 20 random nodes each publish a randomly-generated 20MB file split into 40 512KB blocks and indexed with 5 to 15 randomly-selected keywords. To accommodate network dynamics, nodes retry publication for 10 seconds or until the desired replication factor is achieved. We currently do not implement erasure coding, but rather simulate it by increasing the replication factor, using 10 logically adjacent and 15 logically distant file chunk replicas and 15 logically distant manifest replicas. Each node performs *hourly maintenance*, checking the replication factor of each locally stored key manifest.

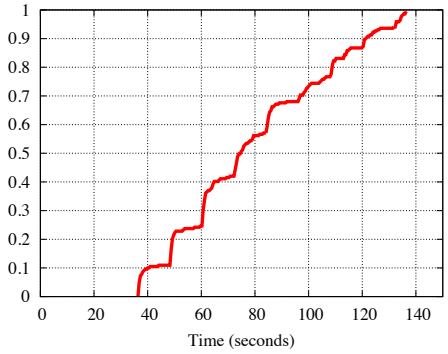
**Results.** We found our testbed to be highly dynamic, matching a partially adversarial network: Connectivity was intermittent and asymmetric,[11] between 10% and 15% of nodes failed silently while others disconnected and reconnected at unpredictable intervals (churn), and still more responded to connection requests but refused to store data. Our *best-case* reliable connectivity was between $27.5 \pm 0.3$ nodes of any 50-node subset.

Basic DHT operations — UDP ping and lookup — can be used as a baseline measurement of unmodified Vuze. Figure 3(a) shows that even trivial TCP operations, such as "chunk put" (upload), take 10 to 15 seconds due to the nature of our testing environment and not our protocol. Maintenance time depends on the size of the manifest, and on the number of keywords and chunks per file. Bandwidth-intensive operations, such as chunk put, require similar amounts of time as lower-bandwidth actions, and are always faster than get operations, which

[10] Any DHT would work; we picked Vuze to ease implementation.
[11] Node **A** can reach node **B** but **B** cannot reach **A**.




(a) Manifest maintenance, `put`, and `get` operations.




(b) Total time to retrieve a file, from search and download to reconstruction and decryption.

Figure 3: Cumulative distribution (CDF) of time required for various operations. Failures omitted — their values are either unrealistically low or infinitely high, biasing the results.

require more lookups. This suggests that we are bounded by latency rather than throughput.

Figure 3(b) shows the total time required for a client to obtain a file, including search, manifest download, content block fetch, and reconstruction and decryption. The median user retrieved a 20MB file in 65 to 85 seconds — reasonable for non-interactive bulk data transfer. Since we are limited by latency, parallel lookup and download will improve performance and minimize additional overhead for larger files. Furthermore, the time needed for manifest maintenance is comparable to the time needed to fetch a manifest. We conclude that the performance of our *unoptimized* client is acceptable in practice for publishers, searchers, and storers even in highly unstable network environments such as our testbed. These performance measurements incorporate a significant number of failed DHT operations (omitted from the graphs). While ICMP ping failure rate was 2.87%, 7.77% of Vuze UDP pings failed, 15.12% of Vuze DHT lookups failed, and 11.24% of maintenance operations failed due either to unsuccessful lookup or download, or inability to upload all needed replicas. 20% of nodes were responsible for 80% of these failures, but nodes with working network connections completed their lookups, and there were sufficient guarantors to maintain manifest replication.

## 5 Related work

Most censorship resistant systems either do not provide searchable content indexes or do not allow keyword-based search, and rely on client-side key management [28], difficult-to-remember "content hashes" — alphanumeric strings at least a dozen characters long [3, 7] — trusted directories [1], specialized trusted hardware, or third-party indexes. A number of systems avoid encryption by using secret sharing, splitting files across multiple servers so no single server holds enough secrets to reconstruct the file nor learn anything about its contents [10, 25]. None of these approaches can be used for censorship resistance since out-of-band key management would complicate search and retrieval, and neither specialized hardware nor directory servers are secure against a powerful adversary. Modifying existing systems for both in-band search and plausible deniability is not trivial, since decryption keys must be simultaneously discoverable by clients but not storers. Neither hashes of content plaintext nor keywords can serve as decryption keys since this undermines plausible deniability of the storer. Any search feature would have to be "one-way," so both encryption and metadata separation are required.

## 6 Conclusion

We described a plausibly-deniable censorship resistant search and storage system that provides flexible, robust, and self-contained peer-to-peer storage while resisting strong (state-level) adversaries. It combines "one-way indexing" for plausible deniability and easy keyword search, with erasure coding, aggressive replication, and content-oblivious replica maintenance for robust and highly available storage. It also mitigates pollution and exhaustion attacks though a combination of curated content and a self-cleaning mechanism. Very few honest nodes are needed to successfully maintain file replication even in the presence of adversaries and high member churn. Through simulation and implementation we showed that our design is practical, highly robust in low-reliability environments, and can support massive amounts of stored content ($2^{60}$ blocks) *with negligible loss* at node failure rates up to 70%.

## 7 Acknowledgments

The authors would like to thank Eric Myhre, Rob Jansen, James Tyra, and our anonymous reviewers for their help with an early version of this work. This work was supported in part by NSF grant 0917154.

## References


[1] ADYA, A. ET. AL. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In *OSDI* (2002).

[2] Azureus, now called Vuze : Bittorrent Client, 2011. `http://azureus.sf.net/`.

[3] BENNETT, K., GROTHOFF, C., HOROZOV, T., AND LINDGREN, J. T. An encoding for censorship-resistant sharing. Tech. rep., GNUnet, 2003.

[4] BENNETT, K., HOROZOV, C. G. T., AND PATRASCU, I. Efficient sharing of encrypted data. In *ACISP* (2002).

[5] China 'blocks' iTunes music store. *BBC News*. August 22, 2008.

[6] CHUN, B., CULLER, D., ROSCOE, T., BAVIER, A., PETERSON, L., WAWRZONIAK, M., AND BOWMAN, M. PlanetLab: An overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev. 33*, 3 (2003).

[7] CLARKE, I., SANDBERG, O., WILEY, B., AND HONG, T. W. Freenet: A distributed anonymous information storage and retrieval system. In *PET* (2000).

[8] COX, L., AND NOBLE, B. Samsara: Honor among thieves in peer-to-peer storage. *SIGOPS OS Review 37*, 5 (2003).

[9] Creative Chinese dodge censors to search for 'Uncle Jiang'. *The Indian Express* (2011).

[10] DINGLEDINE, R., FREEDMAN, M. J., AND MOLNAR, D. The Free Haven project: Distributed anonymous storage service. In *PET* (2000).

[11] FIAT, A., AND SAIA, J. Censorship resistant peer-to-peer content addressable networks. In *SODA* (2002).

[12] HAEBERLEN, A., MISLOVE, A., AND DRUSCHEL, P. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *NSDI* (2005).

[13] ISDAL, T., PIATEK, M., KRISHNAMURTHY, A., AND ANDERSON, T. Privacy-preserving P2P data sharing with OneSwarm. *SIGCOMM Comput. Commun. Rev. 40* (2010).

[14] KÜGLER, D. An analysis of GNUnet and the implications for anonymous, censorship-resistant networks. In *PET* (2003).

[15] MITTAL, P., CAESAR, M., AND BORISOV, N. X-Vine: Secure and pseudonymous routing using social networks. In *NDSS* (2012).

[16] MOGHADDAM, H. M., LI, B., DERAKHSHANI, M., AND GOLDBERG, I. SkypeMorph: Protocol obfuscation for Tor bridges. Tech. Rep. 8, Cheriton School of Computer Science, University of Waterloo, 2012.

[17] Tor project: obfsproxy. Accessed February 15, 2012.

[18] Pakistan blocks YouTube website. *BBC News* (2008).

[19] QUINLAN, S., AND DORWARD, S. Venti: A new approach to archival storage. In *FAST* (2002).

[20] RABIN, M. O. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM 36*, 2 (1989).

[21] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A scalable content-addressable network. *SIGCOMM Comput. Commun. Rev. 31*, 4 (2001).

[22] RISEUP.NET. Server seizure, April 2012.

[23] SANDBERG, O. Distributed routing in small-world networks. In *ALENEX* (2006).

[24] STOICA, I., MORRIS, R., LIBEN-NOWELL, D., KARGER, D., KAASHOEK, M. F., DABEK, F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for Internet applications. In *SIGCOMM* (2001).

[25] STORER, M. W., GREENAN, K. M., MILLER, E. L., AND VORUGANTI, K. POTSHARDS: Secure long-term storage without encryption. In *USENIX Technical* (2007).

[26] VASSERMAN, E. Y., JANSEN, R., TYRA, J., HOPPER, N., AND KIM, Y. Membership-concealing overlay networks. In *CCS* (2009).

[27] WALDMAN, M., AND MAZIÈRES, D. Tangler: A censorship-resistant publishing system based on document entanglements. In *CCS* (2001).

[28] WALDMAN, M., RUBIN, A., AND CRANOR, L. Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system. In *USENIX Security* (2000).

[29] WikiLeaks. `http://wikileaks.org/`, 2008.

[30] YE, J., AND FOWLER, G. A. Chinese bloggers scale the 'Great Firewall' in riot's aftermath. *The Wall Street Journal* (2008).

[31] ZITTRAIN, J., AND EDELMAN, B. Internet filtering in China. *IEEE Internet Computing 7*, 2 (2003).