The Free Haven Project:
Distributed Anonymous Storage Service

Roger Dingledine, MIT, arma@mit.edu
Michael J. Freedman, MIT, mfreed@mit.edu
David Molnar, Harvard University, dmolnar@fas.harvard.edu

December 17, 2000

Abstract

We present a design for a system of anonymous storage which resists the attempts of powerful adversaries to find or destroy any stored data. We enumerate distinct notions of anonymity for each party in the system, and suggest a way to classify anonymous systems based on the kinds of anonymity provided. Our design ensures the availability of each document for a publisher-specified lifetime. A reputation system provides server accountability by limiting the damage caused from misbehaving servers. We identify attacks and defenses against anonymous storage services, and close with a list of problems which are currently unsolved.

1 Introduction

Anonymous publication and storage services allow individuals to speak freely without fear of persecution, yet such systems remain poorly understood. Political dissidents must publish in order to reach enough people for their criticisms of a regime to be effective, yet they and their readers require anonymity. Less extreme examples involve cases in which a large and powerful private organization attempts to silence its critics by attacking either the critics themselves or those who make the criticism publically available. Additionally, the recent controversy over Napster and Gnutella has highlighted both a widespread demand for anonymous publication services for non-political purposes, and the consequences of such services failing to provide the anonymity expected.

Systems meeting these needs are just starting to be deployed, and the exact requirements and design choices are not yet clear. Events in 1999 and 2000 have highlighted some shortcomings of already deployed systems; the identification and removal of Napster users who downloaded Metallica songs[30] and the Gnutella Wall of Shame[12] are two examples. These shortcomings led to the development of a new generation of anonymous publication services, such as Freenet[11], which focus specifically on providing anonymity.

It is in this spirit that the Free Haven Project aims to design, implement, and deploy a functioning distributed anonymous *storage* service. We distinguish storage from publication in that storage services focus less on accessibility and more on persistence of data. In the process, we hope to clarify some of the requirements for such systems and highlight design choices.

It is not enough simply to talk about "anonymous" storage and publication. In section 2, we enumerate the many different *kinds* of anonymity which cover different aspects of the system, all important for the realization of a truly anonymous system.

Free Haven meets these requirements with a design based on a community of servers called the *servnet*. Each server, or *servnet node*, holds pieces of some documents. These pieces are called *shares*. In addition, each servnet node has a persistent identification or *pseudonym* which allows it to be identified by other servnet nodes or potential Free Haven users. Section 3 describes the design of the Free Haven system and the operations that it supports, including inserting and retrieving documents.

We chose to use a network of pseudonymous servers in order to give each server a *reputation*. This reputation allows servers to be 'paid' without needing the robust digital cash scheme required for systems such as Anderson's Eternity Service[2]. Servers form contracts to store given shares for a certain period of time; successfully fulfilling the contract increases the server's reputation and consequently its ability to store some of its own data on other servnet nodes. This gives an incentive for each server to behave well as long as cheating servers can be identified. We show a technique for identifying cheating servers in section 3.7.

The overall idea is similar to the "give up space now, get space forever" scheme used in Intermemory[10], but allows servers to lose reputation if they start behaving badly. In section 3.9 we discuss the *reputation system*, which is the system that keeps track of trust in each server.

Some of the contracts between servers are formed when a user inserts data into the servnet. Most of them, however, will be formed when two servers swap shares by *trading*. Trading allows the servnet to be *dynamic* in the sense that servnet nodes can join and leave easily and without special treatment. To join, a servnet node starts building up a reputation by storing shares for others. To leave, a server trades away all of its shares for short-lived shares, and then waits for them to expire. The benefits and mechanisms of trading are described in section 3.5.

Such a system has powerful adversaries which can launch a range of attacks. We describe some attacks on the Free Haven design in section 4 and show how well the design does (or does not) resist each attack. We then compare our design with other systems aimed at anonymous storage and publication using the kinds of anonymity described in section 6, allowing us to distinguish systems which at first glance look very similar. We conclude with a list of challenges for anonymous publication and storage systems, each of which reflects a limitation in the current Free Haven design.

2 Anonymity for Anonymous Storage

The word "anonymous" can mean many different things. Some systems claim "anonymity" without specifying a precise definition. While the anonymity requirements of communication channels have been considered previously in depth [6, 19], we are not aware of a similar investigation into the requirements for publication and storage systems.

Information is stored in units called *documents*. The *author* of a document is the entity which initially created the document. The *publisher* of a document is the entity which places the document into the system. Documents may have *readers*, which are entities who retrieve the document from the system. An anonymous storage system may have *servers*, which are participants who provide special services required to keep the system running, such as dedicated disk space or bandwidth.

We do not give formal anonymity definitions here. Instead, we attempt to lay the groundwork for future definitions by enumerating different aspects of anonymity relevant to anonymous storage. This enumeration will allow us to compare Free Haven with related work.

In all of these notions of anonymity, there are at least three distinct subnotions based on what the adversary is assumed to already know. A document may be picked first, and then the adversary wishes to learn who authored, read, published, and so on. A user may be picked first, and the adversary wishes to know which documents the user authored, read, published, and so on. Finally, an adversary may know a document *and* a user, and then attempt to confirm its suspicion that the two are linked.

Author Anonymity: A system is author anonymous if an adversary cannot link an author to a document.

Publisher Anonymity: A system is publisher anonymous if it prevents an adversary from linking a publisher to a document.

Reader Anonymity: To say that a system has reader anonymity means that a document cannot be linked with its readers. Reader anonymity protects the privacy of a system's users.

Server Anonymity: Server anonymity means no server can be linked to a document. Here, the adversary always picks the document first. That is, given a document's name or other identifier, an adversary is no closer to knowing which server or servers on the network currently possess this document.

Document Anonymity: Document anonymity means that a server does not know which documents it is storing. Document anonymity is crucial if mere possession of some file is cause for action against the server, because it provides protection to a server operator even after his or her machine has been seized by an adversary. This notion is sometimes also known as 'plausible deniability', but see below under query anonymity.

Passive-server document anonymity means that if the server is allowed to look only at the data that it is storing, it is unable to figure out the contents of the document. This can be achieved via some sort of secret sharing mechanism. That is, multiple servers split up either the document or an encryption key that recreates the document (or both). An alternative approach is to encrypt the document before publishing, using some key which is external to the server.

Active-server document anonymity refers to the situation in which the server is allowed to communicate and compare data with all other servers. Since an active server may act as a reader and do document requests itself, active-server document anonymity seems difficult to achieve without some trusted party that can distinguish server requests from "ordinary" reader requests.

Query-Anonymity: Query anonymity means that the server cannot determine which document it is serving when satisfying a reader's request. For an overview of private information retrieval (PIR), see [31]. A weaker form of query anonymity is *server deniability* – the server knows the identity of the requested document, but no third party can be sure of its identity. Query anonymity can provide another aspect of 'plausible deniability'. This concept is related to deniable encryption[7].

It seems that some of these notions of anonymity may imply each other. We leave this investigation as future work.

2.1 Anonymity and Pseudonymity

So far, we have restricted ourselves to describing anonymity. We extend these notions to allow for the use of *pseudonyms*: if two transactions in the system can be linked, then the attributes which allow them to be linked make up a pseudonym. For example, in an *author-pseudonymous* system, the documents digitally signed by "Publius" could all be verified as "belonging to Publius" without anyone coming to know who "Publius" is in 'real life.'

Both anonymity and pseudonymity protect the privacy of the user's *location* and *true name*. Location refers to the actual physical connection to the system. The term "true name" was introduced by Vinge[48] and popularized by May[33] to refer to the legal identity of an individual. Knowing someone's true name or location allows you to hurt him or her.

Many different actions can be linked to the same pseudonym, while an anonymous system allows no linking at all. This allows the pseudonym to acquire a *reputation*. Free Haven uses pseudonyms to give each server a reputation; the reputation influences how much data a server can store and provides an incentive to act correctly.

2.2 Partial Anonymity

Often an adversary can gain some partial information about the users of a system, such as the fact that they have high-bandwidth connections or all live in California. Preventing an adversary from obtaining *any* such information may be impossible. Instead of asking "is the system anonymous?" the question shifts to "is it anonymous enough?"

We might say that a system is *partially anonymous* if an adversary can only narrow down a search for a user to one of a "set of suspects." If the set is large enough, it is impractical for an adversary to act as if any single suspect were guilty. On the other hand, when the set of suspects is small, mere suspicion may cause an adversary to take action against all of them.

An alternate approach to classifying levels of anonymity is presented by [41], where anonymity levels for users range from "exposed" to "beyond suspicion". These levels are in terms of an idealized adversary's reasonable belief that a user or set of users has performed some particular action.

Independently, Syverson and Stubblebine have developed a logic for talking about the adversary's view of a set of suspects[46]. The logic gives a formal meaning to a "set of suspects" and the notion of an adversary's belief.

2.3 Reasoning about Anonymity

Suppose an author signs his true name to a document before placing it into an anonymous publication system. Is the system still anonymous? This situation raises a crucial question: where does the responsibility of an anonymous publication system begin, and where does it end? What can such a system reasonably be expected to protect? We can give an answer to these questions by explicitly specifying a model for anonymous publication.

We model anonymous publication systems as a single entity (call it Ted) which coordinates communication between other entities in the network. In our model we have a set of senders $\{Alice_i\}$, and a set of recipients $\{Bob_j\}$. When an Alice sends a message to a Bob, Ted receives the message and delivers it to the appropriate Bob. The privacy characteristics of Ted as a communication channel define the level of anonymity that Ted provides.

These privacy characteristics include linkability; ability to reply, persistence of this ability, privacy of this reply; content leaks; channel leaks; persistence of speech; and authorized readers. We emphasize that Ted is not simply a "trusted third party" (despite the name), but provides a specific set of characteristics and does not provide others. For a more complete look at privacy characteristics, look at the first author's thesis [13].

In addition, we will need to complicate this notion with other characteristics, such as reliability of delivery, cost of using a given path, and availability and fragility of the network.

Thus if we can convince ourselves that a given anonymous publishing design is in some sense 'equivalent' to a Ted with certain privacy characteristics, then we can more easily reason about the level of protection provided by that design – by reasoning instead about Ted. In particular, we can ask the question "what is the responsibility of the system" with respect to Ted.

More formally, for each message M_i which $Alice_i$ sends, there is some probability distribution D_i which describes the chance of each Bob being the recipient of the message. If we can replace Ted with a decentralized system which provides an indistinguishable probability distribution for all messages, then we have shown that the decentralized system is equivalent to thi Ted. This may give us an easier way to differentiate between the level of anonymity provided by various projects, because comparing Teds is easier and more intuitive than trying to reason about the effects of trading or caching issues directly.

This description requires significant work before it can become a formal model. For instance, we need to define exactly what we mean by privacy characteristics and enumerate them all; we need to figure out what it means for a probability distribution to be equivalent in this context; and we need to determine exactly how to describe a probability distribution over a complex system like Freenet or Mojo Nation.

3 The Free Haven Design

The overall system consists of the publication system, which is responsible for storing and serving documents, and the communications channel, which is responsible for providing confidential and anonymous communications between parties. This section focuses on the design of the publication system as a back-end for the communications channel.

The agents in our publication system are the **publisher**, the **server**, and the **reader**. These agents are layered over the communications channel; currently they communicate with one another via addresses which are implemented as remailer reply blocks[34]. Remailer reply blocks are a collection of encrypted routing instructions which serve as an address for a pseudonym on the Cypherpunks remailer network.

Publishers are agents that wish to store documents in the service; servers are computers which store data for publishers; and readers are people who retrieve documents from the service.

Free Haven is based on a community of servers called the *servnet*. In this community, each server hosts data from the other servers in exchange for the opportunity to store its own data in the network. The servnet is dynamic: data moves from one server to another every so often, based on each server's trust of the others. Servers transfer data by trading. That is, the only way to introduce a new file into the system is for a server to use (and thus provide) more space on its local system. This new file will migrate to other servers by the process of trading.

Each server has a public key and one (or more) reply blocks, which together can be used to provide secure, authenticated, pseudonymous communication with that server. Every machine in the servnet has a database which contains the public keys and reply blocks of the other servers on the network.

Documents are split into shares and stored on different servers. Publishers assign an expiration date to documents when they are published; servers make a promise to keep their shares of a given document until its expiration date is reached. To encourage honest behavior, some servers check whether other servers "drop" data early, and decrease their trust of such servers. This trust is monitored and updated by use of a *reputation system*. Each server maintains a database containing its perceived reputation of the other servers.

3.1 Publication

When an author (call her Alice) wishes to store a new document in Free Haven, she must first identify a Free Haven server which is willing to store the document for her. Alice might do this by running a server herself. Alternatively, some servers might have public interfaces or have publically available reply blocks and be willing to publish data for others.

To introduce a file F into the servnet, the publishing server first uses Rabin's information dispersal algorithm (IDA) [40] to break the file into shares $f_1, \ldots, f_n$ where any k shares are sufficient to recreate F. The server then generates a key pair (PK_{doc}, SK_{doc}), constructs and signs a data segment for each share f_i, and inserts those segments as new data into its local server space. Attributes in each share include a timestamp, expiration information, the public key which was used to sign it (for integrity verification), information about share numbering, and the signature itself.

The robustness parameter k should be chosen based on some compromise between the importance of the file and the size and available space. A large value of k relative to n makes the file more brittle, because it will be unrecoverable after a few shares are lost. On the other hand, a smaller value of k implies a larger share size, since more data is stored in each share.

We maintain a content-neutral policy towards documents in the Free Haven system. That is, each server agrees to store data for the other servers without regard for the legal or moral issues for that data in any given jurisdiction. For more discussion of the significant moral and legal issues that anonymous systems raise, we refer to the first author's thesis[13].

3.2 Retrieval

Documents in Free Haven are indexed by $H(PK_{doc})$, the hash of the public key from the keypair which was used to sign the shares of the document. Readers must locate (or be running) a server that performs the document request. The reader generates a key pair $(PK_{client}, SK_{client})$ for this transaction, as well as a one-time remailer reply block. The server broadcasts a request $H(PK_{doc})$, along with the client's public key, PK_{client}, and the reply block. This request goes to all the other servers that the initial server knows about. These broadcasts can be queued and then sent out in bulk to conserve bandwidth.

Each server that receives the query checks to see if it has any shares with the requested hash of PK_{doc}. If it does, it encrypts each share using the public key PK_{client} enclosed in the request, and then sends the encrypted share through the remailer to the enclosed address. These shares will magically arrive out of the ether at their destination; once enough shares arrive (k or more), the client recreates the file and is done.

3.3 Share Expiration

Each share includes an expiration date chosen at share creation time. This is an absolute (as opposed to relative) timestamp indicating the time after which the hosting server may delete the share with no ill consequences. Expiration dates should be chosen based on how long the publisher wants the data to last; the publisher has to consider the file size and likelihood of finding a server willing to make the trade.

By allowing the publisher of the document to set its expiration time, Free Haven distinguishes itself from related works such as Freenet and Mojo Nation that favor frequently requested documents. We think this is the most useful approach to a persistent, anonymous data storage service. For example, Yugoslav phone books are currently being collected "to document residency for the close to one million people forced to evacuate Kosovo"[37]; those phone books might not have survived a popularity contest. The Free Haven system is designed to provide privacy for its users. Rather than being a publication system aimed at convenience like Freenet, it is designed to be a private low-profile storage system.

3.4 Document Revocation

Some publishing systems, notably Publius, allow for documents to be "unpublished" or *revoked*. Revocation has some benefits. It would allow the implementation of a read-write filesystem, and published documents could be updated as newer versions became available.

Revocation could be implemented by allowing the author to come up with a random private value x, and then publishing some hash $H(x)$ inside each share. To revoke the document, the author could broadcast his original value x to all servers as a signal to delete the document.

On the other hand, revocation allows new attacks on the system. Firstly, it complicates accountability. Revocation requests may not reach all shares of a file, due either to a poor communication channel or to a malicious adversary who sends unpublishing requests only to some members of the servnet. Secondly, authors might use the same hash for new shares, and thus "link" documents. Adversaries might do the same to make it appear that the same author published two unrelated documents. Thirdly, the presence of the hash in a share assigns "ownership" to a share that is not present otherwise. An author who remembers his x has evidence that he was associated with that share, thus leaving open the possibility that such evidence could be discovered and used against him later (that is, breaking forward author anonymity).

One of the most serious arguments against revocation was raised by Ross Anderson [2]. If the capability to revoke exists, an adversary has incentive to find who controls this capability, and threaten or torture him until he revokes the document.

We could address this problem by making revocation optional: the share itself could make it clear whether that share can be unpublished. If no unpublishing tag is present, there would be no reason to track down the author. (This solution is used in Publius.) But this too is subject to attack: if an adversary wishes to create a pretext to hunt down the publisher of a document, he can republish the document *with* a revocation tag, and use that as "reasonable cause" to target the suspected publisher.

Because the ability to revoke shares may put the original publisher in increased physical danger, as well as allow new attacks on the system, we chose to leave revocation out of the current design.

3.5 Trading

In the Free Haven design, servers periodically trade shares with each other. There are a number of reasons why servers trade:

To provide a cover for publishing: If trades are common, there is no reason to assume that somebody offering a trade is the publisher of a share. Publisher anonymity is enhanced.

To let servers join and leave: Trading allows servers to exit the servnet gracefully by trading for short-lived shares and then waiting for them to expire. This support for a dynamic network is crucial, since many of the participants in Free Haven will be well-behaved but transient relative to the duration of the longer-lived shares.

To permit longer expiration dates: Long-lasting shares would be rare if trading them involved finding a server that promised to be available for the next several years.

To accomodate ethical concerns of server operators: Frequent trading makes it easy and unsuspicious for server operators to trade away a particular piece of data with which they do not wish to be associated. If the Catholic Church distributes a list of discouraged documents, server operators can use the hash of the public key in each share to determine if that document is in the list, then trade away the share without compromising their reputation as a server or the availability of the document. In a non-dynamic environment, the server would suffer a reputation hit if it chose not to keep the document. While we do not currently offer this functionality, trading allows this flexibility if we need it down the road. In particular, the idea of servers getting 'ISP exemption' for documents they hold currently seems very dubious.

To provide a moving target: Encouraging shares to move from server to server through the servnet means that there is never any specific, static target to attack.

The frequency of trading should be a parameter set by the server operator. When a server Alice wants to make a trade, she chooses another server Bob from her list of known servers (based on reputation), and offers a share x and a request for size and/or duration of a return share. If Bob is interested, he responds with a share y of his own.

Trades are considered "fair" based on the two-dimensional currency of $size \times duration$. That is, the bigger the size and the longer the document is to be held, the more expensive the trade becomes. The price is adjusted based on the preferences of the servers involved in the trade.

The negotiation is finalized by each server sending an acknowledgement of the trade (including a receipt, as described in section 3.6) to the other. In addition, each server sends a receipt to both the buddy of the share it is sending and the buddy of the share it is receiving; buddies and the accountability they provide are described in section 3.7. Thus, the entire trading handshake takes four rounds: the first two to exchange the shares themselves, and the next two to exchange receipts while at the same time sending receipts to the buddies.

By providing the receipt on the third round of the trading handshake, Alice makes a commitment to store the share y. Similarly, the receipt that Bob generates on the fourth round represents a commitment to store the share x. Bob could cheat Alice by failing to continue the protocol after the third step: in this case, Alice has committed to keeping the share from Bob, but Bob has not committed to anything.

At this point, Alice's only recourse is to broadcast a complaint against Bob and hope that the reputation system causes others to recognize that Bob has misbehaved. The alternative is to use a fair exchange protocol[35, 20], which is unreasonably communications-intensive without a trusted third party.

When Alice trades a share to a server Bob, Alice should keep a copy of the share around for a while, just in case Bob proves untrustworthy. This will increase the amount of overhead in the system by a factor of two or so (depending on duration), but provide greatly increased robustness. In this case, when a query is done for a share, the system responding should include a flag for whether it believes itself to be the "primary provider" of the data, or just happens to have a copy still lying around. The optimum amount of time requires further study.

3.6 Receipts

A receipt contains a hash of the public keys for the source server and the destination server, information about the share traded away, information about the share received, and a timestamp. For each share, it includes a hash of that document's key, which share number it was, its expiration date, and its size.

This entire set of information about the transaction is signed by server Alice. If Bob (or any other server) has to broadcast a complaint about the way Alice handled the transaction, furnishing this receipt along with the complaint will provide some rudimentary level of "proof" that Bob is not fabricating his complaint. Note that the expiration date of both shares is included within the receipt, and the signature makes this value immutable. Thus, other servers observing a receipt can easily tell whether the receipt is still "valid"—that is, they can check to see whether the share is still supposed to be kept on A. The size of each share is also included, so other servers can make an informed decision about how influential this transaction should be on their perceived reputation of the two servers involved in the trade.

We really aren't treating the receipt as proof of a transaction, but rather as proof of *half* of a transaction – an indication of a commitment to keep a given share safe. This is because the trading protocol is not bulletproof: the fact that Alice has a receipt from Bob could mean that they performed a transaction, or it could mean that they performed 3 out of the 4 steps of the transaction, and then Alice cheated Bob and never gave him a receipt. Thus, the most a given server can do when it detects a misbehaving server is broadcast a complaint and hope the reputation system handles it correctly.

3.7 Accountability

Malicious servers can accept document shares and then fail to store them. If enough shares are lost, the document is unrecoverable. Malicious servers can continue their malicious behavior unless there are mechanisms in place for identifying and excising them.

We propose a "buddy system" that creates an association between pairs of shares from a given document. Each share is responsible for maintaining information about the location of the other share, or *buddy*. When a share moves, it notifies its buddy.[1]

Periodically, a server holding a given share should query for its buddy, to make sure its buddy is still alive. If the server that is supposed to contain its buddy stops responding, the server with the share making the query is responsible for reporting an anomaly. This server announces which server had responsibility for the missing share when it disappeared. The results of this announcement are described below under section 3.9.

We considered allowing abandoned shares to optionally spawn a new share if their buddy disappears, but discarded this notion. Buddy spawning would make the service much more robust, since lost shares can be regenerated. However, such spawning could cause an exponential population explosion of shares for the wrong reasons. If two servers are out of touch for a little while but are not misbehaving or dead,

[1] More precisely, both the server it's moving from and the server it's moving to notify the buddy, as described in section 3.5.

both shares will end up spawning new copies of themselves. This is a strong argument for not letting shares replicate.

When a share x moves to a new machine, there are two *buddy notifications* sent to its buddy x'. But since the communications channel we have chosen currently has significant latency, a notification to x' might arrive after x' has already been traded to a new server. The old server is then responsible for forwarding these buddy notifications to the new server which it believes currently holds x'. Since the old server keeps a receipt as a record of the transaction, it can use this information to remember the new location of x'. The receipt, and thus the forwarding address, is kept by the old server until the share's expiration date has passed.

When a buddy notification arrives at a server which has traded away the share, the forwarder is checked and the notification is forwarded as appropriate. This forwarding is *not* done in the case of a document request, since this document request has presumably been broadcast to all servers in the servnet.

We have attempted to distinguish between the design goals of robustness and accountability. The system is quite robust because a document cannot be lost until a high percentage of its shares has been lost. Accountability, in turn, is provided by the buddy checking and notification system among shares, which protects against malicious or otherwise ill-behaving servers. Designers can choose the desired levels of robustness and accountability independently.

3.8 Communications Channel

The Free Haven design requires a means of anonymously passing information between agents. One such means is the remailer network, including the Mixmaster remailers first designed by Lance Cottrell [17]. Other examples of anonymous communication channels are Onion Routing[47] and Zero Knowledge Systems' Freedom[18]. We refer to David Martin's thesis for a comprehensive overview of anonymous channels in theory and practice[32].

The design and implementation of an anonymous communication channel is an ongoing research topic [1, 6, 8, 9, 22, 24, 25, 28, 38, 41]. The first implementation of the Free Haven design will use the Cypherpunk and Mixmaster remailers as its anonymous channel. For design details, see [16].

3.9 Reputation System

The reputation system in Free Haven is responsible for creating accountability. Accountability in a system so committed to anonymity is a difficult task. There are many opportunities to try to take advantage of other servers, such as merely neglecting to send a receipt after a trade, or wrongly accusing another server of losing a share. Some of the attacks are quite insidious and complex. Some history and issues to consider when developing a reputation system can be found in much more detail in [14].

Other systems exist which use reputations to ensure correct or "better" operation. The most directly relevant is the PGP Web of Trust model for public keys[39]. Other systems include the Advogato[29] and Slashdot message moderation systems, AOL's Instant Messenger[3], and much of real world commerce and law. In another vein, MANET[27] is a DARPA project to produce "a compromise-tolerant structure for information gathering."

Careful trust management should enable each server to keep track of which servers it trusts. Given the large number of shares into which documents are divided—and the relatively few shares required to reconstitute a document—no document should be irretrievably lost unless a large number of the servers prove evil.

Each server needs to keep two values that describe each other server it knows about: reputation and credibility. Reputation signifies a belief that the server in question will obey the Free Haven Protocol. Credibility represents a belief that the utterances of that server are valuable information. For each of these two values, each server also needs to maintain a confidence rating. This serves to represent the "stiffness" of the reputation and credibility values.

Servers should broadcast *referrals* in several circumstances, such as when they log the honest completion of a trade, when they suspect that a buddy of a share they hold has been lost, and when the reputation or credibility values for a server change substantially.

3.10 Introducers

Document request operations are done via broadcast. Each server wants to store its documents on a lot of servers, and if it finds a misbehaving server it wants to complain to as many as possible. But how do Free Haven servers discover each other?

The reputation system provides an easy method of adding new servers and removing inactive ones. Servers that have already established a good reputation act as "introducers." New servers can contact these introducers via the anonymous communication channel; the introducers will then broadcast referrals of this new server. This broadcast by itself does not imply an endorsement of the new server's honesty or performance; it is simply an indication that the new server is interested in performing some trades to increase its reputation. Likewise, a server may mark another as "dormant" given some threshold of unanswered requests. Dormant servers are not included in broadcasts or trade requests. If a dormant server starts initiating requests again, we conclude it is not actually dormant and resume sending broadcasts and offering trades to this server.

3.11 Implementation Status

The Free Haven system is still in its design stages. Although we have a basic proof-of-concept implementation, we still wish to firm up our design, primarily in the areas of accountability and bandwidth overhead. Before deploying any implementation, we want to convince ourselves that the Free Haven system offers better anonymity than current systems. Still, the design is sufficiently simple and modular to allow both a straightforward basic implementation and easy extensibility.

4 Attacks on Free Haven

Anonymous publishing and storage systems will have adversaries. The attacks and pressures that these adversaries may employ might be technical, legal, political, or social in nature. The system's design and the nature of anonymity it provides also affect the success of non-technical attacks.

We now consider possible attacks on the Free Haven system based on their respective targets: on the availability of documents and servnet operation; on the accountability offered by the reputation system; and on the various aspects of anonymity relevant to anonymous storage and publication, as described in section 2. For a more in-depth consideration of attacks, we refer to [13].

This list of attacks is not complete. In particular, we do not have a systematic discussion of what *kinds* of adversaries we expect. Such a discussion would begin with the most powerful adversaries possible, asking questions like "what if the adversary controls all but one of the servers in the servnet?" and scaling back from there. In analyzing systems like Free Haven, it is not enough to look at the everyday, plausible scenarios – every effort must be made to provide security against adversaries more powerful than any the designers ever expect. Indeed, adversaries have a way of being more powerful than anyone ever expects.

4.1 Attacks on Documents or the Servnet

Physical attack: Destroy a server.

Prevention: Because we are breaking documents into shares and only k of n shares are required to reconstruct the document, an adversary must find and destroy many servers before availability is compromised.

Legal action: Find a physical server, and prosecute the owner based on its contents.

Prevention: Because of the passive-server document anonymity property that the Free Haven design provides, the servnet operator may be able to plausibly deny knowledge of the data stored on his computer. This depends on the laws of the country in question.

Social pressure: Bring various forms of social pressure against server administrators. Claim that the design is patented or otherwise illegal. Sue the Free Haven Project and any known server administrators. Conspire to make a cause "unpopular", convincing administrators that they should manually prune their data. Allege that they "aid child pornographers" and other socially-unacceptable activities.

Prevention: We rely on the notion of jurisdictional arbitrage. Information illegal in one place is frequently legal in others. Free Haven's content-neutral policies mean that there is no reason to expect that the server operator has looked at the data he holds, which might make it more difficult to prosecute. We further rely on having enough servers in enough different jurisdictions that organizations cannot conspire to intimidate a sufficient fraction of servers to make Free Haven unusable.

Denial of service: Attack the servnet by continued flooding of queries for data or requests to join the servnet. These queries may use up all available bandwidth and processing power for a server.

Prevention: We must assume that our communications channel has adequate protection and buffering against this attack, such as the use of client puzzles [26]. Most communications channels we are likely to choose will not protect against this attack. This is a real problem.

Data flooding: Attempt to flood the servnet with shares, to use up available resources.

Prevention: The trading protocol implicitly protects against this type of denial of service attack against storage resources. The ability to insert shares, whether "false" or valid, is restricted to trading: that server must find another which trusts its ability to provide space for the share it would receive in return.

Similarly, the design provides protection against the corrupting of shares. Altering (or "spoofing") a share cannot be done, because the share contains a particular public key, and its integrity is verifiable by that key. Without knowledge of the original key which was used to create a set of shares, an adversary cannot forge new shares for a given document.

Share hoarding: Trade until a sufficient fraction of an objectionable document is controlled by a group of collaborating servers, and then destroy this document. Likewise, a sufficiently wealthy adversary could purchase a series of servers with very large drives and join the servnet, trading away garbage for "valuable data." He can trade away enough garbage to have a significant portion of all the data in the servnet on his drives.

Prevention: We rely on the overall size of the servnet to make it unlikely or prohibitively expensive for any given server or group of collaborating servers to obtain a sufficient fraction of the shares of any given document. The failure of this assumption would leave us with no real defense.

4.2 Attacks on the Reputation System

While attacks against the reputation system[2] are related to attacks directly against servers, their goal is not to directly affect document availability or servnet operation. Rather, these attacks seek to compromise the means by which we provide accountability for malicious or otherwise misbehaving servers.

Some of these attacks, such as temporary denials of service, have negative repercussions on the reputation of a server. These repercussions might be qualified as "unfair", but are best considered in the following light: if a server is vulnerable to these attacks, it may not be capable of meeting the specifications of the Free Haven protocol. Such a server is not worthy of trust to meet those specifications. The reputation system does not judge intent, merely actions.

Simple Betrayal: An adversary may become part of the servnet, act correctly long enough to gain a good reputation, then betray this trust by deleting files before their expiration dates.

Prevention: The reputation economy is designed to make this unprofitable. In order to obtain enough "currency" to store data, a server must reliably store data for others. Because a corrupt server must store at least as much data for others as the amount of data it deletes, such an adversary at worst does no overall harm to the system and may even help. This "50% useful work" ratio is a rather loose lower bound — it requires tricking a great number of high-credibility servers into recommending you. A server which engages in this behavior should be caught by the buddy system when it deletes each share.

Buddy Coopting: If a corrupt server (or group of colluding servers) can gain control of both a share and its buddy, it can delete both of them without repercussions.

Prevention: We assume a large quantity of shares in the servnet, making buddy capture more difficult. Servers also can modify their perceived reputation of a server if narrow trading parameters, or constant trading, suggests an attempt to capture buddies. More concretely, a possible work-around involves separating the reply-block addresses for trading and for buddy checking, preventing corrupt servers from acquiring the buddies of the shares they already have. Such an approach adds complexity, and possibly opens other avenues for attack.

False Referrals: An adversary can broadcast false referrals, or even send them only to selected servers.

Prevention: The confidence rating of credibility can provide a guard against false referrals, combined with a single-reporting policy (i.e., at most one referral per target per source is used for reputation calculations).

Trading Receipt Games: While we believe that the signed timestamps attest to who did what and when, receipt-based accountability may be vulnerable to some attacks. Most likely, these will involve multi-server adversaries engaging in coordinated bait-and-switch games with target nodes.

Entrapment: There are several ways in which an adversary can appear to violate the protocols. When another server points them out, the adversary can present receipts which show her wrong and can accuse her of sending false referrals. A more thorough system of attestations and protests is necessary to defend against and account for this type of attack.

4.3 Attacks on Anonymity

There are a number of attacks which might be used to determine more information about the identity of some entity in the system.

[2]Parts of this section were originally written by Brian Sniffen in [43].

Attacks on reader anonymity: An adversary might develop and publish on Free Haven a customized virus which automatically contacts a given host upon execution. A special case of this attack would be to include mime-encoded URLs in a document to exploit reader software which automatically loads URLs. Another approach might be to become a server on both the servnet and the mixnet, and attempt an end-to-end attack, such as correlating message timing with document requests. Indeed, servers could claim to have a document and see who requests it, or simply monitor queries and record the source of each query. Sophisticated servers might attempt to correlate readers based on the material they download, and then try to build statistical profiles and match them to people (outside Free Haven) based on activity and preferences; we prevent this attack by using each reply block for only one transaction.

Attacks on server anonymity: Adversaries might create unusually large shares, and try to reduce the set of known servers who might have the capacity to store such shares. This attacks the partial anonymity of these servers. An adversary could become a server, and then collect routine status and participation information (such as server lists) from other servers. This information might be extended with extensive knowledge of the bandwidth characteristics and limitations of the Internet to map servnet topology. By joining the mixnet, an adversary might correlate message timing with trade requests or reputation broadcasts. An alternate approach is simply to spread a Trojan Horse or worm which looks for Free Haven servers and reports which shares they are currently storing.

Attacks on publisher anonymity: An adversary could become a server and log publishing acts, and then attempt to correlate source or timing. Alternatively, he might look at servers who might recently have published a document, and try to determine who has been communicating with them recently.

There are entirely social attacks which can be very successful, such as offering a large sum of money for information leading to the current location of a given document, server, reader, etc.

We avoid or reduce the threat of many of these attacks by using an anonymous channel which supports pseudonyms for our communications. This prevents most or all adversaries from being able to determine the source or destination of a given message, or establish linkability between each endpoint of a set of messages. Even if server administrators are subpoenaed or otherwise pressured to release information about these entities, they can openly disclaim any knowledge.

5 Related Work

There are a number of projects and papers which discuss anonymous publication services. We start this section by providing an overview of some of the related projects and papers. After this section, we continue by examining the amount of anonymity that each project offers.

5.1 The Eternity Service

This work was inspired by Anderson's seminal paper on The Eternity Service[2]. As Anderson wrote, "[t]he basic idea is to use redundancy and scattering techniques to replicate data across a large set of machines (such as the Internet), and add anonymity mechanisms to drive up the cost of selective service denial attacks."

A publisher uploads a document and some digital cash, along with a requested file duration (cost would be based on document size and desired duration). In the simple design, a publisher would upload the document to 100 servers, and remember ten of these servers for the purposes of auditing their performance. Because he does not record most of the servers to whom he submitted the file, there is no way to identify which of the participating eternity servers are storing his file. Document queries are done via broadcast, and document delivery is achieved through one-way anonymous remailers.

There are issues which are not addressed in his brief paper: for instance, if documents are submitted anonymously but publishers are expected to remember a random sample of servers so they can audit them, what do they do when they find that some server is cheating? Anderson passes this responsibility on to the digital cash itself, so servers do not receive payment if they stop providing the associated service. He does not elaborate on the possible implications of this increased accountability to the anonymity of the publishers.

Eternity has several problems that hinder real-world deployment. Most importantly, Eternity relies on a stable digital cash scheme, which is not available today. There is no consideration to maintaining a dynamic list of available servers and allowing servers to smoothly join and leave. Anderson further proposes that a directory of files in the system should itself be a file in the system. However, without a mechanism for updating or revising files, this would appear very difficult to achieve.

5.2 Napster

The Napster service[36] is a company based around connecting people who are offering MP3 files to people who want to download them. While they provide no real anonymity and disclaim all legal liability, an important thing to note about the Napster service is that it is highly successful. Thousands of people use Napster daily to exchange music; if there were greater security (and comparable ease of use), we believe that many thousands more would participate. The existence of Napster shows that demand exists for a distributed storage and retrieval service.

5.3 Gnutella

Gnutella[15] is a peer-to-peer Napster clone. Developed originally by Nullsoft, it is currently maintained as an open source project. The Gnutella developers claim that querying the network is "anonymous." Analysis of the Gnutella protocol reveals features which make this statement problematic.

The header of a Gnutella packet includes two fields: *TTL* (time to live: the number of additional hops after which the packet should be dropped) and *Hops taken* (the number of hops this packet has made since its creation). The TTL is started at some default value based on the expected size of the network, and the Hops value is effectively an inverse of the TTL during the travel of the packet. Because the Hops value is 1 when the packet is initially sent, it is clear when a server is generating a query.

In addition, while the protocol is designed for a user to set up connections with his "friends", there is no infrastructure in place for finding new friends. Instead, the Gnutella site offers a default set of friends with which users can start. Most users will never change this file if the service is functional. This means that the actual network is a hierarchical system, as shown in pictures of the Gnutella network topology[45]. There are a small number of central nodes which would be ideal targets for collecting information about users and queries.

Moreover, only queries are protected. The actual downloads are done by point-to-point connections, meaning that the IP addresses of server and reader are both revealed to each other. This is done for reasons of efficiency, but it is far from anonymous.

Sites such as the Gnutella Wall of Shame [12], which attempts to entrap child pornographers using the Gnutella service, demonstrate that the direct file-transfer portion of the Gnutella service does not adequately protect the anonymity of servers or readers.

5.4 Eternity USENET

Adam Back proposed[4] a simpler implementation of the Eternity Service, using the existing Usenet infrastructure to distribute the posted files all around the world.

To achieve anonymity in publishing, Eternity Usenet employs cypherpunks type I and type II (mixmaster) remailers as gateways from email to newsgroups. Publishers PGP-sign documents which they wish to publish into the system: these documents are formatted in html, and readers make http search or query requests to 'Eternity Servers' which map these requests into NNTP commands either to a remote news server or a local news spool. With the initial implementation, the default list of newsgroups to read consists only of `alt.anonymous.messages`. The Eternity Server effectively provides an interface to a virtual web filesystem which posters populate via Usenet posts. Eternity Usenet uses normal Usenet mechanisms for retrieval, posting, and expiring, so publishers may not have control over the expiration time or propagation rate of their document.

Reader anonymity for Eternity USENET is provided when the system is used in "local proxy" mode, in which the user downloads the entire eternity newsgroup from a remote server. The server can still link the reader to that day's contents of an eternity newsgroup, so the reader anonymity is not as strong as we might like.

Back treats Usenet as an append-only file system. His system provides support for replacing files (virtual addresses) because newer posts signed with the same PGP key are assumed to be from the same publisher. Addresses are claimed on a first-come first-served basis, and PGP signatures provide linkability between an initial file at a given address and a revision of that file. It is not clear what happens when two addresses are claimed at once – since Usenet posts may arrive out of order, it would seem that there might be some subtle attacks against file coherency if two different Eternity Servers have a different notion of who owns a file.

While the system is not directly 'censorable' as we usually consider it, the term 'eternity' is misleading. Usenet posts expire based on age and size. Back does not provide an analysis of how long a given document will survive in the network. The task of making a feasible distributed store of Eternity documents is left as a future work.

5.5 Freenet

Like Gnutella, Freenet[11] is a peer to peer network of servers. When a user wishes to request a document, she hashes the name of that document (where she gets this name is outside the scope of Freenet) and then queries her own server about the location. If her server does not have it, it passes the query on to a nearby server which is "more likely" to have it. Freenet clusters documents with similar hashes nearby each other, and uses a routing protocol to route queries "downhill" until they arrive at the desired document.

Freenet bases document lifetime on the popularity of the document: frequently requested files get duplicated around the system, whereas infrequently requested files live in only a few places or die out completely. While this is a valid choice for a system that emphasizes availability and efficiency, it precludes certain uses of the system, e.g., the Yugoslav phone book collection project described earlier.

Freenet explicitly sets out to provide anonymity. Their goals include both sender and reader anonymity, as well as plausible deniability for servers – the notion that a server does not know the contents of documents it is storing. They provide this last, which we call passive-server document anonymity, by referencing files by $H(name)$ and having users encrypt the documents themselves with *name* before inserting them. This means that anybody who knows the original *name* string can decrypt the document, but the server storing the document is unable to invert $H(name)$ to determine *name*.

Freenet has a similar potential flaw with publisher and reader anonymity to Gnutella, due to the presence of the TTL and Depth (comparable to Hops) fields in the Freenet message headers. Freenet takes steps to avoid the problems of Gnutella's Depth and TTL headers by randomly assigning values to both fields, so that a depth of 1 does not necessarily mean that a request originated with a given node. Packets with TTL 1 are randomly either expired or forwarded onward.

Document requests are also sent through the caching-enabled network (rather than peer-to-peer as they are in Gnutella). Because of these measures, Freenet seems to provide 'more' anonymity than

Gnutella.

Further, statistical attacks similar to those described in the Crowds [41] paper might work to pinpoint the location of a given reader or publisher; caching provides protection against this since the network topology for a given document changes after each request. These attacks need to be analyzed further.

Freenet makes files highly accessible and offers some level of anonymity. But since the choice to drop a file is a purely local decision, and since files that aren't requested for some time tend to disappear automatically, it can't guarantee a specified lifetime for a document. We expect that Freenet will provide a very convenient service for porn and popular audio files, but anything less popular will be driven off the system.

5.6 Mojo Nation

Mojo Nation[23] is another peer-to-peer design for robustly distributing resources. The basic operations it supports are publishing and retrieving, but it differs from other works because it employs a digital cash system to help protect against abuse of the system.

In Mojo Nation, a user who wishes to publish a document (call her Alice) uses error correction techniques to split the document into eight pieces, any four of which are sufficient to reconstruct. She then combines hashes of these eight pieces into a second document called a *sharemap*, and proceeds to do the eight-way splitting on this sharemap as well. She sends descriptions of the eight pieces of the sharemap to a separate agent called a *content tracker*, which is responsible for keeping track of how to reconstruct each document.

Other participants in the system serve as *block servers*. They offer storage on their machine to the system. Each block server has a bitmask which describes the subset of 'hash space' (hashes of a piece of a document, that is) that it will store. For each piece of her document, Alice pays the appropriate block server to store that piece. Alice learns about the set of block servers available and interested in her pieces through yet another agent called a *metatracker*. Multiple block servers overlapping on which bitmasks they cover allow for greater redundancy. Alice informs the *publication tracker* when she has published a document, and then other block servers might go to the block servers to which she published and purchase those document pieces.

To retrieve a document, Bob queries the content tracker and receives information about the eight pieces that will reconstruct the sharemap for that document. He asks the metatracker which block servers serve the address space for those pieces, and then purchases them from the appropriate block servers. He then reconstructs the sharemap, and from there repeats the process with the eight pieces of the document he is retrieving. Because of the error correction codes, Bob actually only needs to purchase any four of the pieces for each reconstruction phase.

As in Freenet, document pieces expire based entirely on choices local to the block server. That is, in most cases the most popular files will stay in the system, and the unpopular files will be dropped.

The entire system works based on currency called *mojo*. Participants in the system 'pay' mojo to other participants when they ask for a service that uses resources. In this way, Mojo Nation reduces the potential for damage from resource flooding attacks. A credit and reputation system allows the interactions to be streamlined based on trust built up from past experience.

Mojo Nation employs a centralized bank server to handle Mojo transactions and accounting. It's also not clear that the job of the metatracker can be done in a decentralized way (that is, without producing a bottleneck either because loss of the metatracker implies loss of that service, or because there's no way to smoothly inform participants of metatrackers joining and leaving the system). A good distributed (truly decentralized) anonymous electronic cash system would be much more useful, but as far as we know there still isn't one available.

The goals of Mojo Nation are not anonymity. Rather, they want to be a ubiquitous efficient distributed file distribution system which focuses on document accessibility. It is not yet clear how robust the overall system will be, but the design certainly appears to scale well.

5.7 Publius

Publius[49] attacks the problem of anonymous publishing from a different angle, employing a one-way anonymous channel to transfer documents from publishers to servers. The Publius protocol is designed to maintain availability of documents on these servers.

In this system, a publisher generates a key K for her document, and encrypts the document with this key. She performs Shamir's secret-sharing algorithm to build a set of n key shares, any k of which is sufficient to reconstruct K. From there, she chooses some n of the Publius servers and anonymously delivers the encrypted message plus one share to each of these n servers.

In this way, the document is replicated over each server, but the key is split over the n servers. Document reading is implemented by running a local web proxy on the reader's machine; the n addresses chosen as servers are concatenated into a URL which is presumably published or otherwise remembered. The local proxy fetches each share independently, reconstructs the original key K, and then decrypts the document.

The Publius system provides publisher anonymity by means of a one-way anonymous channel between authors and servers. In addition, because Shamir's secret-sharing protocol is used and each server only receives one share, Publius provides both computational and information-theoretic passive-server document anonymity: a single server is not able to determine anything about a document it stores.

A minor flaw is that readers cannot determine if a share is corrupt simply by examining it: the reader must request all of the shares and attempt to reconstruct in order to determine the integrity of a share. A verifiable secret sharing scheme [44] might make the system more efficient.

Publius provides no smooth decentralized support for adding new servers and excising dead or malicious servers. More importantly, Publius provides no accountability – there is no way to prevent publishers from entirely filling the system with garbage data.

6 An Analysis of Anonymity

We describe the protections offered for each of the broad categories of anonymity. In Table 1, we provide an overview view of Free Haven and the different publishing systems which we examined. We consider the level of privacy provided – computational (C) and perfect-forward (P-F) anonymity – by the various systems.

Computational anonymity means that an adversary modelled as a polynomial-time Turing Machine has no better than a $\frac{1}{2} + neg(k)$ chance of breaking anonymity, for some reasonable security parameter k and negliglible function $neg(k)$. Perfect forward anonymity is analogous to perfect forward secrecy: a system is perfect forward anonymous if no information remains after a transaction is complete which could later identify the participants if one side or the other is compromised. This notion is a little bit trickier – think of it from the perspective of an adversary watching the user over a long period of time. Is there anything that the adversary can discover from watching several transactions that he can't discover from watching a single transaction?

Free Haven provides computational and perfect forward author anonymity, because authors communicate to publishers via an anonymous channel. Servers trade to other servers via pseudonyms, providing computational but not perfect forward anonymity, as the pseudonyms can be broken later. Because trading is constant, however, Free Haven achieves publisher anonymity for publishers trying to trade away all shares of the same document. The use of IDA to split documents provides passive-server document anonymity, but the public key embedded in each share (which we require for integrity checking) makes it trivial for active servers to discover what they are storing. Because requests are broadcast via an anonymous channel, Free Haven provides computational reader anonymity, and different reply blocks used and then destroyed after each request provide perfect forward reader anonymity.

Gnutella fails to provide publisher anonymity, reader anonymity, or server anonymity because of the

Project	Publisher		Reader		Server		Document	Query
	C	P-F	C	P-F	C	P-F	C	C
Gnutella								
Eternity Usenet	+	+	?				+	
Freenet	+	+	?				+	
Mojo Nation	?	?					+	
Publius	+	+					+	
Free Haven	+	+	+	+	+		+	

Table 1: Anonymity Properties of Publishing Systems

peer-to-peer connections for actual file transfer. Because Gnutella servers start out knowing the intended contents of the document they are offering, they also fail to provide document anonymity.

Eternity Usenet provides publisher anonymity via the use of one-way anonymous remailers. Server anonymity is not provided, because every Usenet server which carries the eternity newsgroup is a server. Adam Back has pointed out that passive-server document anonymity can be provided by encrypting files with a key derived from the URL; active servers might find the key and attempt to decrypt stored documents. Reader anonymity is not provided by open public proxies unless the reader uses an anonymous channel because the proxy can see the content and timing of a user's queries and downloads. For local proxies, which connect to a separate news server, however, the situation is better because the news server knows only what the user downloads. Even so, this is not quite satisfactory, because the user can be tied by the server to the contents of the eternity newsgroup at a certain time.

Freenet achieves passive-server document anonymity because servers are unable to reverse the hash of the document name to determine the key with which to decrypt the document. For active-server document anonymity, the servers can check whether they are carrying a particular key, but cannot easily match a stored document to a key due to the hash function. Server anonymity is not provided because given a document key, it is very easy to locate a server that is carrying that document – querying any server at all will result in that server carrying the document! Because of the TTL and Hops fields for both reading and publishing, it is also not clear that Freenet achieves publisher or reader anonymity, although they are much better in these regards than Gnutella. We note that the most recent Freenet design introduces randomized TTL and Hops fields in each request, and plans are in the works to allow a publish or retrieve operation to traverse a mixnet chain before entering the Freenet system. These protections will make attacks based on tracking queries much more difficult.

Mojo Nation achieves passive-server document anonymity, because the server holding a share doesn't know how to reconstruct that document. The Mojo Nation design is amenable to integrating publisher anonymity down the road – a publisher can increase his anonymity by paying more Mojo and chaining requests through participants that act as 'relays'. The specifics of prepaying the path through the relays are not currently being designed. It seems possible that this technique could be used to ensure reader anonymity as well, but the payment issues are even more complex. Indeed, the supplied digital cash model is not even anonymous currently; users need to uncomment a few lines in the source, and this action breaks Chaum's patents.

Publius achieves document anonymity because the key is split between the n servers, and without sufficient shares of the key a server is unable to decrypt the document that it stores. The secret sharing algorithm provides a stronger form of this anonymity (albeit in a storage-intensive manner), since a passive server really can learn nothing at all about the contents of a document that it is helping to store. Because documents are published to Publius through a one-way anonymous remailer, it provides publisher anonymity. Publius provides no support for protecting readers by itself, however, and the servers containing a given file are clearly marked in the URL used for retrieving that file. Readers can use a system such as ZKS Freedom or Onion Routing to protect themselves, but servers may still be liable for storing “bad” data.

We see that systems can often provide publisher anonymity via one-way communication channels, effectively removing any linkability; removing the need for a reply block on the anonymous channel means that there is “nothing to crack”. The idea of employing a common mixnet as a communications channel for each of these publication systems is very appealing. This would mean that we could leave most of the anonymity concerns to the communication channel itself, and provide a simple back-end file system or equivalent service to transfer documents between agents. Thus the design of the back-end system could be based primarily on addressing other issues such as availability of documents, protections against flooding and denial of service attacks, and accountability in the face of this anonymity.

7 Future Work

Our experience designing Free Haven revealed several problems which have no simple solutions; further research is required. We state some of these problems here and refer to the first author's thesis[13] for in-depth consideration.

Deployed Free Low-Latency Pseudonymous Channel: Free Haven requires pseudonyms in order to create server reputations. The only current widely deployed channels which support pseudonyms seem to be the Cypherpunk remailer network[34] and ZKS Freedom mail. The Cypherpunk and ZKS version 1 networks run over SMTP and consequently have high latency. This high latency complicates protocol design. The recently announced version 2 of ZKS Freedom mail runs over POP and may offer more opportunity for the kind of channel we desire.

Accountability and Reputation: We found it extremely difficult to reason about the accountability in Free Haven, especially when considering the “buddy system.” At the same time, accountability is critical to ensuring that documents remain available in the system. Future work in this area might develop an “anonymous system reputation algebra” for formally reasoning about a server's reliability based on various circumstances – this would allow us to verify trust protocols. We sketch this problem in more detail in [14].

Modelling and Metrics: When desiging Free Haven, we made some choices, such as the choice to include trading, based only on our intuition of what would make a robust, anonymous system. A mathematical model of anonymous storage would allow us to test this intuition and run simulations. We also need *metrics*: specific quantities which can be measured and compared to determine which designs are “better.” For example, we might ask “how many servers must be compromised by an adversary for how long before any document's availability is compromised? before a specific targeted document's availability is compromised?” or “how many servers must be compromised by an adversary for how long before the adversary can link a document and a publisher?” This modelling might follow from the work of Gulcu and Tsudik[22], Kesdogan, Egner, and Bschkes[28], and Berthold, Federrath, and Kohntopp[6] which apply statistical modelling to mix-nets.

Formal Definition of Anonymity: Closely related to the last point is the need to formalize the “kinds of anonymity” presented in section 2. By formally defining anonymity, we can move closer to providing meaningful *proofs* that a particular system provides the anonymity we desire. We might leverage our experience with cryptographic definitions of semantic security and non-malleability to produce similar definitions and proofs[21]. A first step in this direction might be to carefully explore the connection remarked by Rackoff and Simon between secure multiparty computation and anonymous protocols[42].

Usability Requirements and Interface: We stated in the introduction that we began the Free Haven Project out of concern for the rights of political dissidents. Unfortunately, at this stage of the project, we have contacted few political dissidents, and as a consequence do not have a clear idea of the usability and interface requirements for an anonymous storage system. Our concern is heightened by a recent paper which points out serious deficiencies in PGP's user interface [50].

Efficiency: It seems like nearly everyone is doing a peer-to-peer system or WWW replacement these days. Which one will win? Adam Back pointed out[5] that in many cases, the efficiency and perceived benefit of the system is more important to an end user than its anonymity properties. This is a major problem with the current Free Haven design: we emphasize a quality relatively few potential users care about at the expense of something nearly everyone cares about. Is there a way to create an anonymous system with a tolerable loss of perceived efficiency compared to its non-anonymous counterpart? And what does "tolerable" mean, exactly?

We consider the above to be challenge problems for anonymous publication and storage systems.

8 Conclusion

Free Haven is a decentralized storage service which provides anonymity to publishers, readers, and servers, provides a dynamic network, and ensures the availability of each document for a publisher-specified lifetime. None of these requirements is new by itself, but Free Haven addresses all of them at once.

The current Free Haven design is unsuitable for wide deployment, because of several remaining problems. The primary problem is efficiency. An inefficient design will lead to a system with few users. A system with few users will not provide the anonymity we desire.

Free Haven uses inefficient broadcasts for communication. One way to address this problem is by coupling Free Haven with a widely-deployed efficient file sharing service such as Freenet or Mojo Nation. Popular files will be highly accessible from within the faster service; Free Haven answers queries for less popular documents which have expired in this service.

Filling this role requires facing problems particular to a long-term persistent storage service. Without the requirement of long-term persistent storage, strong accountability measures are less necessary. Without these measures, computational overhead can be greatly lowered, making unnecessary many communications that are used to manage reputation metrics. And without the requirement for such anonymity and the resulting latency from the communications channel, readers could enjoy much faster document retrieval. Solving each of these problems is important: even if Free Haven is not the utility of first resort, it must respond to requests in a timely and reliable manner.

These problems are far from being solved. Until the risks involved in using such systems can be better evaluated, they cannot be used in good conscience for situations where failure is not an option. Much more work remains.

Acknowledgements

Professor Ronald Rivest provided invaluable assistance as Roger's Masters and Michael's Bachelors thesis advisor and caused us to think hard about our design decisions. Professor Michael Mitzenmacher made possible David's involvement in this project and provided insightful comments on information dispersal and trading. Beyond many suggestions for overall design details, Brian Sniffen provided the background for the reputation system, and Joseph Sokol-Margolis was useful for considering attacks on the system. Andy Oram was instrumental in helping to restructure the paper to improve flow and clarity. Adam Back and Theodore Hong commented on our assessment of their systems and made our related work section much better. Wei Dai caught a very embarrassing error in our description of signature schemes, for which we thank him. Furthermore, we thank Susan Born, Nathan Mahn, Jean-François Raymond, Anna Lysyanskaya, Adam Smith, and Brett Woolsridge, for further insight and feedback.

References

[1] Masayuki Abe. Universally verifiable mix-net with verification work independent of the number of servers. In *Advances in Cryptology – EUROCRYPT '98*, pages 437–447.

[2] Ross Anderson. The Eternity Service. http://www.cl.cam.ac.uk/users/rja14/eternity/eternity.html.

[3] Aol instant messenger. http://www.aol.com/aim.

[4] Adam Back. The Eternity Service. http://phrack.infonexus.com/search.phtml?view&article=p51-12.

[5] Adam Back. Re: another distributed project. http://freehaven.net/archives/freehaven/dev/Aug-2000/msg00027.html.

[6] Oliver Berthold, Hannes Federrath, and Marit Kohntopp. Anonymity and unobservability on the Internet. In *Workshop on Freedom and Privacy by Design: CFP 2000*, 2000.

[7] Ran Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. Deniable encryption. In *Advances in Cryptology – CRYPTO '97*.

[8] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1982.

[9] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1:65–75, 1988.

[10] Yuan Chen, Jan Edler, Andrew Goldberg, Allan Gottlieb, Sumeet Sobti, and Peter Yianilos. A prototype implementation of archival intermemory. In *Proceedings of the fourth ACM Conference on Digital libraries (DL '99)*, 1999.

[11] Ian Clarke. The Free Network Project. http://freenet.sourceforge.net/.

[12] The Cleaner. Gnutella wall of shame. http://www.zeropaid.com/busted/.

[13] Roger Dingledine. The Free Haven Project. Master's thesis, MIT, 2000.

[14] Roger Dingledine, Michael J. Freedman, and David Molnar. Accountability. In *Peer-to-peer*. O'Reilly and Associates, 2001.

[15] Ian Hall-Beyer *et. al.* Gnutella. http://gnutella.wego.com/.

[16] Michael J. Freedman. Design and Analysis of an Anonymous Communication Channel for the Free Haven Project. http://theory.lcs.mit.edu/~cis/cis-theses.html, May 2000.

[17] Electronic Frontiers Georgia (EFGA). Anonymous remailer information. http://anon.efga.org/Remailers/.

[18] Ian Goldberg and Adam Shostack. Freedom network 1.0 architecture, November 1999.

[19] Ian Goldberg, David Wagner, and Eric Brewer. Privacy-enhancing technologies for the internet. In *Proceedings of IEEE COMPCON '97*.

[20] O. Goldreich, S. Even, and Lempel. A randomized protocol for signing contracts. In *Advances in Cryptology – CRYPTO '82*.

[21] Oded Goldreich. *Modern Cryptography, Probabilistic Proofs, and Pseudo-Randomness.* Springer-Verlag, 1999.

[22] C. Gulcu and G. Tsudik. Mixing e-mail with Babel. In *Proceedings of the ISOC Symposium on Network and Distributed System Security*, pages 2–16, 1996.

[23] Autonomous Zone Industries. Mojonation. http://www.mojonation.com/.

[24] M. Jakobsson. Flash mixing. In *Principles of Distributed Computing PODC '99.*

[25] M. Jakobsson. A practical mix. In *Advances in Cryptology – EUROCRYPT '98.*

[26] Ari Juels and John Brainard. Client puzzles: A cryptographic defense against connection depletion attacks. In *Proceedings of the 1999 Network and Distributed System Security Symposium*, February 1999.

[27] Clifford Kahn, David Black, and Paul Dale. MANET: Mobile agents for network trust. http://www.darpa.mil/ito/psum1998/F255-0.html, 1998.

[28] Dogan Kesdogan, Jan Egner, and Roland Buschkes. Stop and go mixes: Providing probabilistic anonymity in an open system. In *1998 Information Hiding Workshop*, pages 83–98.

[29] Raph Levien. Advogato's trust metric. http://www.advogato.org/trust-metric.html.

[30] Mark Lewis. Metallica sues Napster, universities, citing copyright infringement and RICO violations. http://www.livedaily.com/archive/2000/2k04/wk2/MetallicaSuesNapster,Univ.html.

[31] Tal Malkin. *Private Information Retrieval.* PhD thesis, MIT. see http://theory.lcs.mit.edu/ cis/cis-theses.html.

[32] David Michael Martin. PhD thesis, Boston University, 2000. http://www.cs.du.edu/˜dm/anon.html.

[33] Tim May. Cyphernomicon. http://www2.pro-ns.net/ crypto/cyphernomicon.html.

[34] David Mazieres and M. Frans Kaashoek. The design and operation of an e-mail pseudonym server. In *5th ACM Conference on Computer and Communications Security*, 1998.

[35] S. Micali. Certified e-mail with invisible post-offices. In *Talk at RSA '97.*

[36] Napster. http://www.napster.com/.

[37] University of Michigan News and Information Services. Yugoslav phone books: perhaps the last record of a people. http://www.umich.edu/˜newsinfo/Releases/2000/Jan00/r012000e.html.

[38] A. Pfitzmann, B. Pfitzmann, and M. Waidner. ISDN-Mixes : Untraceable communication with small bandwidth overhead. In *GI/ITG Conference: Communication in Distributed Systems*, pages 451–463. Springer-Verlag, 1991.

[39] PGP FAQ. http://www.faqs.org/faqs/pgp-faq/.

[40] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance, April 1989.

[41] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *DIMACS Technical Report*, 97(15), April 1997.

[42] Simon and Rackoff. Cryptographic defense against traffic analysis. In *STOC 1993*, pages 672–681, 1993.

[43] Brian T. Sniffen. Trust Economies in the Free Haven Project. http://theory.lcs.mit.edu/~cis/cis-theses.html, May 2000.

[44] Markus Stadler. Publicly verifiable secret sharing. In *EUROCRYPT '96*, 1996. http://citeseer.nj.nec.com/stadler96publicly.html.

[45] Steve Steinberg. Gnutellanet maps. http://gnutella.wego.com/file_depot/0-10000000/110000-120000/116705/folder/151713/network3.jpg.

[46] Paul Syverson and Stuart Stubblebine. Group principals and the formalization of anonymity. In *World Congress on Formal Methods 1999*, 1999.

[47] P.F. Syverson, D.M. Goldschlag, and M.G. Reed. Anonymous connections and onion routing. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, May 1997.

[48] Vernor Vinge. True Names. Short story.

[49] Marc Waldman, Aviel Rubin, and Lorrie Cranor. Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system.

[50] Alma Whitten and J.D. Tygar. Why johnny can't encrypt. In *USENIX Security 1999*, 1999. http://www.usenix.org/publications/library/proceedings/sec99/whitten.html.