Free Haven

November 29, 2000

The Free Haven Project is dedicated to designing a system of anonymous storage that resists the attempts of powerful adversaries to find or destroy any stored data. Our goals include:

- Anonymity for all parties: the publishers that insert documents, the readers that retrieve documents, and the servers that store documents.
- Persistence. The publisher of a document determines its lifetime – not the servers holding the document.
- Flexibility. The system functions smoothly as servers are added or remove themselves.
- Accountability. We apply a reputation system to servers that attempts to limit the damage done by those that misbehave.

In this chapter, we'll show how Free Haven tries to meet these goals. We spend a particularly large amount of time on anonymity. It is not adequate to speak of "anonymity" as a monolithic concept. In section 2, we'll enumerate the many different *kinds* of anonymity that are important to protect participants in the system.

Free Haven differs from the other projects in this book in the wide range of difficult goals we have taken on. We try to assure anonymity, server accountability, and persistent storage for data independent of its popularity, all at the same time. Here are some comparisons to other projects:

Gnutella The strength of Gnutella is its extremely flexible network design. But when a search is performed, servers respond with an external IP or URL where the user can download the document. Since this actual retrieval is done without any privacy protection, using Gnutella is not a good choice if publishers or readers want anonymity. Further, documents in the Gnutella network last only as long as their host server; when that user logs out for the night, all of his files leave with him.

Freenet and Mojo Nation These systems make files highly accessible and offer 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, these systems don't guarantee a specified lifetime for a document. Indeed, 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.

Publius This project is closest to ours, because it addresses file storage rather than easy accessibility. But 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.

Currently, Free Haven sacrifices efficiency and convenience to achieve its design requirements. Free Haven is designed more for anonymity and persistence of documents than for frequent querying. We expect that interesting material will be retrieved from the system and published in a more accessible fashion (such as in Freenet or normal web pages). Then the document in Free Haven would only need to be accessed if the other sources were shut down, or the reader required stronger anonymity. For more discussion of such "gatewaying" issues, we refer to Chapter 19, *Gateways*.

1 Privacy in Data Sharing Systems

Privacy is a term with positive connotations that every person can appreciate. One key way to achieve privacy, however—anonymity—is widely misunderstood both in daily life and in computer networking. The media and politicians stress socially disapproved activities (such as the exchange of unauthorized music files or erotic pictures involving children) and ignore the important contributions that anonymity provides. Anonymity is used on an everyday basis in forums ranging from radio shows to Usenet newsgroups, by people who suffer from child abuse, drug dependency, or other social problems.

Anonymous publication and storage services allow individuals to speak freely without fear of persecution. Political dissidents must publish their views in order to reach enough people for their criticisms of a regime to be effective, yet they and their readers require anonymity at the same time. 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.

Developers and potential users of other peer-to-peer systems should be interested in the techniques we are developing to preserve anonymity in Free Haven, because they may prove useful in protecting the privacy of users in other systems as well. Many people would like to participate in communities and share information of many kinds without revealing who they are. Their reasons may range from the trivial—such as avoiding spam—to deep social concerns. It is time to face these concerns directly so solutions can be designed fundamentally into peer-to-peer systems.

Peer-to-peer systems which attempt to address anonymity are just starting to be deployed, and the exact requirements and design choices are not yet clear. Recent events have highlighted some shortcomings of current systems. For instance, the limitations of Gnutella were dramatized by the Gnutella Wall of Shame, where someone lured readers to a Web site by claiming to offer child pornography and then published each visitor's IP address. While Napster allowed people with MP3 files to find each other, it also made it easy for the band Metallica to find people who were offering unauthorized copies of Metallica songs and force them off the system.

These shortcomings cause people to look toward a new generation of anonymous publication services that address anonymity. In developing Free Haven, we hope to clarify some of the requirements for such systems and highlight the design choices.

1.1 Reliability with anonymity

In the physical world, people use safety deposit boxes to protect valuable items. Everything from passports to house titles to krugerrands – if it's important, it goes in the box, which is kept at the local bank. The bank has armed guards, smiling tellers, and a history going back to the Knights Templar. Now suppose someone suggested to you that instead of going to the bank, it would be a better idea to hand your gold bars to the next guy on the street and ask him to "just hold these for a bit." You'd look at a person with such notions as though he had three heads...yet in some sense, this is exactly what distributed peer-to-peer file sharing systems like Free Haven ask you to do.

The critical point is that for a safety deposit box, the only thing which really matters is reliability and availability: can you get your items when you want them? The rest is irrelevant. If the guy on the street could guarantee that you'll get your gold back and follow through, he would be "just as good" as the bank. In fact, if you're interested in protecting your privacy, the guy on the street may be better – he doesn't know or care who you really are. Of course, in the physical world, it's still a bad idea to give gold bars to random people on the street. Online, however, cryptography allows things to work out differently.

Many systems in addition to Free Haven need reliability, particularly peer-to-peer systems that ask people to share resources. When offering and retrieving resources, users want to preserve their privacy. When evaluating whether to transfer custody of their resources to another party on the system, users want to know whether that party can be trusted.

Initially, these goals seem mutually exclusive, but the solution is to allow users to have *pseudonyms*, and to assign a *reputation* to each pseudonym. Free Haven differs from other systems in that the servers in the Free Haven system are known only by their pseudonyms, and we provide an automated system to track reputations (honesty and performance) for each server. A server's reputation influences how much data it can store in Free Haven and provides an incentive to act correctly. Reputation can be a complex matter — just think of all the reader reviews and "People also bought..." ratings on the Amazon.com retail site — so we'll leave a discussion to Chapter 17, *Reputation*. Establishing trust through the use of pseudonyms is covered in Chapter 15, *Trust*.

What lets a malicious adversary find a person in real life? One way is to know his or her *true name*, a term first used by fiction author Vinge [13] and popularized by May[6]. The true name is the legal identity of an individual and can be used to find an address or other real-life connection. Obviously, a pseudonym should not be traceable to a true name.

As an author can use a pseudonym to protect his or her true name, in a computerized storage system a user can employ a pseudonym to protect another form of identity called *location*. This is an IP address or some other aspect of the person's physical connection to the computer system. In a successful system, a pseudonym always reflects the activities of one particular entity–but no one can learn the true name or location of the entity. The ability to link many different activities to a pseudonym is the key to supporting reputations.

2 Anonymity for Anonymous Storage

The word "anonymous" can mean many different things. Indeed, some systems claim "anonymity" without specifying a precise definition. This introduces a great deal of confusion when users are trying to evaluate and compare publishing systems to understand what protections they can expect from each system.

A publishing situation creates many types of anonymity—many requirements that a system has to meet in order to protect the privacy of both content providers and users. Here, we'll define the *author* of a document as whoever initially created it. The author may be the same or different from the *publisher*, who places the document into Free Haven or another storage system. Documents may have *readers*, who retrieve the document from the system. And many systems, including Free Haven, have *servers*, who provide the resources for the system, such as disk space and bandwidth.

Free Haven tries to make sure that no one can trace a document back to one of these people—or trace one of them forward to a document. In addition, we want to prevent adversaries who are watching both a user and a document from learning anything that might convince them that the user is connected to that document. Learning some information that might imply a connection allows "linking" the user to that action or document.

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.

Isolated-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 – Freenet takes this approach. Mojo Nation takes a different approach to get the same end: it uses a "two-layer" publishing system, in which documents are split up into shares, and then a separate 'share map' is similarly split and distributed to participants called *content trackers*. In this way, servers holding shares of a document cannot easily locate the share map for that document, so they cannot determine which document it is.

Connected-server document-anonymity refers to the situation in which the server is allowed to communicate and compare data with all other servers. Since a connected server may act as a reader and do document requests itself, connected-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 can not determine which document it is serving when satisfying a reader's request. 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'.

2.1 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.

3 The Design of Free Haven

Free Haven offers a community of servers called the *servnet*. Despite the name, all servers count the "same", and within the servnet Free Haven is a peer-to-peer system. There are no "clients" in the old client-server sense; the closest approximation are users looking for files and potential publishers. Users query the entire servnet at once, not any single server in particular. Potential publishers do convince a single server to publish a document, but the actual publishing of a document is done by a server itself in a peer-to-peer fashion.

All of these entities – servnet node, document requestor, and publisher – make up the Free Haven players. Thanks to pseudonymity, nobody knows where any server is located – including the one they use as their entry point to the system. Users query the system via broadcast.

Servers don't have to accept just any document that publishers upload to them. That would permit selfish or malicious people to fill up the available disk space. Instead, servers form contracts to store each other's material for a certain period of time.

Successfully fulfilling a contract increases a server's reputation and thus its ability to store some of its own data on other servers. This gives an incentive for each server to behave well, as long as cheating servers can be identified, which we illustrate in section 3.9. In section 3.11 we discuss the reputation system, which is the system that keeps track of trust in each server.

Some of these contracts are formed when a user inserts new data into the servnet through a server she operates. Most of them, however, are formed when two servers swap parts of documents (shares) by *trading*. Trading allows the servnet to be *dynamic* in the sense that servers can join and leave easily and without special treatment. To join, a server starts building up a reputation by storing shares for others – we provide a system where certain servers can act as *introducers* in order to smoothly add new servers. 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 3.7.

The following sections explain how the design of Free Haven allows it to accomplish its goals. Section 3.1 describes the design of the Free Haven system and the operations that it supports, including the insertion and retrieval of documents. We describe some potential attacks 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 5, 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.

3.1 Elements of the System

This chapter focuses on Free Haven's publication system, which is responsible for storing and serving documents. Free Haven also has a communications channel, which is responsible for providing confidential and anonymous communications between parties. Since this communications channel is implemented using pre-existing systems that are fairly well-known in the privacy community, we won't discuss it here. On the other hand, the currently available systems are largely insufficient for our accountability requirements; see Chapter 16, *Accountability*.

The agents in our publication system are the author, the server, and the reader. As we stated in section 2, authors are agents that produce documents and wish to store them in the service, servers are computers that store data for authors, and readers are people who retrieve documents from the service.

These agents know each other only by their pseudonyms and communicate only using the secure communications channel. Currently, the pseudonyms are provided by the Cypherpunks remailer network [7], and the communications channel consists of remailer reply blocks provided by that network. 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 that contains the public keys and reply blocks of other servers in the servnet.

As we said in section 3, documents are split into pieces and stored on different servers; each piece of a document is called a *share*. Unlike Publius or Freenet, servers in Free Haven give up something (disk space) and get other servers' disk space in return. In other words, you earn the right to store your data on the rest of the servnet after you offer to store data provided by the rest of the servnet.

The servnet is dynamic: shares move from one server to another every so often, based on each server's trust of the others. 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.

Authors 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, servers check whether 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.

3.2 Storage

When an author (call her Alice) wishes to store a new document in Free Haven, she must first identify a Free Haven server that's 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.

3.3 Publication

To introduce a file F into the servnet, the publishing server first splits it into shares. Like the Publius algorithm described in Chapter 9, *Publius*, we use an algorithm that creates a large number (n) of shares, but allows the complete document to be recreated using a smaller number (k) of those shares. We use Rabin's information dispersal algorithm (IDA) [9] to break the file into shares $f_1, \ldots, f_n$. (For any integer i, the notation f_i indicates share i of document f.)

The server then generates a key pair (PK_{doc}, SK_{doc}), constructs and signs a data segment for each share, and inserts these shares into its local server space. Attributes in each share include a timestamp, expiration information, $h(PK_{doc})$ (a message digest or hash of the public key from the keypair[1]), 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, see the first author's thesis [2].

Designing a protocol that encourages content-neutrality may well mean that server administrators are less liable for the content on the network. The inspiration here is common carrier law, wherein phone companies are not responsible for the content they carry over their systems.

[1] Chapter 15, *Trust*, describes the purpose of message digests. Briefly, the digest of any data item can be used to prove that the data item is correct. However, no one can regenerate the data item from the digest, so the data item itself remains private to its owner.

3.4 Retrieval

Documents in Free Haven are indexed by the public key PK_{doc} 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 servnet server broadcasts a request containing a message digest or hash of the document's public key, $H(PK_{doc})$, 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.

3.5 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" [8]; 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.

3.6 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 a hash of it 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.

One of the most serious arguments against revocation was raised by Ross Anderson [1]. 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.

3.7 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.

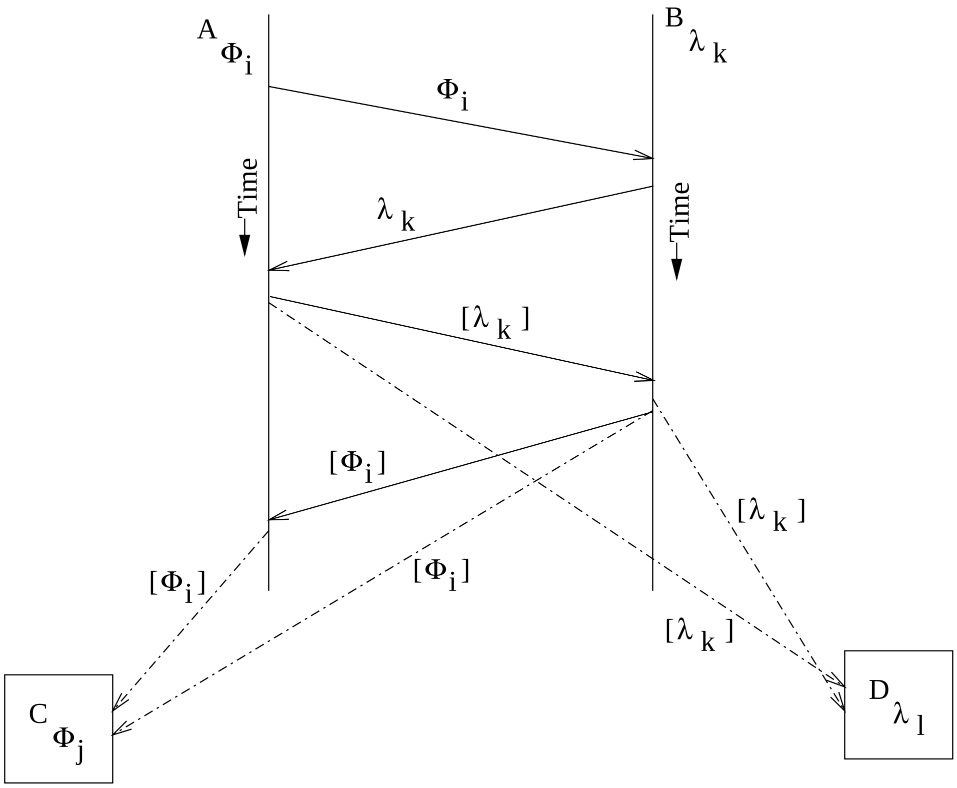

Figure 1: Trade handshake timing diagram

The frequency of trading should be a parameter set by the server operator. When a server A wants to make a trade, it chooses another server B from its list of known servers (based on reputation), and offers a share x and a request for size or duration of a return share. If B is interested, it responds with a share y of its 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 3.8) 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 3.9. 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, A makes a commitment to store the share y. Similarly, the receipt that B generates on the fourth round represents a commitment to store the share x. B could cheat A by failing to continue the protocol after the third step: in this case, A has committed to keeping the share from B, but B has not committed to anything. At this point, A's only recourse is to broadcast a complaint against B and hope that the reputation system causes others to recognize that B has misbehaved. The alternative is to use a fair exchange protocol, which is unreasonably communications-intensive without a trusted third party.

When A trades a share to a server B, A should keep a copy of the share around for a while, just in case B 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.

A diagram describing a trade is given in Figure 1. In this diagram, server A starts out in possession of share Φ_i – that is, share i of document Φ – and server B starts out in possession of document λ_k. In this case, server C has share j of document Φ, and server D has share l of document λ. i and j are buddies, and k and l are buddies.

3.8 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 A. If B (or any other server) has to broadcast a complaint about the way A handled the transaction, furnishing this receipt along with the complaint will provide some rudimentary level of "proof" that B is not fabricating its 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.

3.9 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've designed a "buddy system" that creates an association between two shares within 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.[2]

Periodically, a server holding a given share should query for its buddy, to make sure its buddy is still alive. Should the server that is supposed to contain its buddy stop 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 3.11.

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, 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 comes in, the forwarder is checked and the notification is forwarded if 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 threshold of its shares have 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.

3.10 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. This system is described in fairly non-technical terminology in Chapter 12, *Mixmaster*.

Other examples of anonymous communication channels are Onion Routing [12] and Zero Knowledge Systems' Freedom [3]. David Martin's thesis offers a comprehensive overview of anonymous channels in theory and practice [5].

The first implementation of the Free Haven design will use the Cypherpunk and Mixmaster remailers as its anonymous channel.

[2]More precisely, both the server it's moving from and the server it's moving to notify the buddy, as described in section 3.7.

3.11 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. The history and issues to consider when developing a reputation system can be found in much more detail in Chapter 16, *Accountability*.

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 an astoundingly 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.

3.12 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.

3.13 Implementation Status

The Free Haven Project 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, allowing 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 [2].

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. Because in real life, 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 isolated-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 bully 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 or other protections described in Chapter 16, *Accountability.* 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 is signed 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, subject to deletion.

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[3] 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.

[3] Parts of this section were originally written by Brian Sniffen in [11].

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.

A server which engages in this behavior should be caught by the buddy system when it deletes each share.

Buddy Co-opting: 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 reputation ratings if precise 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 servers.

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.

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 combined 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 An Analysis of Anonymity

We describe the protections offered for each of the broad categories of anonymity. In Table 1, we provide an overview 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 with "reasonable" computing resources and knowledge is unable to break the anonymity involved. The adversary may do anything it likes to try to break the system, but is limited in how much power it has; for example, it may not be able to break the cryptography involved in building a system or be able to break into the computers of every single machine running the system.

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?

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

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 isolated-server document anonymity, but the public key embedded in each share (which we require for authenticating buddy messages) makes it trivial for connected 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 peer-to-peer connections for actual file transfer. Because Gnutella servers start out knowing the intended contents of each 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 isolated-server document anonymity can be provided by encrypting files with a key derived from the URL; connected 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 what a user queries or downloads, and at what time. 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 isolated-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 connected-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 which will make attacks based on tracking queries much more difficult.

Mojo Nation achieves document-anonymity, as described earlier, 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 is stores. The secret sharing algorithm provides a stronger form of this anonymity (albeit in a storage-intensive manner), since an isolated 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 pseudonym 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.

6 Future Work

Our experience designing Free Haven revealed several problems which have no simple solutions; further research is surely required. We state some of these problems here and refer to [2] 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 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.

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 could draw from a wide variety of previous work.

Formal Definition of Anonymity: Closely related to the last point is the need to formalise 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 [4]. 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 [10].

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 [14].

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 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.

7 Conclusion

Free Haven aims to solve a problem which no other system currently addresses – creating a decentralized storage service which at the same time protects the anonymity of publishers, readers, and servers, provides a dynamic network, and ensures the availability of each document for a publisher-specified lifetime. We have made progress in identifying the requirements for each of these goals, and designing solutions that meet them.

The current Free Haven design is unfortunately unsuitable for wide deployment, because of several remaining problems. The primary problem is efficiency – unless we can provide a sufficiently friendly and efficient interface to the documents stored in the system, we will find ourselves with very few servers. Indeed, since we need systems that are relatively reliable, we can't make as good use of typical end-user machines as a system like Freenet can. This small number of servers will in turn decrease the amount of robustness that our system offers.

Free Haven uses inefficient broadcasts for communication. A large step 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.

Free Haven sets out to accomplish several goals not considered en masse by other peer-to-peer publishing/storage systems: anonymity for all parties, content-neutral persistence of data, flexibility, and accountability. These ambitious goals are the root cause of existing design difficulties. Without the requirement of long-term persistent storage, strong accountability measures are not as 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. Yet, the study and emphasis of these ambitious goals are Free Haven's contribution and importance in a rapidly-envolving peer-to-peer digital world.

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 (which we followed up with the discussion of reputation systems in Chapter 16, *Accountability*), and Joseph Sokol-Margolis was useful for considering attacks on the system. Andy Oram, our editor, was instrumental in turning this from an academic paper into something which is actually readable. Adam Back and Theodore Hong commented on our assessment of their systems and made our related work section much better. Furthermore, we thank Susan Born, Nathan Mahn, Jean-François Raymond, Anna Lysyanskaya, Adam Smith, and Brett Woolsridge, for further insight and feedback.

References

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

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

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

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

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

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

[7] 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.

[8] 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.

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

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

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

[12] 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.

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

[14] 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.