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

July 25, 2000

Abstract

The Free Haven Project is a design for a system of anonymous storage which resists the attempts of powerful adversaries to find or destroy stored data. We explicitly specify requirements for such a storage system and enumerate distinct notions of anonymity. We suggest a way to classify anonymous systems based on the kinds of anonymity provided. Accountability of servers is explicitly addressed through the use of a "trust network," which attempts to limit the damage done by misbehaving servers. Design choices common to several anonymous storage and publication systems are identified. We end by identifying possible attacks, defenses, and future questions.

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 at the same time. Less extreme examples involve cases in which a large and powerful private organization, such as the Church of Scientology, 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. Recent events have highlighted some shortcomings of already deployed systems; the identification and removal of Napster users who downloaded Metallica songs [25] and the Gnutella Wall of Shame [11] are two examples. These shortcomings are driving the development of a new generation of anonymous publication services, such as Freenet [10], 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 availability and more on persistence of data.

In the process, we hope to clarify some of the requirements for such systems and highlight design choices. We recognize that such services raise significant moral and legal issues which are outside the scope of this paper; for our consideration of these issues, we refer to the first author's thesis [12].

Here we present a design for a system of anonymous storage and begin investigating the requirements for such a system. In particular, we recognize that a system must meet some standard of reliability and utility before it can be useful. Our design operates on a basic unit of data called a *document*. Our requirements for reliably processing these documents are covered in section 2.

We also show that it is not enough simply to talk about "anonymous" storage and publication. In section 3, 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 4 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 gains the server trust and the 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, which we illustrate in section 4.9. The idea is similar to the "give up space now, get space forever" scheme used in Intermemory [9], but allows servers to lose trust if they start behaving badly. In section 4.11 we discuss the "trust network," which is the system that keeps track of trust in each servnet node.

Some of these "contracts" 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 trust by storing shares for others. To leave, a node trades away all of its shares and then disappears. The benefits and mechanisms of trading are described in section 4.7.

Naturally, such a system has powerful adversaries which can launch a range of attacks. We describe some attacks on the Free Haven design in section 5 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 7, 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 Requirements For Anonymous Storage

2.1 Parties

We begin our discussion of requirements for anonymous storage by enumerating the parties in an anonymous storage system. 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.

2.2 Storage Requirements

These requirements are independent of the system's anonymity, but must be fulfilled if the system is to be useful.

Integrity: An authorized reader must receive the same document as originally placed into the system by the publisher, or recognize that the document has been altered.

Controlled Retrieval: The system must specify some policy for retrieving the document.

Efficient Retrieval: An authorized reader must be able to efficiently retrieve the document.

Persistence: It must be clear how long the document can remain stored, and the system must fill all commitments to store a document for a given amount of time.

3 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 [5, 18], we are not aware of a similar investigation into the requirements for publication and storage systems.

We do not give formal 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. Server-anonymity and document-anonymity are crucial if mere possession of some file is cause for action against the server, because they provide protection to a server operator even after his or her machine has been seized by an adversary.

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 is achieved via some sort of secret sharing mechanism, either sharing the document or sharing the key for recreating the document (or both) across servers.

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 which can distinguish server requests from "ordinary" reader requests.

Query-Anonymity: Query-anonymity refers to the notion that over the course of a given document query or request, the "identity" of the document itself is not revealed to the server. In short, this means that although a server may have many different documents stored, which document was served for a given request is not knowable by the server. For an overview of private information retrieval (PIR), see [27].

A weaker form of query-anonymity may be realized through server deniability. The server knows the identity of the requested document, but no third party can be convinced of its identity. This concept is related to deniable encryption [13].

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

3.1 Anonymity and Pseudonymity

So far, we have restricted ourselves to describing anonymity. We could 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 [43] and popularized by May[29] 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 servnet node a reputation; the reputation influences how much data a node can store and provides an incentive to act correctly.

3.2 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, then 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.

Independently, Syverson has developed a logic for talking about the adversary's view of a set of suspects [41].

3.3 Uses of An Ideal World

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 approach this question by introducing an "ideal world" for anonymous publication, influenced by work on secure multiparty computation [6, 26]. The ideal anonymous system in this world consists of a trusted third party (TTP) Ted with secure channels to each party in the system. This TTP receives confidential messages, strips off the source information, and confidentially forwards the messages to their destinations in an unlinkable fashion. Our goal is to come up with a decentralized system that is able to simulate this TTP for each operation. Equivalently, if Alice is communicating through Ted to Bob, a set of protocols which allows Alice to communicate directly to Bob without Ted is said to be anonymous if the transcripts of communication are indistinguishable from those which include Ted. If they are distinguishable, then that difference is exactly the "break" that causes the system to fail to be anonymous.

This informal description requires significant work before it can become a formal model. For one thing, we have not precisely specified the meaning of a "transcript," nor have we investigated the notion of "security" necessary for the channels between Ted, Alice, and Bob. Such work is outside the scope of this paper. Our point is that if an attack succeeds even in the ideal world, then it is "outside" the scope of an anonymous publication service.

4 The Free Haven Design

4.1 Overview

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 paper focuses on the design of the publication system as a back-end for the communications channel.

The agents in our publication system are the **author**, 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 [30]. Authors are agents that produce documents and wish to store them in the service, servers are computers which store data for authors, and readers are people who retrieve documents from the service.

Free Haven is based on a community of servers (which as a whole is termed the "servnet") where each server hosts data from the other servers in exchange for the opportunity to store its own data in the servnet. 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. This means that 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 perhaps more) remailer 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 other nodes in the servnet.

Authors assign an expiration date to documents when they are published; servers make a promise to maintain the availability of a given document until its expiration date is reached. Honest behavior is enforced by other servers losing trust in a server that "drops" data. This trust is monitored and updated by use of a *trust network*. Each server maintains a database containing its trust of the other servers.

4.2 Storage

When an author (call her Alice) wishes to store a new document in Free Haven, she must first identify a Free Haven servnet node which is willing to store the document for her. This might be accomplished by having Alice run a node herself. Alternatively, some nodes might have public interfaces or have publically available reply blocks and be willing to publish data for others.

4.3 Publication

To introduce a file F into the servnet, the publishing server first uses Rabin's information dispersal algorithm (IDA) [35] to break the file into shares $f_1, \ldots, f_n$ where any k shares are sufficient to recreate the file. 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 documents 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.

4.4 Retrieval

Documents in Free Haven are indexed by the public key PK_{doc} used to sign the shares of the document. Readers must locate (or be running) a server which performs the document request. The reader comes up with a key pair $(PK_{\mathrm{client}}, SK_{\mathrm{client}})$ for this transaction, as well as a one-time remailer reply block. The servnet server does a broadcast of "{'request', PK_{doc}, PK_{client}, reply block}" to all servnet nodes that it knows about. These broadcasts can be queued and then sent out in bulk to conserve bandwidth.

Each server that receives the query will check to see if it has any shares with the requested PK_{doc}, and if it does it will encrypt each share in the enclosed public key PK_{client}, and then send 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 ($\geq k$), the client recreates the file and is done.

4.5 Share Expiration

Each share includes an expiration date chosen at share creation time. This is an absolute (as opposed to relative) timestamp which indicates 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 introducing server wants the data to last, considering 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 many of the related works such as Freenet. We maintain an entirely content-neutral approach to the data stored in the system. That is, in the implicit contract between servers, 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.

Designing a protocol which encourages content-neutrality may well mean that server administrators are less liable for the content on the network. Examples include common carrier law, wherein phone companies are not responsible for the content they carry over their systems. However, our strongest argument for maintaining a content-neutral system is that we think this is the most useful approach to a persistent anonymous data storage service. The Free Haven system is designed to provide privacy for its users; rather than being a persistent publication system aimed at convenience, is it designed to be a private low-profile storage system.

4.6 Document Revocation

Some publishing systems allow for documents to be "unpublished" or *revoked*. Revocation has some benefits. Revocation would allow the implementation of a read-write filesystem. Published documents could be updated as newer versions became available. It also allows political dissidents who publish under their real name to realize their mistake and unpublish incriminating documents.

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 author could broadcast his original value x to all servnet nodes 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 an unpublishing tag $H(x)$ 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 breaking forward author-anonymity.

One of the most serious arguments against revocation was raised by Ross Anderson [2]. If the capability to revoke exists, then an adversary can 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. If an adversary wishes to create a pretext to hunt down the publisher of a document, he can still republish the document *with* a revocation tag, and use that as "reasonable cause". Furthermore, node operators or networks may be required by law to refuse unrevocable shares.

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

4.7 Trading

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

Cover for Publishing: If trades are common, then there is no reason to assume that somebody offering a trade is the publisher of a share.

Servers Join and Leave: Trading allows servers to gracefully exit the servnet by giving away all their shares. 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.

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

Accomodates 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. Server operators can use the public key in each share to check to what document a share belongs, then trade away the share without compromising their reputation as a servnet node or the availability of the document.

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

When a server A wants to make a trade (frequency of trade should be a parameter set by the server operator), it chooses another server B from its list of known servers (based on trust and history), and offers a share Φ_{i} and a request for size or duration of a return share. If B is interested, it responds with a share λ_{k} of its own. (The notation X_{y} indicates share y of document X.)

Trades are considered "fair" based on the two-dimensional currency of $size \times duration$, plus a function of the preferences of the servers involved in the trade.

The negotiation is finalized by each server sending an acknowledgement of the trade (including receipt, as described in section 4.8) 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 4.9. Thus, the entire trading handshake takes four rounds: the first two to exchange the shares themselves, and the next two to exchange receipts and at the same time send receipts to the buddies.

By providing the receipt on the third round of the trading handshake, A makes a commitment to store the share λ_{k}. Similarly, the receipt that B generates on the fourth round represents a commitment to store the share Φ_{i}. B could attack A by failing to continue the protocol after the third line: 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 Trust system causes others to recognize that B has misbehaved.

We suggest that when a server A trades a share to a server B, server 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 amount of time is a parameter which requires further study.

4.8 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 five elements is signed by server A. If a broadcast is performed by B (or any other node) complaining about the behavior of A, then 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". 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 trust of the two servers involved in the trade.

We really are not treating the receipt as *proof* of a transaction, but rather as an indication of a commitment to keep safe a given share. This is because the most a given server can do when it detects a misbehaving server is broadcast a complaint and hope the Trust system handles it correctly.

4.9 Accountability

Malicious servers can accept document shares and then fail to store them. If enough shares are lost, the document is unrecoverable. Further, malicious servers can continue to be malicious if there are no mechanisms in place for identifying and excising ill-behaved nodes.

We propose a "buddy system" which associate pairs of shares in a given document with each other. Each share is responsible for maintaining information about the location of the other share, or *buddy*. When a share moves, it notifies its buddy. These notifications are signed by the public key of the document, which is inside each share; it is for this reason that we include $PK_{\mathrm{d}}oc$ in each share and not simply a hash of the public key.

Periodically, a server holding a given share should query for its buddy, to make sure its buddy is still alive. Should its buddy stop responding, then the remaining share (or more correctly, the host currently holding that share) is responsible for announcing which node had responsibility for it when it disappeared, as described below under section 4.11.

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 shares that are destroyed can be regenerated. Such spawning could cause an exponential population explosion of shares: if a share is out of touch for a little while but is not dead, then both shares will end up spawning new copies of themselves. This is a strong argument for not letting shares replicate.

Since the communications channel we have chosen currently has significant latency, each server is responsible for forwarding buddy notifications (based on information stored in the receipts it keeps). These forwarding addresses are therefore available 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 (which includes a buddy check operation), since this document request has presumably been broadcast to all nodes in the servnet.

We have attempted to distinguish between the design goals of robustness and accountability. The fact that a document cannot be lost until a certain threshold of its shares have been lost provides a simple robustness. Accountability, in turn, is provided by the buddy checking and notification system among shares, which protects against malicious or otherwise ill-behaving nodes. Designers can choose the desired levels of robustness and accountability independently.

4.10 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 [16]. Other examples of anonymous communication channels are Onion Routing [42] and Zero Knowledge Systems' Freedom [17]. We refer to David Martin's thesis for a comprehensive overview of anonymous channels in theory and practice [28].

The design and implementation of an anonymous communication channel is an ongoing research topic [1, 5, 7, 8, 20, 21, 22, 24, 33, 36]. The first implementation of the Free Haven design will use the Cypherpunk and Mixmaster remailers as its anonymous channel. For design details, see [15].

4.11 Trust Network

The Trust Network in Free Haven is responsible for creating accountability. Accountability in the face of such strong anonymity is a difficult task; there are many opportunities to try to take advantage of other servers, ranging from merely neglecting to send a receipt after a trade to wrongly accusing a different node of losing a share to more insidious and complex attacks.

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 [34]. Other systems include the Advogato and Slashdot message moderation systems, AOL's Instant Messenger [3], and much of real world commerce and law.

Careful trust management should enable each node to keep track of which nodes it trusts. With the cushioning provided by the information dispersal algorithm, only a large number of the nodes turning evil will result in actual loss of documents.

Each node needs to keep two values describing each other node it knows about: trust and metatrust. Trust signifies a belief that the node in question will obey the Free Haven Protocol. Metatrust represents a belief that the utterances of that node are valuable information. For each of these two values, each node also needs to maintain a confidence and metaconfidence rating. This serves to represent the "stiffness" of the trust value.

Nodes 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 trust or metatrust in a node changes substantially.

The Trust Network provides an easy method of adding new nodes and removing inactive ones. New nodes can contact "introducers" via the anonymous communication channel; these introducers will then broadcast referrals of this new node. Likewise, a node may mark another as "dormant" given some threshold of unanswered requests, such that dormant nodes are not included in broadcasts or trade requests. If a dormant node starts initiating requests again, we conclude it is not actually dormant and resume sending broadcasts and offering trades to this node.

The design of a decentralized "web of trust" network is a complicated research problem itself, beyond the scope of discussion in this paper. The trust network should be able to interpret referrals of other nodes, referrals which are accompanied by receipts, and disagreements between referrals. A node should be able to gain and lose trust independently, and it should recognize when to broadcast its own trust referral. We reference [38] for deeper consideration.

4.12 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 any deployable implementation, we want to ensure 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.

5 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. Obviously, the success of technical attacks is contingent upon the system's security and robustness to attack. The system's design and the nature of anonymity it provides also affects 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 trust network; and on the various aspects of anonymity relevant to anonymous storage and publication, as described in section 3. For a more in-depth consideration of attacks, we refer to [12].

5.1 Attacks on Documents or the Servnet

Physical attack: Destroy a servnet node.

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 nodes before availability is compromised.

Legal action: Find a physical servnet node, 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 servnet node administrators. Claim that the design is patented or otherwise illegal. Sue the Free Haven Project and any known node 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 servnet nodes 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 node.

Prevention: We must assume that our communications channel has adequate protection and buffering against this attack, such as the use of client puzzles [23]. 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 or 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.

5.2 Attacks on the Trust Network

While attacks against the Trust Network[1] are related to attacks directly against nodes, 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 nodes.

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

[1] Parts of this section were originally written by Brian Sniffen in [38].

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

Prevention: The trust economy is designed to make this unprofitable. The size-time currency means that a corrupt node has to initially store data at least equivalent to that it later deletes. A node which engages in this behavior should be caught by the buddy system when it deletes each share.

Buddy Coopting: If a corrupt node (or group of colluding nodes) 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. Nodes also can modify trust 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 nodes 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 direct these to specific hosts.

Prevention: The metaconfidence trust rating 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 trust 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-node 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 node 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.

5.3 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 node 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 servnet node, and then collect routine status and participation information (such as server lists) from other nodes. 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 trust 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 node administrators are subpoenaed or otherwise pressured to release information about these entities, they can openly disclaim any knowledge. Obviously, the level of anonymity provided by the is based on its robustness to traffic analysis and similar attacks.

6 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.

6.1 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.

6.2 Napster

The Napster service [31] 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 suspect that many thousands more would participate. The existence of Napster shows that demand exists for a distributed storage and retrieval service.

6.3 Gnutella

Gnutella [14] 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 [40]. There are a small number of central nodes which would be ideal targets for collecting information about users.

In addition, 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 [11], 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.

6.4 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.

6.5 Freenet

Like Gnutella, Freenet [10] 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. For example, Yugoslav phone books are currently being collected "to document residency for the close to one million people forced to evacuate Kosovo"[32]; these phone books might not survive a popularity contest.

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 called isolated-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 expired or forwarded onwards.

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 has "more" anonymity than Gnutella provides.

Further, statistical attacks similar to those described in the Crowds [36] 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.

Another attack to consider is that simply doing a request from a strict political environment will cause a copy of the data to migrate into that jurisdiction, giving law enforcement more reason to shut those servers down. On the other hand, it may well be that such environments will close down servers without requiring "sufficient cause", so the political and legal arguments are meaningless.

6.6 Publius

Publius [44] 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 isolated-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 [39] might make the system more efficient.

The entire scheme is based on a static, system-wide list of available servers. Since these servers are permanent, there is no support for adding new servers or purging dead ones. More importantly, there is no support for recognizing misbehaving servers and removing them.

7 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), information-theoretic (I-T), 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 negligible function $neg(k)$. Information-theoretic is the same, except with no computational restrictions on the adversary. 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.

Project	Publisher			Reader			Server			Document		Query	
	C	I-T	P-F	C	I-T	P-F	C	I-T	P-F	C	I-T	C	I-T
Gnutella													
Eternity Usenet	$\surd$		$\surd$	?		?							
FreeNet	?		?	?		?				$\surd$			
Publius	$\surd$		$\surd$							$\surd$	$\surd$		
Free Haven	$\surd$		$\surd$	$\surd$		$\surd$	$\surd$			$\surd$			
FH + ideal mix	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$			

Table 1: Anonymity Properties of Publishing Systems

For the purposes of this discussion, we will assume that the anonymous channel used by Free Haven is only computationally secure. An "ideal mix", in contrast, would be a perfectly anonymous channel which resists any attack we of which we could think, including those by a computationally unbound adversary. All anonymous channels are assumed to keep no records of communication.

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 acheives 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 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 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. 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, 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.

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 last line in this table is important to note. While it implies that Free Haven achieves anonymity in many areas, this is misleading: the ideal anonymous channel is actually providing the first nine aspects of anonymity. Assuming a robust ideal anonymous channel, there would be no linkability between transactions, and mere computational ability on the part of the adversary would be insufficient to identify the parties in a transaction.

This would mean that we could leave most of the anonymity 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.

8 Future Work

Our experience designing Free Haven revealed several problems which proved to be too hard to solve at this time. We state some of these problems here and refer to the first author's thesis for in-depth consideration [12].

Deployed Free Low-Latency Pseudonymous Channel: Free Haven requires pseudonyms in order to create server reputations. The only current widely deployed channels which supports pseudonyms seem to be the Cypherpunk remailer network [30] and ZKS Freedom mail. These networks run over SMTP and consequently have high latency. This high latency complicates protocol design. In addition, Freedom mail requires payment and is not yet open source. Payment may be complicated for international users or may allow participation to be traced, and open source facilitates peer review of the implementation.

Accountability and Trust: 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 trust algebra" for formally reasoning about a servnet node's reliability based on various circumstances which would allow us to verify trust protocols.

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 [20], Kesdogan, Egner, and Buschkes [24], and Berthold, Federrath, and Kohntopp [5] which apply statistical modelling to mix-nets.

Formal Definition of Anonymity: Closely related to the last point is the need to formalise the "kinds of anonymity" presented in section 3. 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[19]. 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[37].

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 [45].

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

9 Conclusion

We have presented a design for a content-neutral decentralized anonymous storage service and investigated the kinds of anonymity required for such a service. The design protects anonymity at least as much, and in some cases more, than related work. Unfortunately, we have also shown that this does not say that much, because large problems remain with evaluating the level of protection provided by any of these systems, much less providing "good" protection. 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 long and 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 design and background for the trust system, and Joseph Sokol-Margolis was useful for considering attacks on the system. 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, Brett Woolsridge, and our anonymous referees 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] Oliver Berthold, Hannes Federrath, and Marit Kohntopp. Anonymity and unobservability on the Internet. In *Workshop on Freedom and Privacy by Design : CFP 2000*, 2000.

[6] Ran Canetti. PhD thesis, MIT.

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

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

[9] 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.

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

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

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

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

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

[15] 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.

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

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

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

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

[20] 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.

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

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

[23] 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.

[24] Kesdogan, Egner, and Bschkes. Stop and go mixes : Providing probabilistic anonymity in an open system. In *1998 Information Hiding Workshop.*

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

[26] Anna Lysyanskaya. Personal communication.

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

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

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

[30] 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.

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

[32] 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.

[33] 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.

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

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

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

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

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

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

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

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

[42] 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.

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

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

[45] 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.