The Free Haven Project: Design and Deployment of an Anonymous Secure Data Haven

by

Roger R. Dingledine

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Computer Science and Engineering

and

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2000

Author ....
Department of Electrical Engineering and Computer Science
May 22, 2000

Certified by ....
Ron Rivest
Webster Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

The Free Haven Project: Design and Deployment of an Anonymous Secure Data Haven

by

Roger R. Dingledine

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2000, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Computer Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The Free Haven Project aims to deploy a system for distributed data storage which is robust against attempts by powerful adversaries to find and destroy stored data. Free Haven uses a mixnet for communication, and it emphasizes distributed, reliable, and anonymous storage over efficient retrieval. We provide an outline of a formal definition of anonymity, to help characterize the protection that Free Haven provides and to help compare related services. We also provide some background from case law about anonymous speech and anonymous publication, and examine some of the ethical and moral implications of an anonymous publishing service. In addition, we describe a variety of attacks on the system and ways of protecting against these attacks. Some of the problems Free Haven addresses include providing sufficient accountability without sacrificing anonymity, building trust between servers based entirely on their observed behavior, and providing user interfaces that will make the system easy for end-users.

Thesis Supervisor: Ron Rivest
Title: Webster Professor of Computer Science and Engineering

Contents

Chapter 1

Introduction and Requirements

1.1 Motivation

The internet is moving in the direction of increasing freedom of information. National boundary lines are growing increasingly blurred. At the same time as a strong sense of global community is growing, technical advances have provided greatly increased bandwidth and an enormous amount of computing power and well-connected storage. However, the increases in speed and efficiency have not brought comparable increases in privacy and anonymity on the internet – indeed, governments and especially corporations are beginning to realize that they can leverage the internet to provide detailed information about the interests and behaviors of existing or potential customers. Court cases, such as the Church of Scientology's lawsuit against Johan Helsingius [42] or the more recent OpenDVD debate [55] (and subsequent arrest of DeCSS author Jon Lech Johansen), demonstrate that the internet currently lacks an adequate infrastructure for truly anonymous publication or distribution of documents or other data.

Indeed, there are a number of other deeper motivations for the deployment of an anonymous publishing service such as Free Haven. Not only do we hope to assist those like Helsingius and Johansen, but we have the wider goals of pushing the world a few more steps in the direction of free and open information and communication. In Germany, Internet Service Providers such as AOL are liable for the content that passes across their systems [82]. Recent British legislation threatens to make citizens responsible for the content of encrypted documents that they're holding, even if they don't possess the ability to read these documents [74]. Such restrictions on the free flow of information, however, are already being attacked: for example, American hackers are attempting to break holes in China's "Great Firewall" to allow Chinese citizens access to Western media [11].

In addition to such revolutionary actions, there are a wide range of activist projects which employ the internet for publicity but focus on helping real people in the real world. Such projects include

Pirate Radio [29], a loose confederation of radio operators joined in the belief that ordinary citizens can regulate the airwaves more efficiently and more responsibly than a government organization; as well as mutual aid societies such as Food Not Bombs! [45], an organization which "serves free food in public places to dramatise the plight of the homeless, the callousness of the system and our capacity to solve social problems through our own actions without government or capitalism."

By providing tools to enable safer and more reliable communication for organizations fighting for increased rights of individuals rather than nations or corporations, as well as strengthening the capabilities of political dissidents and other individuals to speak out anonymously about their situations, the members of the Free Haven Project hope to help pave the way to a modern society where freedom of speech and freedom of information are integral parts of everyday life.

1.2 Project Summary

The Free Haven Project intends to deploy a system which provides a good infrastructure for anonymous publication. Specifically, this means that the publisher of a given document should not be known; that clients requesting the document should not have to identify themselves to anyone; and that the current location of the document should not be known. Additionally, it would be preferable to limit the number of opportunities where an outsider can show that a given document passed through a given computer. A more thorough examination of our requirements and notions of anonymity can be found in Chapter 2.

The overall design 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 data of its own in the servnet. When an author wishes to publish a document, she breaks the document into shares, where a subset (any k of n) is sufficient to reconstruct the document. Then for each share, she negotiates for some server to publish that share on the servnet. The servers then trade shares around behind the scenes. When a reader wishes to retrieve a document from the servnet, she requests it from any server, providing a location and key which can be used to deliver the document in a private manner. This server broadcasts the request to all other servers, and those which are holding shares for that document encrypt them and deliver them to the reader's location. Also behind the scenes, the shares employ what is essentially the 'buddy system' to maintain some accountability: servers which drop shares or are otherwise unreliable get noticed after a while, and are trusted less. A trust module on each server maintains a database on the behavior of each other server, based on past direct experience and also what other servers have said. For communication both between servers and between the servnet and readers, we rely on an existing mixnet infrastructure to provide an anonymous channel.

The system is designed to store data without concern for its popularity or controversial nature.

Possible uses include storing source code or binaries for software which is currently under legal debate, such as the recent DeCSS controversy or other software with patent issues; publishing political speech in an anonymous fashion for people afraid that tying their speech to their public persona will damage their reputation; or even storing more normal-looking data like a set of public records from Kosovo.

Free Haven is designed more for anonymity and persistence of documents than for frequent querying — we expect that in many cases, interesting material will be retrieved from the system and published in a more available fashion (such as normal web pages) in a jurisdiction where such publishing is more reasonable. Then the document in the servnet would only need to be accessed if the other sources were shut down.

The potential adversaries are many and diverse: governments, corporations, and individuals all have reason to oppose the system. There will be social attacks from citizens and countries trying to undermine the trust in the security of the system, as well as attacking the motivation for servnet node operators to continue running nodes. There will be political attacks, using the influence of a country's leaders to discourage use of the servnet. There will be government and legal attacks, where authorities attempt to shut down servnet nodes or arrest operators. Indeed, in many cases ordinary citizens can recruit the power of the government through lawsuits or subpoenas. Multinational corporations will hold sway over several countries, influencing them to pass similar laws against anonymous networks. There will be technical attacks, both from individuals and from corporations and national intelligence agencies, targeted either at the system as a whole or at particular documents or node operators, to reduce the quality of service or gain control of part of the network. Clearly the system needs to be designed with stability, security, and longevity in mind.

1.3 Design Requirements

More formally, requirements beyond anonymity for a stable and useful system fall into two categories:

- Required Operations:
  - The system must provide a mechanism for anonymously **inserting** a document into the servnet.
  - The system must provide a mechanism for anonymously **retrieving** a document from the servnet, including verifying that the retrieved document is identical to the original document.
  - The system must provide a mechanism for **expiring** documents: the duration of a document should be decided by the publisher when that document is published to the servnet, and the document should be available (and immutable) until that duration expires.

  - The system must provide a mechanism for smoothly **adding servers** to the servnet without impacting functionality.
  - The system must provide a mechanism for **recognizing inactive or dead servers**; it should consequently cease to use or query them.

- Guiding Principles:
  - The system must be **robust**: loss of perhaps up to half of the participating servers should not imply loss of any documents. In addition, the amount of damage that compromised or otherwise 'evil' servers can perform should be limited. This might be accomplished by a trust network, where each node actively maintains an opinion of other nodes, and nodes inform each other when they change an opinion.
  - The system must be **simple**: complex protocols and heuristics invite security weaknesses. It must be self-contained and based on realistic technological expectations. For instance, we cannot rely on a stable international electronic cash infrastructure.
  - The system must be **modular** enough that components can be upgraded in-place without impacting functionality.
  - The system must be **decentralized**: to maintain efficiency, security, and reliability, no single server or small subset of the servers should be a bottleneck anywhere in the protocol.
  - The system must provide **flexibility** on a per-server level: server operators should be able to decide how paranoid or trusting they are, how many resources to provide to the servnet, etc.
  - The components upon which the system relies must be **free and open source**, in the sense that modification and redistribution is explicitly permitted.
  - The system is **content-neutral**: popularity or popular opinion of a document should not influence its duration in the servnet.

We assume that there will be some generous individuals out there who believe in the goals of the system and will donate some services. Notice that efficiency isn't on the list – we can afford to have more overhead (both in time and in bandwidth) if we get stronger anonymity out of it.

1.4 About this document

This paper was written as a joint effort by the members of the Free Haven project, a group of students led by Roger Dingledine (primary author). The other project members contributing to this document are Michael Freedman (Sections 3.1, 7.1, 8.1, all of Chapter 6, and part of Appendix B), David Molnar (Section 3.1 and Appendix B), Brian Sniffen (Sections 3.3, part of 5.8, 7.3, and 8.3), and Todd Kamin (Section 5.10).

Chapter 2

Defining Anonymity

Many anonymous publication systems claim 'anonymity' without specifying a precise definition. Indeed, they often fail to specify what protections users and operators receive from their system, as well as what protections users and operators do not receive. These protections are a function of both the actual publication system and the communications channel which it utilizes.

While the anonymity requirements of communications channels have been considered previously in depth [39] [15], this chapter addresses anonymity at a higher level: the publication systems themselves.

The current Free Haven project design does not achieve all of the attributes of anonymity that we might want to provide. This chapter is thus not an enumeration of design goals which Free Haven meets, but rather an enumeration of design goals for the ideal anonymous publication system.

The following sections start out by describing what a speaker might expect to achieve. We then list some agents in an anonymous publication system, and address anonymity for each of these agents separately. We present some intuitive notions of types or degrees of anonymity, as well as some models of adversaries that help in thinking about how anonymous a system as a whole might be. Finally, we attempt to assess real-world projects in the face of these new definitions and models.

2.1 Defining Privacy

Privacy is the ability to control dissemination of information about yourself. The level of anonymity you have in a given scenario is a result of the choices you make about your privacy.

In particular, the speaker may have control in several different dimensions over dissemination of information about his own speech:

1. **Linkability:** The speaker controls the ability of readers to link his utterances. If there are distinguishing characteristics that provide this linkability, these characteristics are called a

pseudonym.

2. **Replies:** The speaker controls whether readers can reply to his utterances. Further parameters include whether the reply is private, and persistence of the ability to reply (perhaps the ability to reply expires a week after the publication).
3. **Content Leaks:** The speaker controls how much partial information about himself is leaked based on the content of his speech. For instance, the speaker may choose to reveal certain information such as credentials for being authorized to read the New York Times on the web.
4. **Channel Leaks:** The speaker controls how much partial information is leaked based on the communications channel he chooses to use. For example, a satellite TV broadcaster may use an encrypted communications channel in an attempt to allow only paying customers access to programming. Alternatively, a speaker may intentionally use an imperfectly anonymous channel for the sake of efficiency or convenience.
5. **Persistence:** The speaker controls how long his speech persists after the publication.
6. **Readers:** The speaker controls which other parties will be able to read his speech.

2.2 Agents and Operations

The above list describes some freedoms which a given *speaker* may have available to him. A number of these freedoms have analogs when considered in the context of other parties in the communication.

In general, there are several agents in an anonymous publication system: these include the author, the reader, and the server. There are a number of operations that the system might support, such as (at a minimum) inserting a document into the system, and retrieving the document from the system. We address each of these agents and operations separately, to try to build an intuitive notion about which acts allow dissemination of information.

2.2.1 Author-anonymity

Author-anonymity means that the original author of a given document should not be known. This characteristic of anonymity is one of the integral parts of almost any anonymous network or service. Even so-called ‘anonymous remailers’, which are simply anonymous forwarders and don’t support persistence or storage of the data, provide author-anonymity. Indeed, anonymous remailers can be combined with public storage and distribution systems such as Usenet to offer a rudimentary but very easy to construct and deploy publication service that allows persistently available data and provides author-anonymity.

2.2.2 Publisher-anonymity

While author-anonymity addresses the original author of the document itself, publisher-anonymity addresses the agent that originally introduces the document into the system. In some cases the author and the publisher may be the same entity, but in the general case the author may make use of a separate individual, either a third party or a server in the system, to introduce the document. Separating these two notions of 'origin' makes it clearer what protections the system provides.

2.2.3 Reader-anonymity

Reader-anonymity means that readers requesting a document should not have to identify themselves to anyone. In particular, this means that when a reader performs a document request at a given server, this server is unable to determine the identity or location of the reader.

This class of anonymity is crucial for protecting people from disclosing that they are interested in or accessing certain types of material. For instance, a user of the system might not want it known whether she is downloading material from the Democrats' web page, or from the Republicans' web page. Reader-anonymity ensures the privacy of the vast majority of the system's *users*, a concern that is often ignored.

2.2.4 Server-anonymity

Server-anonymity means that the location of the document should not be known or knowable. Specifically, 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 (or shares of it). This restriction implies that the retrieved documents do not provably pass through any given server that receives a request. This protection is crucial for materials where mere possession is cause for action against the server. The most obvious example of such materials is child pornography, but in fact any material which is the subject of an injunction or criminalized by law needs this kind of protection. Depending upon jurisdiction, this material runs the gamut from Amnesty International pamphlets to DeCSS to neo-Nazi propaganda to advertisements for alcohol.

Many services rely on sheer volume of servers, each containing the data, to dissuade organizations from attacking any given server for possessing the data. However, this approach to constructing a decentralized service also dissuades large corporations from participating in these networks, due to liability and reputation concerns. Also, there may be some circumstances, such as the OpenDVD suits[78], where adversaries are willing to expend the energy to track down all servers which offer a given document. Indeed, making an example out of even a few high profile server operators can go a long way towards reducing the availability of a document.

2.2.5 Document-anonymity

Document-anonymity means that the server does not know the contents of the document that it is storing or helping to store. This is broken down into two scenarios. 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. Generally 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. On the other hand, connected-server document-anonymity means that the server is allowed to communicate and compare data with all other servers in the system, but is still unable to determine the contents of the document. Since a connected server may well be able to act as a reader and do a document request itself, it seems that connected-server document-anonymity is difficult to achieve without some trusted party acting as intermediary and authenticating and authorizing readers.

Notice that merely encrypting the document before publishing it into the system is not sufficient to achieve document-anonymity: we are concerned here not with confidentiality of the published document, but instead with whether the given server can recreate the bits that were inserted into the system.

2.2.6 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. This process of private information retrieval (PIR) is typically done by a computationally intensive (or bandwidth-intensive) process of downloading lots of other documents at the same time to mask which document is being retrieved, or perhaps by using lots of servers in the transaction. Alternatively, the use of massive amounts of resources might be solved by particularly clever mathematics. For an overview of PIR, we suggest Malkin's thesis [60].

2.3 Linkability: Anonymity vs. Pseudonymity

The above classes of anonymity describe the issues regarding each of the agents or operations in the system. However, there are some other broader characteristics of anonymity to consider.

Anonymity, when compared to pseudonymity, means that the agent performing the operation has no observable persistent characteristics. For instance, turning on a radio is a nice way of receiving information anonymously (modulo Tempest attacks — actually, [28] describes a company which records which radio station cars on the highway are tuned to. So far, they claim to have surveyed over 125 million cars.). Reading the advertisements in the New York Times is another good example, though again it is not perfect, since it seems conceivable that somebody could track who purchases a given newspaper and somehow extend this to who reads a given newspaper. In general, broadcast media are good ways to achieve anonymity. A Usenet feed is a bad example here, because if the user is not the systems administrator of that node, he can be monitored by the systems administrator; if he is the systems administrator of that node, then in some sense his upstream provider can 'prove' that he received the information that he read (even though the upstream provider has no idea which articles he actually physically read).

Pseudonymity, on the other hand, means there is some characteristic associated with the agent for that transaction, and this characteristic provides some mechanism for recognizing that other transactions also involved this party.

Both anonymity and pseudonymity in this context retain the notion of privacy of location. Location describes the actual physical connection to the communications medium: the speaker is in some sense physically *at* his location. Anonymity is in some sense 'more private' than pseudonymity, because there's less to trace, but having a pseudonym does not necessarily imply that location is public – the pseudonym could well be a reply block on a mixnet, or even simply a keypair which an author uses to sign all of his documents.

2.4 Partial Anonymity vs. Full Anonymity

The notion of full anonymity is similar to the use of one-time passwords in encryption: fully anonymous speech means that hearing the speech gives a listener no more information about the identity of the speaker than he had beforehand. In reality, though, every set of candidates is limited in size, and indeed the adversary often has partial information about the suspect – for instance, 'he or she has a high-bandwidth Internet connection', or 'he or she probably lives in California based on activity patterns and routing analysis'.

So in this clearer context, the question shifts from 'is it anonymous?' to 'is it anonymous enough?' If the original set of suspects has n members, then for sufficiently large n a system which leaks no information that might reduce the set of suspects seems to be 'anonymous enough'. However, we have to bear in mind that we may well be trying to protect the identities of all the users of a given service – that is, even evidence implying that a given user is one of n users of the service may be sufficient to make him suspicious, thus compromising his anonymity. This may discourage corporations and persons who are particularly concerned about their reputation from participating in a given anonymous service, and indeed it may put ordinary users at risk as well.

It is not even as simple as whether a user is inside or outside the 'set of suspects'. Often there is no clearly delineated set, and for each user no boolean value of 'suspected' or not. A given member of this set might be more suspicious than another member; if the adversary has this knowledge beforehand, then the system can still be fully anonymous if it does not leak any new information to confirm or deny that adversary's initial guesses.

2.5 Characteristics of Communication Channels

The speaker has control over whether to publish his speech over a given channel, based on the characteristics of that particular channel. This means that he might tailor his speech, or choose not to speak at all, based on how much protection the channel provides and how much anonymity he desires.

2.5.1 Computational vs. Information-Theoretic Anonymity

One issue to consider is the notion of how protected a given address is: does it rely on computational complexity to protect its anonymity (e.g., a reply block address on a conventional mixnet), or does it use some other technique to make the address unknowable even in the face of a computationally powerful adversary?

There are really two alternatives to computational anonymity. The first alternative is that the adversary has the transcript of the communication, but is still unable to break its anonymity – this is what we call information-theoretic anonymity. This might be modeled by a trusted third party (with trusted channels) acting as moderator, or it might perhaps be implemented in some mechanism similar to a one-time pad.

The second option is that the adversary is simply unable to obtain a transcript of the communication, or perhaps the usefulness of the transcript 'expires' quickly enough to be equivalent to no transcript at all. This might happen in the case of some physical medium which is untappable (perhaps quantum channels give us some starting point for this), or in the case of an improved mixnet where a given address no longer maps to or is traceable to its destination after the transaction. This also implies that there is no mechanism for the adversary to take a snapshot of the entire network at that point – if he could, then he might be able to go offline with that snapshot and break the anonymity of the channels.

2.5.2 Perfect Forward Anonymity

Perfect forward secrecy means that after a given transaction is done, there is nothing new that the adversary can 'get' to help him decrypt the transcript. Similarly, perfect forward anonymity is the notion that after a given transaction is done, there is nothing new that the adversary can get that can help him identify the location or identity of either of the communicating parties.

In effect, this might be achieved by negotiating a 'session location' (the anonymity analog of a session key) between the parties for the purposes of that transaction. For instance, this session location might be a double-blinded virtual location in a high-entropy onion-routed network, where the transaction takes place effectively instantaneously, and then all records of paths to and from the virtual location are destroyed.

In this case, a snapshot could in fact be taken of the system at that point, but this falls under the realm of computational anonymity as described above.

2.6 What does it mean to 'break' an anonymous system?

This question also comes up in the context of encryption: what does it mean to break an encryption system? Goldreich argues [40] that since we do not know which parts of the plaintext might be useful to a given adversary or in a given situation, we must protect every aspect of the plaintext. More precisely, we must preserve the amount of partial information that the adversary may have a priori about the plaintext (due to a non-uniform distribution, for example); that is, we must prevent the adversary from determining any new information beyond what he starts with.

It is tempting to claim that even this characterization is insufficient for anonymous systems, as we describe above regarding Partial Anonymity: even knowing that an individual is among the group of users may be sufficient to make him suspicious, and thus sufficient to break the anonymity of the system. On the other hand, even an ideal channel could not prevent this disclosure: if there is a method of enumerating the users of a service, then it seems that there is no means of preventing some amount of information from leaking. Similarly, if there is no method of enumerating any users of the service, then it seems that we have achieved ideal anonymity.

There are issues to bear in mind from the social engineering perspective as well. No matter how powerful or effective your anonymity protection is, if a user signs his name to his document his anonymity is broken. Also, more subtle attacks such as word choice correlation or writing analysis might yield clues that allow better than even chances of guessing. All of the above models could be based on the assumption that a given document is a random distribution of bits. However, we might instead assume that there is some amount of a priori information that the adversary can guess or assume about the document, and as long as our communications don't leak *more than* this information, our system is anonymous.

We can picture the ideal anonymous system as a trusted third party (TTP) with secure channels to each party in the system. This TTP would receive confidential messages, strip off the source information, and confidentially forward the messages to their destination in an unlinkable fashion. Therefore, 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 leak privacy.

2.7 Modeling the Adversary

Potential adversaries have a variety of capabilities, and similarly they have a variety of goals.

We break down basic capabilities into active attacks versus passive attacks – whether or not the adversary has the ability to actively modify or prevent communications. However, it is not just a matter of whether an adversary has full access to some part of the system or no access: it is more reasonable to model the adversary as controlling some fraction of the edges in the communications channel ('edge' adversaries), and some other fraction of the servers in the publication system ('server' adversaries). This would in turn result in the adversary having some probability of achieving his goal; this probability increases as percentage control over the system increases.

While some adversaries might be able to monitor or alter the link between two servers, another class of capability includes the ability to observe or even modify the private state of a mixnet node or Free Haven server – that is, data which is internal to the server and does not get sent over the network.

Further still, adversaries have capabilities describing their capacity to perform computation – an adversary is said to be computationally bounded, or unbounded, as per the normal definitions.

We also need some notion of the time interval over which the adversary may collect information or modify state. If an adversary has the ability to watch or affect edges or servers over a long period of time, he may have a larger overall impact on the system than if he only had control for a short amount of time.

Aside from the capabilities that an adversary might show, there is also the matter of building a taxonomy of goals that an adversary might hope to achieve. In very broad form, an adversary attacks either to identify a target or to destroy functionality of the system.

Attacks to identify can be further broken down into attacks to determine the identity of the target, and attacks to provide proof to a third party of the identity of the target. Such potential targets include publishers, readers, and servers.

Attacks to destroy include attacks to modify or prevent communications, attacks against the functionality or owners of servers, and attacks against the integrity of documents themselves.

2.8 Toward a formal definition of anonymity

Attempting to formalize anonymity can be more subtle than attempting to formalize encryption or secrecy goals. When assessing the security of a given encryption scheme, there are generally two parties who might have information: the sender and the receiver. The sender generates some bits, and sends these bits over some channel to the receiver; there are generally measures in place to ensure that the bits are not modified in the channel, but the exact characteristics of the channel are not very important. In our case, however, formalizing the notion of anonymity involves identifying the characteristics of the communications channel that might leak information about the identity of either of the parties involved in the transaction.

To put it in more concrete terms, if Alice is running a server at her company, what about the cached data in the corporate-run intrusion detection system down the hall? Do we model that as part of Alice, such that so-called 'edge' adversaries do not have access to it? What if one of the packets in Alice's transmission gets sent off course, and 120 seconds later an ICMP destination unreachable packet returns to her with its data segment an exact copy of one of the packets that she sent to Bob? Is 120 seconds negligible? What isn't negligible?

The notion of anonymity becomes complex very quickly as we introduce real-world factors and considerations such as these. We develop a notion of 'publisher indistinguishability' as the beginning of a mechanism for assessing the anonymity that a system provides. Informally, the adversary knows that a document Φ was published by either party P_0 or party P_1. We say that a system provides publisher indistinguishability if the adversary has negligible advantage over guessing whether P_0 or P_1 is the real publisher.

We might alternatively have defined this aspect of anonymity in terms of two documents Φ_0 and Φ_1 and a single user Alice who is known to be a publisher; the system may have achieved some different level of anonymity if the adversary has only negligible advantage over guessing in determining which of Φ_0 and Φ_1 Alice published.

Clearly this topic has ample room for future research.

Chapter 3

Related and Alternate Works

There are a wide variety of works related to the topics that we address. In particular, these are broken down into three sections: works that describe or implement communications channels, works that describe or implement a publishing system itself, and works that relate to our system of weighing and maintaining trust between servers.

3.1 Anonymous Channels

[1] Several approaches have been proposed to achieve anonymity on an Internet communications channel. This section describes several real-world projects, which we analyze in terms of anonymity provided and resistance to various types of attack. We include a more in-depth review of anonymous communications channels in appendix B, truncated here for length and readability.

3.1.1 Proxy Servers: Anonymizer

Proxy services provide one of the most basic forms of anonymity, inserting a third party between the sender and recipient of a given message. Proxy services are characterized as having only one centralized layer of separation between message sender and recipient. The proxy serves as a "trusted third party," responsible for removing distinguishing information from sender requests.

The Anonymizer was one of the first examples of a *form-based web proxy* [7]. Users point their browsers at the Anonymizer page at `www.anonymizer.com`. Once there, they enter their destination URL into a form displayed on that page. The Anonymizer then acts as an `http` proxy for these users, stripping off all identifying information from `http` requests and forwarding them on to the destination URL.

[1] This section was written by Michael Freedman and David Molnar.

The functionality is limited. Only http requests are proxied, and the Anonymizer does not handle cgi scripts. In addition, unless the user chains several proxies together, he or she may be vulnerable to an adversary which tries to correlate incoming and outgoing http requests. Only the data stream is anonymized, not the connection itself. Therefore, the proxy does not prevent traffic analysis attacks like tracking data as it moves through the network.

Proxies only provide unlinkability between sender and receiver, given that the proxy itself remains uncompromised. This unlinkability does not have the quality of perfect forward anonymity, as proxy users often connect from the same IP address. Therefore, any future information used to gain linkability between sender and receiver (i.e., intersection attacks, traffic analysis) can be used against previously recorded communications.

Sender and receiver anonymity is lost to an adversary that may monitor incoming traffic to the proxy. While the actual contents of the message might still be computationally secure via encryption, the adversary can correlate the message to a sender/receiver agent.

This loss of sender/receiver anonymity plagues all systems which include external clients which interact through a separate communications channel – that is, we can define some distinct edge of the channel. If an adversary can monitor this edge link or the first-hop node within the channel, this observer gains agent-message correlation. Obviously, the ability to monitor this link or node depends on the adversary's resources and the number of links and nodes which exist. In a proxy system, this number is small. In a globally-distributed mixnet, this number could be very large. The adversary's ability also depends on her focus: whether she is observing messages and agents at random, or if she is monitored specific senders/receivers on purpose.

3.1.2 Mixmaster Remailer

The pursuit of anonymity on the Internet was kicked off by David Chaum in 1981 with a paper in Communications of the ACM describing a system called a "Mix-net" [20]. This system uses a very simple technique to provide anonymity: a sender and receiver are linked by a chain of servers called Mixes. Each Mix in the chain strips off the identifying marks on incoming messages and then sends the message to the next Mix, based on routing instructions which are encrypted with its public key. Comparatively simple to understand and implement, this Mix-net (or "mix-net" or "mixnet") design is used in almost all of today's practical anonymous channels.

Until the rise of proxy-based and TCP/IP-based systems, the most popular form of anonymous communication was the *anonymous remailer*: a form of mix which works for e-mail sent over SMTP. We describe the development of remailer systems in greater depth in appendix B; in short, the evolution of remailers has led to the Mixmaster Type II remailer, designed by Lance Cottrell [35].

Each Mixmaster remailer has an RSA public key and uses a hybrid symmetric-key encryption system. Every message is encrypted with a separate 3DES key for each mix node in a chain between the sender and receiver; these 3DES keys are in turn encrypted with the public keys of each mix node. All Mixmaster messages are padded to the same length.

When a message reaches a mix node, it decrypts the header, decrypts the body of the message, and then places the message in a "message pool." Once enough messages have been placed in the pool, the node picks a random message to forward. As the underlying next-hop header in the message has been decrypted, the node knows to which destination to send this message. In this way a chain of remailers can be built, such that the first remailer in the chain knows the sender, the last remailer knows the recipient, and the middle remailers know neither.

Remailers also allow for *reply blocks*. These consist of a series of routing instructions for a chain of remailers which define a route through the remailer net to an address. Reply blocks allow users to create and maintain pseudonyms which receive e-mail. By prepending the reply block to a message and sending the two together to the first remailer in the chain, a message can be sent to a party without knowing his or her real e-mail address.

3.1.3 Rewebber

Goldberg and Wagner applied Mixes to the task of designing an anonymous publishing network called Rewebber[38]. Rewebber uses URLs which contain the name of a Rewebber server and a packet of encrypted information. When typed into a web browser, the URL sends the browser to the Rewebber server, which decrypts the associated packet to find the address of either another Rewebber server or a legitimate web site. In this way, web sites can publish content without revealing their location.

Mapping between intelligible names and Rewebber URLs is performed by a name server called the Temporary Autonomous Zone (TAZ), named after a novel by Hakim Bey. The point of the "Temporary" in the name of the nameserver (and the novel) is that static structures are vulnerable to attack. Continually refreshing the Rewebber URL makes it harder for an adversary to gain information about the server to which it refers.

3.1.4 Onion Routing

The Onion Routing system designed by Syverson, et. al. creates a mix-net for TCP/IP connections [95, 77]. In the Onion Routing system, a mixnet packet, or "onion", is created by successively encrypting a packet with the public keys of several mix servers, or "onion routers."

When a user places a message into the system, an "onion proxy" determines a route through the anonymous network and onion encrypts the message accordingly. Each onion router which receives the message peels the topmost layer, as normal, then adds some key seed material to be used to generate keys for the anonymous communication. As usual, the changing nature of the onion – the "peeling" process – stops message coding attacks. Onions are numbered and have expire times, to stop replay attacks. Onion routers maintain network topology by communicating with neighbors, using this information to initially build routes when messages are funneled into the system. By this process, routers also establish shared DES keys for link encryption.

The routing is performed on the application layer of onion proxies, the path between proxies dependent upon the underlying IP network. Therefore, this type of system is comparable to loose source routing.

Onion Routing is mainly used for sender-anonymous communications with non-anonymous receivers. Users may wish to Web browse, send email, or use applications such as `rlogin`. In most of these real-time applications, the user supplies the destination hostname/port or IP address/port. Therefore, this system only provides receiver-anonymity from a third-party, not from the sender.

Furthermore, Onion Routing makes no attempt to stop timing attacks using traffic analysis at the network endpoints. They assume that the routing infrastructure is uniformly busy, thus making passive intra-network timing difficult. However, the network might not be statistically uniformly busy, and attackers can tell if two parties are communicating via increased traffic at their respective endpoints. This endpoint-linkable timing attack remains a difficulty for all low-latency networks.

3.1.5 ZKS Freedom

Recently, the Canadian company Zero Knowledge Systems has begun the process of building the first mix-net operated for profit, known as Freedom [102]. They have deployed two major systems, one for e-mail and another for TCP/IP. The e-mail system is broadly similar to Mixmaster, and the TCP/IP system similar to Onion Routing.

ZKS's "Freedom 1.0" application is designed to allow users to use a nym to anonymously access web pages, use IRC, etc. The anonymity comes from two aspects: first of all, ZKS maintains what it calls the Freedom Network, which is a series of nodes which route traffic amongst themselves in order to hide the origin and destination of packets, using the normal layered encryption mixnet mechanism. All packets are of the same size. The second aspect of anonymity comes from the fact that clients purchase "tokens" from ZKS, and exchange these token for nyms – supposedly even ZKS isn't able to correlate identities with their use of their nyms.

The Freedom Network looks like it does a good job of actually demonstrating an anonymous mixnet that functions in real-time. The system differs from Onion Routing in several ways.

First of all, the system maintains Network Information Query and Status Servers, which are databases which provide network topology, status, and ratings information. Nodes also query the key servers every hour to maintain fresh public keys for other nodes, then undergo authenticated Diffie-

Hellman key exchange to allow link encryption. This system differs from online inter-node querying that occurs with Onion Routing. Combined with centralized nym servers, time synchronization, and key update/query servers, the Freedom Network is not fully decentralized [37].

Second, the system does not assume uniform traffic distribution, but instead uses a basic "heartbeat" function that limits the amount of inter-node communication. Link padding, cover traffic, and a more robust traffic-shaping algorithm have been planned and discussed, but are currently disabled due to engineering difficulty and load on the servers. ZKS recognizes that statistical traffic analysis is possible [91].

Third, Freedom loses anonymity for the primary reason that it is a commercial network operated for profit. Users must purchase the nyms used in pseudonymous communications. Purchasing is performed out-of-band via an online Web store, through credit-card or cash payments. ZKS uses a protocol of issuing serial numbers, which are reclaimed for nym tokens, which in turn are used to anonymously purchase nyms. However, this system relies on "trusted third party" security: the user must trust that ZKS is not logging IP information or recording serial–token exchanges that would allow them to correlate nyms to users [89].

3.1.6 Web Mixes

Another more recent effort to apply a Mix network to web browsing is due to Federrath et. al.[16] who call their system, appropriately enough, "Web Mixes." From Chaum's mix model, similar to other real-time systems, they use: layered public-key encryption, prevention of replay, constant message length within a certain time period, and reordering outgoing messages.

The Web Mixes system incorporates several new concepts. First, they use an adaptive "chop-and-slice" algorithm that adjusts the length used for all messages between time periods according to the amount of network traffic. Second, dummy messages are sent from user clients as long as the clients are connected to the Mix network. This cover traffic makes it harder for an adversary to perform traffic analysis and determine when a user sends an anonymous message, although the adversary can still tell when a client is connected to the mixnet. Third, Web Mixes attempt to restrict insider and outsider flooding attacks by limited either available bandwidth or the number of used time slices for each user. To do this, users are issued a set number of blind signature tickets for each time slice, which are spent to send anonymous messages. Lastly, this effort includes an attempt to build a statistical model which characterizes the knowledge of an adversary attempting to perform traffic analysis.

3.1.7 Crowds

The Crowds system was proposed and implemented by Michael Reiter and Avi Rubin at T&T Research, and named for collections of users that are used to achieve partial anonymity for Web browsing [85]. A user initially joins some crowd and her system begins acting as a node, or anonymous *jondo*, within that crowd. In order to instantiate communications, the user creates some path through the crowd by a random-walk of *jondos*, in which each *jondo* has some small probability of sending the actual `http` request to the end server. A symmetric key is shared amongst these path *jondos* to ensure link-encryption within a crowd. Once established, this path remains static as long as the user remains a member of that crowd. The Crowds system does not use dynamic path creation so that colluding crowd eavesdroppers are not able to probabilistically determine the initiator (i.e., the actual sender) of requests, given repeated requests through a crowd. The *jondos* in a given path also share a secret *path key*, such that local listeners, not part of the path, only see an encrypted end server address until the request is finally sent off. The Crowds system also includes some optimizations to handle timing attacks against repeated requests, as certain HTML tags cause browsers to automatically issue re-requests.

Similar to other real-time anonymous communication channels (Onion Routing, the Freedom Network, Web Mixes), Crowds is used for senders to communicate with a known destination. The system attempts to achieve sender-anonymity from the receiver and a third-party adversary. Receiver-anonymity is only meant to be protected from adversaries, not from the sender herself.

The Crowds system serves primarily to achieve sender and receiver anonymity from an attacker, not provide unlinkability between the two agents. Due to high availibility of data – real-time access is faster that mix-nets as Crowds does not use public key encryption – an adversary can more easily use traffic analysis or timing attacks. However, Crowds differs from all other systems we have discussed, as users are *members* of the communications channel, rather than merely communicating *through* it. Sender-anonymity is still lost to a local eavesdropper that can observe all communications to and from a node. However, other colluding *jondos* along the sender's path – even the first-hop – cannot expose the sender as originated the message. Reiter and Rubin show that as the number of crowd members goes to infinity, the probable innocence of the last-hop being the sender approaches one.

3.2 Publication Services

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 overview, we examine in detail the amount of anonymity and privacy protection that each project offers.

3.2.1 The Eternity Service

Ross Anderson's paper[cite] on the Eternity Service [5] is the motivation for this entire project. It includes a wonderful vision of how the world might work in the future, in terms of data havens and distributed decentralized data storage. The overall goal is to build a system that provides highly available data: as Anderson phrases, it "[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 would upload 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.

On the other hand, it relies on a stable digital cash scheme, which is certainly not available today. Further, it has a strong correlation between ability to store data into the system and amount of real-world capital available. While our proposal does have a loose correlation between available resources and amount of influence over the servnet, the correlation is not nearly as direct.

There are also a number of contradictions and other 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? Since publishers are anonymous, it would seem that they have no power at all. 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 authors.

Another problem is that there is no consideration at all to maintaining a dynamic list of available servers and allowing servers to smoothly join and leave.

He 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.

3.2.2 The .cz implementation

A team of students at Charles University in Czechoslovakia decided to implement their version [13] of Anderson's idea. They use a mixnet and made overall reasonable design decisions. Tonda Bene wrote his PhD thesis [14] on this design, and provided many more concrete explanations of the details that Anderson skips over in his original paper. However, there are a number of issues that we have with their implementation:

- They implement everything they need by themselves. They do not exist in a vacuum – for example, the world already has 'good enough' loose time synchronization applications such as xntpd, so there is no need to implement a new API and designs. This leads to code bloat and too many layers of abstraction, which makes verifying security very difficult.
- Almost all of their files are binary. It would be easier to examine or modify their configuration files if they were in a human-readable and human-writable format.
- BSD-limited. Their code is not ported to Linux yet, much less other platforms. First of all, the port is probably difficult if they have not yet successfully ported it to another platform. Secondly, a well-designed system (whether in perl, C, or Java) should be extremely portable already – this does not bode well.
- A centralized (single or few) trusted bank system is assumed.

3.2.3 Eternity Usenet

Adam Back proposed [9] a simpler implementation of the Eternity Service, using the existing Usenet infrastructure to distribute the posted files all around the world. This is an excellent idea, but on the other hand this limits the participating servers to systems which already host Usenet news. Further, news administrators much specifically choose to participate in this variant of the eternity service, and so they may well choose not to carry the 'alt' groups that comprise the service.

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.

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.

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. However, he does not appear to provide an explanation of conflict resolution that might arise from two addresses being claimed at the same time – 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.

Also, 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.

Four public-access Eternity Servers are listed at the end of the article; none of these servers is still available. This indicates that active work on Eternity Usenet is not ingoing.

3.2.4 Napster

The Napster service[70] 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, a very 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 presents a very clear argument that the Internet community wants a service like this, at least for music.

3.2.5 Gnutella

Gnutella[33] is a peer-to-peer Napster clone. It was developed by a subsidiary company of AOL, and once it went public it was immediately shut down by AOL (presumably since AOL has interests in not disrupting the music and movie industries). Development is proceeding via a widely scattered group of open-source contributors. Gnutella depends on the "Small Worlds" model to maintain a connected network; see Subsection 8.1.5 for a more detailed description and analysis of this idea.

According to the new developers' web site:

> Gnutella puts a stop to all those shenanigans. When you send a query to the GnutellaNet, there is not much in it that can link that query to you. I'm not saying it's totally impossible to figure out who's searching for what, but it's pretty unlikely, and each time your query is passed, the possibility of discovering who originated that query is reduced exponentially. More on that in the next section.
>
> In short, there is no safer way to search without being watched.
>
> A big however, however. To speed things up, downloads are not anonymous. Well, we have to make compromises. But again, nobody's keeping logs, and nobody's trying to profile you.

There is a strong contradiction between their bold statement about perfection and their warnings that users concerned about maintaining anonymity really should avoid using the system. Behind the media hype, it is clear there are a number of aspects of their protocol [51] that help to reveal identities of users.

The header of a Gnutella packet contains a number of fields. Two of these fields are the '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 given server is generating a query (assuming it is playing by the protocol, which for the vast majority of users is a reasonable assumption). Even if there were no Hops value, the fact that the TTL itself has a default value for most client programs is sufficient to make a server originating a request distinguishable from another server in the system, in the mathematical sense presented in Chapter 2.

Further, the Gnutella network is not so well distributed as they might lead users to believe. While the protocol is designed for a user to set up connections with his 'friends', there is no infrastructure in place for easily building such a set of 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 built not as a flat network but rather as a hierarchical system, as shown in their pictures of the Gnutella network topology [92]. There are a small number of central nodes which would be ideal targets for government agencies or other organizations collecting information about users.

If the TTL and Hops fields aren't enough to reveal identities, it turns out that only the queries have any semblance of anonymity protection. 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.

Sites such as the Gnutella Wall of Shame[25], which attempts to entrap child pornographers using the Gnutella service, show that the direct file-transfer portion of the Gnutella service has been demonstrated to not adequately protect the anonymity of servers or readers.

Gnutella is not designed to be an anonymous communications or publication network. Gnutella is a network designed to provide *availability* for data from one user to the next, and it does a relatively good job of this.

3.2.6 Freenet

The Freenet project [24] is one of the most popular related works. Like Gnutella, Freenet proposes an interconnected network of nodes, each acting as both client and server. 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 complex routing protocol to route queries 'downhill' until they arrive at the desired document.

Freenet is similar to Gnutella in that its main purpose is to provide highly *available* data to its users. There are also a number of differences between Freenet and Gnutella. First of all, Freenet caches data near requestors. Specifically, when a request is answered over a given path, all servers along that path cache that document. This means that future queries for that document will be answered very quickly, assuming the copies of that document haven't expired from the nearby caches. This introduces another important feature: persistence of data. Because nodes cache documents as they are requested, the documents do not disappear from the system when the server offering those documents disappears. This is a key improvement over Napster and Gnutella.

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, it precludes certain uses of the system. For instance, I can see circumstances where a file has a 12 month lifetime but only becomes popular in the last few months of its lifetime. Examples include photos of JFK Jr. saluting his father, or a (timestamped) Idaho phone book that has those ten extra names that the FBI might one day be accused of 'erasing'. Indeed, this is already happening – [76] describes a case where Yugoslav phone books are being collected "to document residency for the close to one million people forced to evacuate Kosovo."

Freenet takes some steps to increase 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 clarify as isolated-server document-anonymity (as opposed to connected-server), by referencing and storing files as $H(name)$ 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*.

However, they have the same flaw with publisher- and reader-anonymity that Gnutella does, due to the presence of the TTL and Depth (comparable to Hops) fields in the Freenet message headers. Because the document requests are also sent through the network (rather than peer-to-peer as they are in Gnutella), there's room for a little bit more anonymity than Gnutella provides. However, nodes nearby a reader still have a greater than even chance of being able to identify that reader. Further, statistical attacks similar to those described in the Crowds [85] paper might work to pinpoint the location of a given reader or publisher; however, the caching does provide some protection against this since the network topology for a given document changes dramatically after each request. This needs 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.

It might even be that a less anonymous system is more likely to be accepted in more parts of the world. This will have to be explored simply by trying it. However, because Freenet provides weaker anonymity than Free Haven, it is currently unsuitable for use by those who are truly concerned about maintaining their privacy, such as political dissidents or other whistleblowers. (See Chapter 4 for more details about these issues.)

3.2.7 Graduated Mirroring

The 'Graduated Mirroring' proposal [87] was introduced by Ron Rivest in response to our initial proposals about a buddy system and other accountability measures that greatly increase the complexity of the system. In short, this idea involves a group of servers, each controlled by a person called the 'manager', which all sign up to receive documents published to the service. Each published document arrives at each server; the manager manually peruses the document and decides how important or valuable he believes it to be. Based on this evaluation, he chooses how many shares of the document to store. For each share, he basically chooses a random share number and generates that share via some information dispersal algorithm – this share generation is implemented by evaluating a polynomial which describes the overall document at a number of random values.

Server managers can modify their support for a given document. If they want to support the document less, they simply throw out some of their shares. If they want to support the document more, they retrieve the document and generate some new shares.

When a document no longer has enough support for readers to be able to reconstruct it, that document has effectively expired; server managers still holding information about that document may then decide to throw away whatever they have about the document – this effectively replaces the notion of a publisher-chosen expiration date with a much simpler notion of server-popularity. This notion of popularity is similar to the notion that Freenet uses, but the duration of a document is based on how much the *servers* like the document rather than on how much the readers like the document.

While this system has some excellent features, including simplicity first and foremost, we have a number of issues with it. The first issue is that it is not what we want to build: the fact that only popular data would get mirrored is counter to our design goal of content-neutrality. We believe that it does not capture the essence of what we want from a data haven.

Indeed, it also goes against our basic assumptions about computing: we have a lot of hardware, and very few people. This trend will get more pronounced as time goes on. Having a person hand-sort and consider each item really cuts down on the number of people who would be willing to host a server. Overall, we believe that paving the way for an automated robust data haven based on privacy of publisher and data is going to have more of an effect in the long run.

3.2.8 Intermemory

The Intermemory Project [36] [22] is an initiative at NEC Research aimed at producing an archival system which makes use of spare space on the Internet. The goal is high availability and high persistence of information. Intermemory uses information dispersal to mitigate the consequences of server failure, and spends much time addressing systems issues such as synchronization of information between many different servers.

Servers join Intermemory and donate a certain amount of space temporarily in return for the right to store some small fraction of that space in the system forever. The "economic" viability of the Intermemory design depends on the assumption that tomorrow's storage will be cheap and plentiful enough to meet the obligations incurred today.

At present, Intermemory exists as a prototype implementation within NEC. The public papers on Intermemory do not even address security, much less anonymity, and it is not clear how the system reacts in the presence of malicious adversaries. We therefore do not formally compare Intermemory to the other systems listed here. Instead, we mention it as an example of a publication and archival system which is designed without the severe constraints necessary to ensure anonymity.

3.2.9 Publius

Publius attacks the problem of anonymous publishing from a different angle. Rather than trying to come up with a routing protocol like Gnutella and Freenet, Publius simply employs some 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. (Publius provides no description of how a directory service might be built, or any other mechanism for remembering URLs after documents are inserted.)

The URL used to retrieve the document specifies all n servers at which the document was stored. This means that to retrieve a given document, the local proxy fetches each document independently, reconstructs the original key K, and then decrypts the document.

The Publius system provides strong publisher-anonymity, because a one-way channel is sufficient for communications to the servers. In addition, because a cryptographically strong secret-sharing protocol is used and each server only receives one share, Publius provides both computational and also information-theoretic isolated-server document-anonymity: a given server is not able to determine anything about a document that it is helping to store.

On the other hand, there are a number of limitations beyond those the authors of the paper enumerate. For instance, 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. Perhaps more importantly, however, there is no support for recognizing misbehaving servers and removing from them the list of available servers.

Another point 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. Providing a mechanism for self-evident share integrity checking might provide significant robustness to the system.

Publius is by far the strongest related work in terms of our notions of anonymity. The paper is very well-written and goes into considerable detail on various attacks and counters to those attacks.

3.2.10 An analysis of anonymity

Many of these related works offer their own variant of 'anonymity' for some of the agents in the system. In this section, we analyze this anonymity that each work provides in the context of the definitions and formalizations proposed in Chapter 2.

This first table provides an overview of the protections for each of the broad categories of anonymity against computationally-limited adversaries. Informally, for polynomially-bounded adversaries who have passive access to some of the edges between agents, a ✓on this table indicates that the adversary has less than a polynomial + $\frac{1}{2}$ chance of correctly guessing the identity of one of the individuals involved in any given transaction.

Project	Publisher	Reader	Server	Document	Query
Gnutella					
FreeNet				✓	
Eternity Usenet	✓				
Publius	✓			✓	
Free Haven	✓	✓	✓	✓	

Table 3.1: Overview: Computational 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.

Freenet achieves document-anonymity because servers are not unable to reverse the hash of the document name to determine the key with which to decrypt the document. However server-anonymity is not provided because given a document, 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, Freenet also fails to achieve publisher- or reader-anonymity.

Eternity Usenet provides publisher anonymity via the use of one-way anonymous remailers. Reader anonymity is not protected, and it is clear that a Usenet service that offers Eternity files is carrying the Eternity feed. Because each downstream host gets its only entire copy of the feed, there is no document-anonymity in Eternity Usenet.

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. Because documents are published to Publius through a one-way anonymous remailer, it provides publisher-anonymity. However, it provides no support for protecting readers, and the servers containing a given file are clearly marked in the URL used for retrieving that file.

Free Haven achieves publisher-anonymity via an anonymous remailer channel. Similarly, reader-anonymity is provided because the responses are directed through a mixnet to a temporary address that the reader provides. Server anonymity is maintained because document requests are performed via broadcast, and the results arrive out of a one-way channel. Free Haven achieves document-anonymity because the document itself is split; assuming a wide enough dispersal of documents, a given server will never see enough shares of a document to be able to reconstruct it. Certainly a server that is not actively trying to rebuild a document (for instance, by doing a request for the other shares) will not have the document available, since it is broken into shares before publication.

This second table provides a more detailed view of the different publishing systems which we examined. There are a number of interesting points that become clearer once we increase the resolution of the table.

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													
FreeNet										✓			
Eternity Usenet	✓	✓	✓										
Publius	✓	✓	✓							✓	✓		
Free Haven	✓	✓	✓	✓		✓	✓			✓			
FH + ideal mix	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			

Table 3.2: Anonymity Properties of Publishing Systems

First of all, we see that the secret sharing algorithm which Publius uses provides a stronger form of document-anonymity, since an isolated server really can learn nothing at all about the contents of a document that it is helping to store. Secondly, we see that those systems which provide publisher anonymity tend to provide a very strong form of it – namely, computational, information-theoretic, and also perfect-forward publisher anonymity. This is because the publishers can make use of one-way communications channels, effectively removing any linkability; removing the need for a reply pseudonym on the mixnet means that there is 'nothing to crack'. Thirdly, we note that Free Haven achieves perfect-forward reader anonymity: this is because readers make up a new pseudonym for every document request, so there is no linkability.

The most important line by far in this table is the last line, though. The fact that it implies that Free Haven is part of the reason it achieves this much anonymity is slightly misleading: it really is the ideal mix which is providing the first nine aspects of anonymity. Assuming a robust ideal mix network like that described in Chapter 2, 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 in fact relegate most of the anonymity to the communications 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 enforced anonymity.

Unfortunately, the design and deployment of such an ideal mix network is a very challenging task, and one for our future works chapter. Until then, we focus on providing increased anonymity with a combination of more conventional mix networks and a publication system tailored for the mixnet.

3.3 Trust Systems

3.3.1 PGP Key Servers

[2] Pretty Good Privacy is a general-use public-key cryptography tool. It provides for encrypted and signed communication. Users exchange their public keys by means of widely-publicized servers:

> Public Key Servers exist for the purpose of making your public key available in a common database where everybody can have access to it for the purpose of encrypting messages to you. While a number of key servers exist, it is only necessary to send your key to one of them. The key server will take care of the job of sending your key to all other known servers.[81]

Each public key on the key servers is signed by people who can verify, by some means, that the person whose name is attached to a key actually controls the associated secret key.

Users who download a public key from the servers set two parameters within their own installation of PGP:

- Confidence that this key represents the user whose name is attached.
- Confidence that this person exercises good judgment in signing other people's keys.

By means of these two values, a network of trust is established. PGP serves as an effective means of communication, and has established a good infrastructure for safely exchanging keys. It fails due to user interface problems: most notably, each key to be accepted as an introducer requires informed attention on the part of the user. As a result, it has not become widespread enough to be generally useful.

3.3.2 Netscape Certificate Authorities

Many commercial web sites wish to ensure that visitors can communicate with them securely. Some also want to strongly verify the identities of their visitors. Certificate Authorities (CAs) exist in a multi-rooted hierarchy. A handful of top-level authorities certify most commercial sites; such sites are then able to establish session keys with their users. Some such sites issue personal certificates to their users.

The flaw here is that users are locked into trusting the established CAs. A user can't decide that he trusts his friends to certify things, but not VeriSign.

[2] This section was written by Brian Sniffen.

3.3.3 AOL Instant Messenger

AIM[8] is a popular messaging client. In order to avoid harassment, users are allowed to file a complaint about those who have sent them messages. Users who accumulate a certain number of complaints per unit time are automatically disconnected from the service. The problem here is that *every* AIM user is therefore trusting *every* other AIM user to act as a censor. AIM has had numerous problems with this complaint feature being used to deny service to various targets.

3.3.4 Internet Relay Chat

The IRC network establishes trust based on a very simple model: IP addresses. A common technique among crackers on IRC is to flood the host of a victim, temporarily knocking him off the network. While the victim is thus distracted, the cracker spoofs packets from the victim to an IRC server, giving privileges to himself and removing them from the victim. When the victim returns to the network, he is now unprivileged, and is subject to further attacks by the now-privileged cracker.

Clearly, non-cryptographic trust models are not useful against modern adversaries.

3.3.5 Mobile Agents for Network Trust

MANET[50] is a DARPA project to produce "a compromise-tolerant structure for information gathering." The motivation is to create a system whereby untrusted networks can cooperate to fight against "mobile adversaries": adversaries who move from one network to another. The MANET project attempts to avoid the problem of corrupted servers by requiring several servers to weigh in on a subject with direct evidence before action is taken. This approach is, even in the eyes of the MANET authors, somewhat naïve. MANET relies on correct execution of mobile code

3.3.6 Publius

The Publius system[100] has an implicit trust model entirely divorced from reality: there is a static set of servers, all of which are widely and publicly known. Documents are staticly stored on several of these servers, having been split using Shamir's Secret Sharing Algorithm. If one of these servers is corrupted, it is assumed it will be replaced.

It is possible to turn Publius into an informal but workable system with a very small amount of work: if the servers are all publicly known, then an informal trust network can be put into place. Create a discussion forum, called perhaps `alt.anonymous.publius`. Sites which wish to become servers advertise here; if users discover that a server has been corrupted, they can denounce it in that forum. Of course, this system provides no formality or assurances whatsoever.

Chapter 4

Applications, Legalities, and Ethics

4.1 Introduction and Assumptions

In the world of academics, it is very common to develop new and interesting technologies without regard for the moral and ethical implications of these technologies. Indeed, it is very common for physicists or theoreticians to spend months on a given problem without even considering applications of their solution – it is their job to solve the problem, and somebody else's job to come up with ways of making the answer useful in a wider context. In the case of Free Haven, it is vitally important that we consider its uses and applications *before* deploying the service. There are a number of other projects developing networks similar to Free Haven, but these related works do not offer the same levels of anonymity as Free Haven. It is precisely this lack of accountability that makes Free Haven so powerful and so useful, but at the same time this lack of accountability makes Free Haven potentially very dangerous. There will be no way of policing the content on the network, nor will there be any way of shutting down the service (or even reliably detecting if it's still deployed!). In this context, we have to consider and compare uses which we consider legitimate, as well as uses which we consider illegitimate.

Before we start enumerating case studies and examples, however, we must make explicit our underlying assumption and belief: the rights and liberties of the individual are the fundamental building blocks of society. As Jefferson described, "governments are instituted among men, deriving their just powers from the consent of the governed." [49] Indeed, as Kropotkin might argue, freedom involves both 'freedom from' and 'freedom to'. " 'Freedom from' signifies not being subject to domination, exploitation, coercive authority, repression, or other forms of degradation and humiliation. 'Freedom to' means being able to develop and express one's abilities, talents, and potentials to the fullest possible extent compatible with the maximum freedom of others." [44] Supporting and maintaining freedoms for the individual, including self-management, responsibility, and independence, is one of the most important causes we can hope to undertake.

4.2 Anonymous speech

> *No long string of citations is necessary to find that the public interest weighs in favor of having access to a free flow of constitutionally protected speech.*
>
> — *Reno, 929 F. Supp. at 851.*

4.2.1 Overview

Anonymous speech has been a hotly disputed topic in the United States since the very creation of our government. The Federalist Papers, published during the original constitutional debates, were published anonymously[75]. Thomas Paine wrote his famous pamphlet[79] entitled "Common Sense" under a pseudonym. Indeed, "[a]nonymous pamphlets, leaflets, brochures and even books have played an important role in the progress of mankind.... Persecuted groups and sects from time to time throughout history have been able to criticize oppressive practices and laws either anonymously or not at all." [96]

One of the features of the Free Haven project is that it provides a tool that enables individuals around the world to engage in anonymous speech. By publishing a document on the Free Haven service, anyone can offer political or other speech on the internet without providing any accountability as to source or authorship. The following examples of case law in the United States explore some of the issues integral to anonymous speech.

4.2.2 A Case Study: ACLU of Georgia v. Miller

On September 24, 1996, the American Civil Liberties Union filed a lawsuit on behalf of 14 plaintiffs, including the AIDS Survival Project, the Atlanta Freethought Society, and Atlanta Veterans Alliance, against the state of Georgia based on Act No. 1029 (Ga. Laws 1996, p. 1505), a law which effectively outlaws anonymity on the Internet. In particular, one of the aspects of this law made it illegal "knowingly to transmit any data through a computer network ...for the purpose of setting up, maintaining, operating, or exchanging data with an electronic mailbox, home page, or any other electronic information storage bank or point of access to electronic information if such data uses any individual name ... to falsely identify the person."

The original intent of this act was to prevent online fraud. Decreased fraud would make the internet and other online resources (such as AOL or local bulletin board systems) safer and more convenient for individuals and businesses. However, the court found that the law was far too vague and broad in its scope, covering transmissions which were not deceiving or fraudulent and in fact had no intention of being either ("intent to defraud" and "intent to deceive" appear nowhere in the act). The defendants said that it was also designed against misappropriation of another's identity, but this too was not specified in the act at all. The above omissions do not occur in other Georgia legislation meant for purposes of criminalizing fraud[1].

In short, the court found that the wording of the law did not at all match the intent of the law. "The act prohibits such protected speech as the use of false identification to avoid social ostracism, to prevent discrimination and harassment, and to protect privacy... it operates unconstitutionally for a substantial category of the speakers it covers." [4] Indeed, what this act *did* criminalize was shown to be ambiguous – criminal statutes must "define the criminal offense with sufficient definiteness that ordinary people can understand what conduct is prohibited and in a manner that does not encourage arbitrary and discriminatory enforcement." [54]

The court closed with a number of interesting quotes characterizing the history and importance of free speech. The US Supreme Court has held that "the loss of First Amendment Freedoms, for even minimal periods of time, unquestionably constitutes irreparable injury" [31] and the ACLU was able to demonstrate that this law threatened 'irreparable injury' to citizens and organizations alike.

The decision was reached in favor of the ACLU on Friday, June 20, 1997: the law was declared unconstitutional.

4.2.3 Analysis

The example of ACLU v. Miller is not just a case of corrupt or ambiguous laws going awry. It is a case of the fundamental freedoms of individuals being infringed by the state. Whether the original intent of the law was to prevent online fraud, the law provided too much flexibility to those who would actually carry out the law to interpret it in whatever manner was most convenient at the time.

"The Court recognized that anonymity is the passport for entry into cyberspace for many persons," said Gerald Weber, Legal Director of the ACLU of Georgia. "Without anonymity, victims of domestic violence, persons in Alcoholics Anonymous, people with AIDS and so many others would fear using the Internet to seek information and support." [23] Another example specific to this case was the Atlanta Veterans Alliance, an organization that serves the needs of gay, lesbian and bisexual military veterans. Not only do members of the AVA employ anonymity and pseudonymity on the internet to avoid harassment and discrimination, but many of them are currently in active military duty, and disclosing their real names during their discussions would mean that they might lose their jobs.

[1] See, e.g., O.C.G.A. 10-1-453, 16-9-1(a), 16-9-2, and 16-8-3.

4.2.4 A Case Study: Lamont v. Postmaster General

Indeed, one of the more subtle aspects of the ACLU v. Miller case is that it addresses anonymous *receipt* of information as well. By referencing Lamont v. Postmaster General (381 U.S. 301 (1965)) [56], the plaintiffs compare the act to a statute that has been influential in free speech Supreme Court cases over the past three decades.

The Postal Service and Federal Employees Salary Act of 1962 required the Postmaster General to monitor mail coming into the United States from certain foreign countries. If the mail was deemed to be 'communist political propaganda', the Post Office would detain the mail and send a note to the addressee. This note would describe the mail in question, and indicate that the addressee should fill out the form and return it if he wanted to receive the communist propaganda. If he didn't fill out the form and return it within twenty days, then the Post Office would conclude that he was not interested in that publication or any further similar publications, and destroy them.

The Post Office implemented the statute by maintaining 10 or 11 screening points around the country, through which all mail from suspicious foreign countries was routed [56]. Customs officials would examine each item of mail, and determine whether it was communist political propaganda. In this case, "[t]he term 'political propaganda' includes any oral, visual, graphic, written, pictorial, or other communication or expression by any person (1) which is reasonably adapted to, or which the person disseminating the same believes will, or which he intends to, prevail upon, indoctrinate, convert, induce, or in any other way influence a recipient or any section of the public within the United States with reference to the political or public interests, policies, or relations of a government of a foreign country or a foreign political party or with reference to the foreign policies of the United States or promote in the United States racial, religious, or social dissensions, or (2) which advocates, advises, instigates, or promotes any racial, social, political, or religious disorder, civil riot, or other conflict involving the use of force or violence in any other American republic or the overthrow of any government or political subdivision of any other American republic by any means involving the use of force or violence." [98] Communist political propaganda is political propaganda which is "issued by or on behalf of any country with respect to which there is in effect a suspension or withdrawal of tariff concessions or from which foreign assistance is withheld pursuant to certain specified statutes." [99] If these Customs officials determined that the mail in question was communist political propaganda, they would send the note instead.

This law effectively requires citizens receiving material which might be communist political propaganda to actively choose to sign their names onto a list which the federal government keeps. Since the card has a checkbox for whether the addressee wishes to receive similar mails in the future, this list is exactly a list of citizens who are interested in reading dissident materials.

Two separate individuals, Dr. Corliss Lamont of New York, and Mr. Heilberg of California, filed complaints in 1963 against this law in their respective District Courts when they were sent a note about material addressed to them. The District Court in New York dismissed the claim as moot, since Lamont had the opportunity to check the box indicating he wanted all further mails to go through unhindered – since he had this opportunity, his First Amendment rights were not being abridged. The California District Court ruled in the opposite direction, claiming that requiring the addressees to actively request mail infringed on their First Amendment rights.

The case was made stronger on March 15, 1965, when the government terminated its practice of keeping a list of citizens who wanted to receive communist political propaganda. Because the Post Office wasn't keeping this list, they required addressees to actively request the delivery of each such piece of mail. With this new twist, the US Supreme Court found the law to be unconstitutional on May 24, 1965, on the grounds that requiring addressees to respond separately for each piece of mail constituted an undue burden on their constitutional right to receive mail. Justice Douglas explains that "[t]his requirement is almost certain to have a deterrent effect, especially as respects those who have sensitive positions. Their livelihood may be dependent on a security clearance. Public officials, like schoolteachers who have no tenure, might think they would invite disaster if they read what the Federal Government says contains the seeds of treason. Apart from them, any addressee is likely to feel some inhibition in sending for literature which federal officials have condemned as 'communist political propaganda.' The regime of this Act is at war with the "uninhibited, robust, and wide-open" [72] debate and discussion that are contemplated by the First Amendment." [56]

Indeed, the Court goes on to address the specific parts of this case that relate to anonymous access to publications – while they never use the word anonymous, they draw a distinction between a passive receipt of the material, and an active request for receiving the material. Justice Brennan argues in the same text: "It is true that the First Amendment contains no specific guarantee of access to publications. However, the protection of the Bill of Rights goes beyond the specific guarantees to protect from congressional abridgment those equally fundamental personal rights necessary to make the express guarantees fully meaningful. ...I think the right to receive publications is such a fundamental right. The dissemination of ideas can accomplish nothing if otherwise willing addressees are not free to receive and consider them. It would be a barren marketplace of ideas that had only sellers and no buyers."

This notion can be extended to the arena of the internet, in terms both of receiving email and of receiving data from a web page. In the former case – passive receipt of data – it is more traditionally clear that the right to receive email should be protected under our Constitution. The latter case, however, presents more confusion: should this 'active' receipt of information receive the same protections as passive receipt of information? We as members of the Free Haven project believe the answer is yes.

4.3 Anonymous Publication

> *If there is any principle of the Constitution that more imperatively calls for attachment than any other it is the principle of free thought – not free thought for those who agree with us but freedom for the thought that we hate.*
>
> — *Oliver Wendell Holmes, Jr.*

The previous two case studies deal in large part with the issue of freedom of speech and anonymous speech, both from the perspective of making the speech and also from the perspective of having the right to receive speech from others in an unhindered and uninhibited fashion.

Our third case study explores the issue specifically of anonymous *publication*, to show that freedom of publication, and indeed anonymous publication, is protected by the First Amendment.

4.3.1 A Case Study: McIntyre v. Ohio Elections Commission

On April 27, 1988, Margaret McIntyre distributed leaflets to persons attending a public meeting at the Blendon Middle School in Westerville, Ohio [63]. The meeting was in part to discuss a proposed school tax levy, and McIntyre was distributing pamphlets signed 'Concerned Parents and Taxpayers' in opposition to this proposed levy. While Mrs. McIntyre was distributing her handbills, an official of the school district warned her that distributing the pamphlets without signing them with her name was a violation of the Ohio Elections Commission.

Specifically, Ohio Code 3599.09(A) prevented distribution of campaign literature not containing the name and address of the person issuing such literature. Margaret McIntyre challenged this law upon being fined $100 for distributing her pamphlets. The fine was reversed by the Common Pleas court, but upheld by the Ohio Court of Appeals and Ohio Supreme Court, which held that the law was necessary to prevent "fraud, libel, or false advertising" and that the requirement for producers of campaign literature to identify themselves "neither impacts the content of their message nor significantly burdens their ability to have it disseminated. This burden is more than counterbalanced by the state interest in providing the voters to whom the message is directed with a mechanism by which they may better evaluate its validity." Ohio did not suggest that all anonymous publications should or could be outlawed, simply that this was a reasonable electoral regulation.

The U.S. Supreme Court reversed the fine. Justice Stevens delivered the majority opinion: he claimed "an author's decision to remain anonymous, like other decisions concerning omissions or additions to the content of a publication, is an aspect of the freedom of speech protected by the First Amendment. The freedom to publish anonymously extends beyond the literary realm." He also pointed out that "in the case of a handbill written by a private citizen who is not known to the recipient, the name and address of the author adds little, if anything, to the reader's ability to evaluate the document's message."

Talley v. California (1960) [96] was heavily cited. In this case, Talley violated a Los Angeles ordinance against distribution of unsigned handbills in distributing leaflets urging boycott of certain merchants practicing discriminatory employment practices. The ordinance was declared unconstitutional. Stevens held that this could be extended to anonymity in advocacy of any political cause, up to and including secret ballot. Ohio's regulation "does not control the mechanics of the electoral process. It is a regulation of pure speech." Political speech was also held to be at the core of the First Amendment.

Stevens states that "[a]nonymity is a shield from the tyranny of the majority", and goes on to describe the purpose of the First Amendment: "to protect unpopular individuals from retaliation – and their ideas from suppression – at the hand of an intolerant society. The right to remain anonymous may be abused when it shields fraudulent conduct. But political speech by its nature will sometimes have unpalatable consequences, and in general, our society accords greater weight to the value of free speech than to the dangers of its misuse." He cites [3], a case in which five plaintiffs are accused and convicted of distributing during wartime pamphlets that urge American citizens to cease production of ordnance and ammunition; the Russian-born plaintiffs argue that citizens should rise up against capitalism, and also that they should realize that the munitions they are building are being used to kill their families in Russia. Justice Holmes writes a strong dissenting opinion, declaring that "Only the emergency that makes it immediately dangerous to leave the correction of evil counsels to time warrants making any exception to the sweeping command, 'Congress shall make no law abridging the freedom of speech.' "

Justice Thomas asks whether " 'freedom of speech, or of the press' as originally understood, protected anonymous political leafletting" and concludes it did. "Regardless of whether one designates the right involved here as one of press or one of speech, however, it makes little difference in terms of our analysis, which seeks to determine only whether the First Amendment, as originally understood, protects anonymous writing." He cites Zenger; opposition in 1779 to identifying Leonidas, writer of articles critical of Congress's economic policies; and other cases of liberty of the press being linked to anonymity. He mentions the Anti-Federalists, who drove the Bill of Rights, and comments that they often criticized attacks on anonymous publishing. He describes the national outcry (centered in Philadelphia) against two Boston papers that refused to print anonymous articles: "The understanding described above, however, when viewed in light of the Framers' universal practice of publishing anonymous articles and pamphlets, indicates that the Framers shared the belief that such activity was firmly part of the freedom of the press." He goes on to cite many instances of further anonymous political speech after the revolutionary period, arguing that anonymous speech is also suitable and reasonable for conditions other than those extremely unusual circumstances.

While the support for anonymous political speech is clear, and the decision of the Court was seven to two, there is nonetheless some interesting material in the dissenting opinions.

Justice Scalia disputes Thomas's use of the historical record, stating that anonymous electioneering was not held to be a violation of freedom of speech or the press, and that regulation of the electoral process was never an issue until the late 1800s, and lack of regulation before then does not render later laws unconstitutional. Laws against anonymous political pamphleteering have existed in most states (including Ohio) since the end of World War I and form a better historical record than revolutionary essays. He goes on to then examine the right to anonymity. "Several of our cases have held that in peculiar circumstances the compelled disclosure of a person's identity would unconstitutionally deter the exercise of First Amendment associational rights. [list of cases] But those cases did not acknowledge any general right to anonymity ... rather, they recognized a right to an exemption from otherwise valid disclosure requirements on the part of someone who could show a 'reasonable probability' that the compelled disclosure would result in 'threats, harassment, or reprisals from either Government officials or private parties.' ... Anonymity can still be enjoyed by those who require it."

Scalia argues that the decision in the McIntyre case was unusual and differs significantly from previous case law: "The existence of a generalized right of anonymity in speech was rejected by this Court in Lewis Publishing Co. v. Morgan, 229 U. S. 288 (1913), which held that newspapers desiring the privilege of second class postage could be required to provide to the Postmaster General, and to publish, a statement of the names and addresses of their editors, publishers, business managers and owners. We rejected the argument that the First Amendment forbade the requirement of such disclosure. Id., at 299. The provision that gave rise to that case still exists, see 39 U. S. C. 3685, and is still enforced by the Postal Service. It is one of several federal laws seemingly invalidated by today's opinion."

He goes on to cite Ginsburg's concurring opinion, and wonders at the extensions of "the Court's unprecedented protection of anonymous speech" including parade permits issued to groups who refuse to provide their identity, anonymous sponsorship of theatre presentations at a city-owned theatre or of speeches at public universities, anonymous letters to the editor in government publications, and other "silliness that follows upon a generalized right to anonymous speech." He points out that a number of foreign democracies, including Australia, Canada, and England, all have prohibitions upon anonymous campaigning.[2]

In closing, Scalia explains: "I do not know where the Court derives its perception that 'anonymous pamphleteering is not a pernicious, fraudulent practice, but an honorable tradition of advocacy and of dissent.' I can imagine no reason why an anonymous leaflet is any more honorable, as a general matter, than an anonymous phone call or an anonymous letter. It facilitates wrong by eliminating

[2] See, e.g., Commonwealth Electoral Act 1918, 328 (Australia); Canada Elections Act, R.S.C., ch. E-2, 261 (1985); Representation of the People Act, 1983, 110 (England).

accountability, which is ordinarily the very purpose of the anonymity."

4.3.2 Analysis

Clearly, there are strong arguments on either side of this issue. The overall opinion was that *content-based* restrictions on whether a given publication can be distributed anonymously conflict with the First Amendment. Furthermore, the justification that Ohio provided for needing protection against anonymous electioneering does not make sense for this particular situation: in this case, knowing that Mrs. McIntyre was the author of the pamphlet in question would not provide readers with the ability to "better evaluate its validity".

It is also worthwhile to note that much of the dissenting opinion focused not on the issue of anonymity of speech itself, but on the issue of whether allowing anonymous political speech was more valuable or less valuable than protecting the elections process. For instance, while Canada may prohibit anonymous campaigning, section 14.1 of the Canadian Copyright Act [30] states that the author has the right "to be associated with the work as its author by name or under a pseudonym and the right to remain anonymous".

4.4 Legal, yes – but Moral?

> *It is unfortunate that the efforts of mankind to recover the freedom of which they have been so long deprived, will be accompanied with violence, with errors, and even with crimes. But while we weep over the means, we must pray for the end.*
>
> — *Thomas Jefferson to Francois D'Ivernois, 1795*

It is clear both from the United States Constitution, and also from the case law described above and held by the US Supreme Court, that anonymous publication is a legal and protected right for US citizens. However, the legal support in the United States – or even in other countries – for anonymous publication is not the only issue we as developers of Free Haven must consider: even if we are legally allowed to deploy a Free Haven service, is it morally a good idea? We have the power to make this choice now, but after we deploy the service we will not have the power to undo our actions.

First of all, there are a myriad of applications for the system which we consider 'bad uses'. These can be broken down into several categories, based on the type of use or offense involved.

4.4.1 Problems with Anonymous Speech

These are issues which generally come up in the context of all anonymous speech or communication systems, rather than specifically in the context of anonymous publication systems. They include:

- **Death threats:** users may be able to make death threats without accountability.
- **Terrorism communications:** users may be able to coordinate and conspire to plan terrorist activities against the state or other organizations or individuals.
- **Kidnapping communications:** similarly, users might conspire and coordinate to plan kidnappings or other illegal actions.
- **Spam:** users might make use of the anonymous channel to spam victims with targeted advertisements or other text.
- **Harassment:** as opposed to targeted spam, stalkers might make directed communications intended to embarrass, defame, or threaten.
- **Blackmail:** users might publish material without disclosing the key, and then threaten to publicize the location of the material.

These issues are addressed in a broader scope by other organizations, such as the ACLU (witty reference here), the Center for National Security Studies (their papers include "The FBI's Domestic Counterterrorism Program"[34], a description of the FBI's current capabilities to combat terrorism), the Cato Institute (their papers include "Nameless in Cyberspace: Anonymity on the Internet"[46], a briefing paper addressing anonymous speech on the internet), Amnesty International, the EFF, XXX. Since these issues are argued in the broader context of free and anonymous speech in general rather than anonymous publication specifically, we consider them outside the scope of this document, and refer the reader to the literature from these organizations. One useful survey paper is the one by Rigby [86].

4.4.2 How reputable is anonymous speech anyway?

Seek not to know who said this or that, but take note of what has been said.
— Thomas Kempis, Of the Imitation of Christ

Another topic that is related to free and anonymous speech is the question of whether making speech anonymous decreases the credibility of its content. On the one hand, cases like the McIntyre v. Ohio Elections Committee findings show that there are in fact situations where knowing the author of a statement doesn't seem to add or detract at all from the content. On the other side of the spectrum, the 'Net Anonymity FAQ' [6] includes a variety of comments pointing out that anonymous postings often have very little useful content. It includes an insightful comment: "I think anonymous posts do help in focusing our attention on the content of one's message. Sure lot of anonymous posts are abusive or frivolous but in most cases these are by users who find the anon facility novel. Once the novelty wears off they are stopping their pranks."

4.4.3 Copyright, Patent, and Trade Secrets

Another issue with substantial impact on society and economics is the fact that anonymous communication, and indeed also anonymous publication, can be used to share documents in a manner that violates copyright or patent laws, or exposes trade secrets. The recent issues with decss and cphack as examples of trade secrets getting published on the internet, as well as the earlier example of the distribution of pgp being restricted due to patent issues, show that there are a number of controversial issues with these laws. On the other hand, more clear-cut cases such as copying a band's music in violation of the band's copyright and wishes are very prevalent, and becoming even more common.

This issue is a real problem. Related projects like Napster are beginning to realize the power of established organizations such as the Record Industry Association of America (RIAA) and the Motion Picture Association of America (MPAA) to protect their assets and lobby Congress for protective laws. The continuing trend towards high-bandwidth internet connections for individual people will exacerbate this conflict: eventually this empowerment of individuals will force a dramatic change in the current copyright system, to enable it to handle this new paradigm of global connectedness.

In the case of music, the RIAA is fighting to maintain the status quo based on a principle of intellectual property and copyrights developed centuries ago[18], which many argue is no longer applicable to today's information-age society. This system probably cannot last, and bands are going to have to adapt to other mechanisms for making money, such as live performances.

Such artists as Billy Idol, Public Enemy, and the Beastie Boys have already attempted to give out MP3s for free to increase their publicity. John Perry Barlow, the lyricist of the Grateful Dead, has published a detailed essay[10] entitled "The Economy of Ideas", which describes why "everything we know about intellectual property is wrong" from the point of view of his band.

In late 1997, Clinton signed an act called the No Electronic Theft (NET) Act, which made it a felony to copy copyrighted materials. It appears that the main change in the law that they had made was changing the definition of financial gain: "the term 'financial gain' includes receipt of anything of value, including copyrighted materials." [71] The Act suggested five year jail sentences for those found distributing more than ten copyrighted works in a given amount of time. Apparently the RIAA and the rest of the music industry have enough influence to convince the Clinton administration that it should enact much more severe laws against intellectual property violators.

But the situation is not hopeless. Alternate copyright systems have been proposed, such as Bruce Schneier and John Kelsey's 'Street Performer Protocol' [90]They describe this protocol as "an electronic-commerce mechanism to facilitate the private financing of public works. Using this protocol, people would place donations in escrow, to be released to an author in the event that the promised work is put in the public domain. This protocol has the potential to fund alternative or 'marginal' works."

In other words, this Street Performer Protocol (or more likely, some future variant of it) could be used to help shift the emphasis away from *purchasing* information and more towards paying the designer for the act of *creating* the information in the first place.

Overall, we consider the fact that Free Haven might be used to further violate copyright and patents laws to be an unfortunate consequence of deploying the system. We believe this is a strong argument against developing a system like this.

4.4.4 More Porn on the Net

Another unfortunate consequence of deploying a service like Free Haven is that it may well speed the proliferation of pornography on the internet. Just as the pornography industry was influential in accelerating the growth of the technologies behind videotapes, many people have commented [reference?] that it is now a major driving force behind increase in hard drive capacity and internet bandwidth. While the pornography industry is in fact of questionable moral usefulness to our society, there can be no doubt that it is often a major factor in new technologies.

On the other hand, most of the porn industry bases its profits on being able to charge subscribers and meter distribution. All of this process is legal and already in place on the internet. Precisely because Free Haven provides such a high level of anonymity, it doesn't provide sufficient accountability for companies to conveniently sell their data in a trackable manner. Free Haven is simply not a very hospitable environment for organizations trying to control or in any way limit access to information. Thus, the porn *industry* as a whole will probably largely ignore it.

But the fact that it provides such a high degree of anonymity leads to a few other possibilities: although companies won't be able to make easy money from it, individuals desiring greater anonymity may make use of the system to distribute illegal media such as child pornography or snuff films in an untraceable manner.

Just as in the case of copyright and patents, we believe this is an unfortunate consequence of the system, and believe it is a strong argument against developing a system like Free Haven.

4.4.5 Jurisdiction and Jurisdictional Arbitrage

Eric Hughes, cofounder of the cypherpunks, describes jurisdictional arbitrage as "moving an action from one country to another country to take advantage of a different law or regulation there." [57] This is a very broad topic, but one that is well worth addressing, since it provides a very good example of a situation where the internet can provide decentralized community solutions that entirely avoid the issue of a given nation's legislation or jurisdiction.

The case of the Teale couple in Canada provides a good example. In this case, a husband and wife pair were suspected of several acts of particularly gruesome manslaughter. The wife pleaded guilty and provided evidence; the Judge banned all those who were present at the trial, including the press, from publishing any evidence or details on the murders, "in order to insure that Paul Teale receives a fair trial" [64]. (Actually, Paul Teale was known as Paul Bernardo before the arrest; many Canadians know of him by the latter name.) Despite the fact that the Canadian court issued the gag order, U.S. papers continued to print news concerning the case [94]. Indeed, the Canadian border officials even went to the point of "stopping trucks carrying U.S. newspapers to keep Canadians from learning what their neighbors to the south know about the case." [73]

Beyond the fact that the United States press was rabid enough to ignore a Court decision from Canada, there's another facet to this case: the internet played a very important role in distributing information about the case. Soon after the court order, a pair of university students in Canada set up a newsgroup by the ironic name of `alt.fan.karla-homolka` (Paul's wife's name). When this newsgroup was banned by a number of Canadian universities, two new newsgroups sprang into existence: `alt.pub-ban` and `alt.pub-ban.homolka`. According to Wired Magazine, "One Net dweller jokingly proposed the ideal tactic: 'The solution is obvious. Take the discussion to `rec.sport.hockey`. You silly Canadians would never ban that group.' " [19] While intended as a joke, this comment makes a very clear statement: banning the internet just isn't feasible.

Douglas Barnes, a member of the Austin Cypherpunks, provides an excellent overview [12] of various classes of jurisdiction, including some surprising examples such as 'effects doctrine', wherein the crime can be prosecuted based on some location that felt the *effects* of the action. In the case of United States v. Aluminum Company of America (ALCOA) [97], the US established precedent for this 'effects doctrine' by successfully bringing an anti-trust action against this foreign company based on the effects on the US of their restraint of trade.

There are a number of legal issues and tricky points behind the concept of jurisdiction, but once again the legal aspects are not the entire picture: is it moral to circumvent a court's decision of information blackout? Who are we to make that decision, or enable others to make that decision?

We believe the short answer to this is 'it depends'. The longer answer is that we feel that it is crucial for this option to be available: while it may not be the appropriate answer for all situations, it may well be a reasonable response for some circumstances.

4.4.6 Slander is Forever

Once published in Free Haven, a document will persist at least until it reaches its expiration timestamp. Since there is currently no mechanism for revocation or unpublishing, any statements made at one point will potentially last for a very long time. This permanence of speech can have negative consequences. For instance, if a document were published in Free Haven claiming that a given person is a police informer, then there is no way that the author can unpublish this statement: it lives until it reaches the expiration date chosen by the publisher. Since the document is anonymous, there is no entity or organization to attack or sue. Another scenario to consider is one where a radical extremist posts a flaming attack on the United States government and attaches his name to it, and then fifteen years down the road is refused from a job at the FBI based on his past life.

The 'right' to force somebody who slanders you to take back their statements is completely removed in Free Haven. On the other hand, this is precisely what Free Haven is built for. We cannot be responsible for people who use it without thinking first. The decision to publish a document in a persistent global forum should be a carefully considered process, weighing the pros of getting the word out against the corresponding repercussions.

4.4.7 Is content-neutral wise?

One of the features that sets Free Haven apart from related works such as Freenet is the fact that Free Haven maintains an entirely content-neutral approach to the data stored in the system. 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.

There are a number of arguments against being content-neutral. Freenet pays attention to the popularity of documents, and provides greater availability and persistence to those documents which are accessed more frequently. This provides more efficient use and distribution of Freenet resources around the world.

Another argument, based on morality rather than practicality, is the idea that server administrators should exercise good moral judgment and decide on a per-document basis which documents should be stored on their server and which are inappropriate. A system like this would clearly not be vulnerable to people sneaking immoral or illegal material onto it.

On the other hand, such a system requires a lot of resources on the part of each server administrator. Indeed, as the trend towards increasing computational capacity and ability continues, we should be looking for solutions that minimize human involvement. A solution could be found such that the server administrator automates the content filtering, but examples indicating this hope might be naive include the recent issue with AOL's 'youth filter' blocking the Democratic website but allowing the Republican site through [58].

From a legal perspective, server operators who are unaware of the particular content that they're hosting, and have reasonable cause to expect that the content is legal, are not responsible. On the other hand, if the operator has the capacity to glance over the data before accepting it, they may be considered to have affirmatively accepted the data and are therefore responsible for its content.

Until the Communications Decency Act, this state of affairs roughly described case law in the

United States. In the case of *Cubby v. Compuserve*,[26] the online service CompuServe was found not liable for a post on its message boards on the grounds that it merely provided a forum for expression and had no prior control over the messages posted. By contrast, the online service Prodigy was found liable in *Stratton vs. Prodigy* for a libelous posting, because the court found that by engaging in moderation of postings, the service became a "publisher" of the postings instead of a mere carrier[93]. The Decency Act changed this by creating statutory protection against liability for ISPs or online services who wish to moderate content passing through their lines; this part of the act, unlike the more famous "indecency" provisions, was not invalidated by the Supreme Court of the United States[69].

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 that they carry over their systems.

Finally, 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, it is designed to be a private low-profile storage system. Requiring operators to read through publication 'submissions' runs counter to this goal.

4.4.8 Personal uses

As an anonymous storage service, Free Haven provides a number of functions that are useful to individual users. For instance, users or companies might employ the system as the ultimate redundant backup server, encrypting their data and then spreading it anonymously and robustly across the world.

More generally, individuals who wish to retain possession of data but not be physically associated with it for a certain period of time can benefit greatly from Free Haven. Perhaps the next person targeted by the Church of Scientology could have an available mechanism for removing the data from their person in a manner that makes it convenient to both offload and later retrieve. Amnesty International workers might benefit greatly from the ability to generate an address list as they tour southeast Asia, and have a convenient way to retrieve it once they return to more hospitable areas. Arming individuals with the ability to defend themselves against larger potentially hostile groups is an important end in itself.

4.4.9 Is privacy bad?

A common belief is that those who choose to communicate via strong cryptography or other cryptographic protections on privacy have 'something to hide', and that normal upright citizens have no need for cryptography. Similarly, some believe that people who speak anonymously are somehow ashamed of the actions that they take behind the shield of anonymity.

However, this idea that only shame generates a desire for privacy is a very narrow view. Privacy in our ordinary lives is something that we take for granted – would you be willing to publish your tax return worldwide? What if your neighbor published the contents of your garbage on the evening news? Is using cryptography to achieve privacy in online activities really any different? If online privacy through cryptography is in fact still distinct from offline privacy through more conventional means, we believe the trend is moving strongly in favor of merging them into a single concept.

Frequently, people respond to these disturbing possibilities by denying that they could happen to them: after all, safety in numbers should be a sufficient defense against any other individual or organization wanting to collect information about 'typical' citizens. However, this defense is terrifyingly naive, considering the explosive growth of storage and data warehousing and retrieval technologies in the past few years. Companies ranging from Doubleclick to Amazon collect a startlingly wide array of information about potential customers, in the name of directed advertising. Insurance companies might cross-reference with Amazon to determine whether their customers have purchased books on car racing. Divorce attorneys might cross-reference with credit card companies to identify and offer services to persons who have recently paid for hotel rooms or purchased other paraphernalia associated with extramarital affairs. Employers might cross-reference with medical histories to determine HIV status or even genetic predispositions. With the continued rise in electronic commerce and global internetworking, extensive databases of personal profiles on every person on Earth are visible on the horizon.

Confining to the police or other intelligence agencies the ability to collect, correlate, or make use of this information does not help much. Building correlations between disparate data sets is a tricky task, and the people asking the questions are almost never the ones building the databases or doing the queries. Because of this, they don't understand the limitations of the data they have available. Government divisions may well be required to make a certain quota of profiles matching certain constraints, such as 'pedophile' or 'drug dealer'. If time is short, budgets are tight, and relaxing some of the query constraints is much easier and cheaper than collecting or verifying more data, the choice seems clear. The result of this is that ordinary innocent citizens will get targeted as 'suspicious' for one reason or another. The transition from surveillance state to police state may well be a very subtle one.

4.4.10 Tool for Political Dissidents

One of the most important goals we could hope to achieve is to aid political dissidents in spreading their statements. The Rewebber document by Goldberg and Wagner enumerates a wide variety of causes through the past few centuries for which publication was a driving force[38]:

- The Protestant Reformation was greatly aided by the invention of the printing press, which enabled widespread distribution of many copies of the Bible
- the French "Voice of the Resistance" used nightly radio broadcasts from constantly-changing temporary locations to reach the people during the German occupation
- the USA used high-power radio stations such as Radio Free Europe during the cold war to combat censorship behind the Iron Curtain
- in past years, banned information was copied through underground channels from person to person in the Soviet Union, in a process known as samizdat (which is Russian for "self-publishing")
- when the Serbian government began jamming the Belgrade independent radio station during the Serb-Croat war, Serbian students used the World Wide Web to mirror broadcasts and combat the government's censorship

Indeed, we might also include

- Saudi dissidents using fax machines to communicate.
- In addition to the Protestant Reformation making use of the printing press, Luther's critiques of the church were of similar magnitude.

By providing tools and a mouthpiece to dissidents, we enable them to speak out about the events that are happening around them. Due to the features of Free Haven, this speech is very difficult to trace. Providing a safe avenue for publishing this speech is an important step to leveling the playing field between individuals and governments or corporations.

It is important to recognize that this publishing system must truly be safe. If we provide an unsafe mechanism for publishing text from political dissidents in dangerous countries, the immediate consequences could be disastrous. Further, a few such 'mistakes' may well dissuade people from using similar technologies in the future even if they become more safe.

4.4.11 Whistleblowing – now with documentation!

One of the canonical examples of good applications for anonymous speech is the ability to provide workers or other individuals a channel for anonymous whistleblowing. That is, if something untoward or illegal is happening in a factory or other workplace, one of the workers can report the incident in a manner that doesn't risk disclosing his identity – often such a disclosure could cost the worker his job.

In the case of Free Haven, we can do more than provide a simple channel for reporting these incidents – we provide an actual publication area in which the individual reporting could include extension documentation about the incident, such as pictures or videos of the sanitation levels in the chicken factory, pollution concentrations, outflow rates, epidemiological statistics, financial records, government funding allocations, etc.

4.4.12 Persistent software distribution

A distributed publication system like Free Haven doesn't have to be used solely for political activism or enabling individuals to speak out about controversial matters. Indeed, such a publication system may well be an excellent resource for software distribution in a few years. Current software distribution schemes involve maintaining a single server at a fixed location somewhere in the world; slightly more robust schemes include separately maintained mirror sites to increase redundancy and thus robustness. On the other hand, a decentralized publishing system like Free Haven would provide much smoother redundancy, since the protocol itself divides the document around the world in a robust fashion.

There are a number of very difficult problems to solve, most notably the persistent naming issue (providing global and permanent names [27] for each document or object in the world, such that revision of these documents is seamlessly integrated into a name update), before a global publishing system can take the place of the web for software distribution. However, some sort of more decentralized global network which emphasizes separation between data and physical location seems likely to be its successor. Providing some hints for future designers and developers of what works and what does not work could be invaluable.

4.5 Conclusions

Free Haven provides a service that is not currently available from any other project or application. Web pages available from the internet have their source easily evident. Usenet articles do not reliably reach all readers, and are subject to unpredictable expiration from disapproving administrators or simply due to space constraints.

Because the Free Haven design requires servers to provide space in proportion to the size of the documents they store, we expect the amount of 'unwanted' (porn, etc) material to similarly be proportional to the number of servers attempting to store such material. This means that if 10% of the servers in this system are provided by pornography companies, then roughly 10% of the material in Free Haven will be their material. Therefore we believe that there is little danger of the system getting entirely swamped by such data. This is very different from systems like Freenet, where popularity of a document causes it to be replicated all around the network, expiring other less-popular documents in the process.

By providing a stable and distributed service for anonymous publishing and anonymous reading, we provide dissidents with more powerful tools for both communication and publication. We believe in – and provide – a stronger notion of free speech than simply the ability to make government-sanctioned statements. Overall, we believe that providing individuals with the power to speak in a free, persistent, and untraceable manner is well worth the risk that the system could also be used for less wholesome activities.

Chapter 5

System Design and Overview

5.1 System Summary

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 chapter 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. 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 users (inside or outside the servnet) 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 data of its own in the servnet. The servnet is dynamic: data moves from one server to another every so often, based partly on chance and partly 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 given 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 remembers information for some of the machines in the servnet (the public key, a remailer block or address, and some characterization of trust for each machine).

Only machines in the servnet are allowed to insert files into the network. The amount of storage space a given machine can use is limited by the amount of space it is willing to host – this limitation is loosely enforced by other servers losing trust in a server that 'drops' data.

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. The trust system is used to keep track of which other servers are likely to keep this promise.

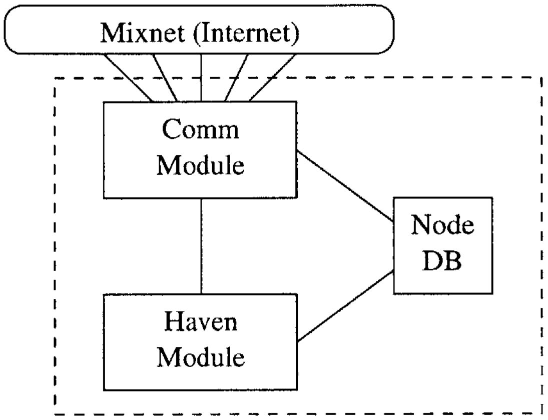

Figure 5-1: Structure of a Free Haven server

The big picture for the structure of a Free Haven server is shown in Figure 5.1. The control center is located in the 'Haven Module': it includes a number of vital operations such as the trading module and the trust module. The 'Comm Module' is responsible for communications with other Free Haven servers; this communication is currently performed via a mixnet, but the system is modular enough that the communications channel could be replaced simply by replacing part of the Comm Module (without affecting the rest of the server). Both the Haven Module and the Comm Module speak to the Node Database, which is a master list of all known servers and all known shares. This Node Database integrates trust information with incoming information about new shares and new servers, to provide answers to a wide range of questions. For instance, the Comm Module might ask the Node Database for the mixnet address associated with a given public key (pseudonym); or the trust module inside the Haven Module might ask the Node Database for a list of candidate servers who might be interested in a share with a certain size and expiration date.

These three modules (the Comm Module, the Haven Module, and the Node Database) are sufficiently distinct that they are designed to run on separate computers within an intranet. This means that a single Comm Module can service multiple Haven Modules, or a single Node Database can be used for several different (fully trusted) servers.

The Haven Module controls a number of aspects of operation of the system as a whole:

- **Storing documents:** When an author wishes to publish a document, she breaks the document into shares, where a subset (any k of n) is sufficient to reconstruct the document, and then for each share, negotiates for some server to publish that share on the servnet. The share module is responsible for handling the arrival of new documents, and maintaining a list of current documents on that server's Node Database.

- **Supplying documents:** When a reader wishes to retrieve a document from the servnet, she requests it from any server, including a location and key which can be used to deliver the document in a private manner. This server broadcasts the request to all other servers, and those which are holding shares for that document encrypt them and deliver them to the reader's location. The share module within the Haven Module is responsible for receiving document requests, identifying which shares are located on the server, and supplying these shares to the reader.
- **Expiring documents:** The share module within the Haven Module is responsible for recognizing when shares have expired, as well as maintaining sufficient space on the system for incoming shares and scratch space for trade requests.
- **Handling trade requests:** The servers trade shares around behind the scenes. The trade module within the Haven Module is responsible for recognizing trade requests, asking the trust module to evaluate the reliability of the server and fairness of the offer, and choosing a new share to offer in return.
- **Initiating trade requests by share:** The trade module is also responsible for periodically identifying shares which have been on the server for a sufficient duration. The trade module should query the trust module to determine a suitable server to which to offer the share.
- **Initiating trade requests by server:** The trade module should also periodically query the trust module to determine if there are any servers which should be 'tested' to increase confidence in the trust module's measure of trust in that server.
- **Handling server introductions:** One of the most important parts of the design of Free Haven is the capacity to seamlessly integrate new servers. The trust system is responsible for recognizing when requests and broadcasts are arriving from unknown servers, and then initiating the correct messages to the Comm Module and other servers to arrange to receive contact information for those new servers.
- **Expiring old servers:** Unavailable servers need to be flagged in the database such that the Comm Module does not include them in broadcasts, since continued mail bombardment from the mixnet to a closed account will eventually slow down the service (as well as anger a lot of systems administrators). The trade module is responsible for recognizing when requests are ignored and informing the trust module (which will pass it on to the Node Database). The

Comm Module takes care of reactivating a server which has been flagged as expired; it notices this when a message arrives from that server.

- **Initiating trust broadcasts:** Periodically, the trust module will select a server from the Node Database and broadcast its current opinion of that server's reliability. This broadcast allows other servers to keep informed of actions which do not affect them directly.

- **Handling trust broadcasts:** When a trust broadcast arrives regarding a given server, the trust module must integrate this new information into its evaluation of that server. This is based in part on the current trust of the broadcasting server, the current trust of the subject of the broadcast, and the actions described in the broadcast, if any.

- **Initiating buddy checks:** To maintain some level of accountability, shares employ what is essentially the 'buddy system': servers which drop shares or are otherwise unreliable get noticed after a while, and are trusted less. The buddy module within the Haven Module is responsible for keeping track of the location of the buddy of each share the server currently possesses. Periodically, it should perform a document request for this share, to verify continued availability of this share.

- **Handling buddy checks:** Buddy checks are identical to normal document requests, as above. They must be identical to prevent servers from replying only to buddy checks and thus trick other servers into believing that they are still supplying the document to readers as well.

- **Initiating buddy broadcasts:** When the buddy module concludes that the buddy for one of that server's current shares is lost, it can choose to perform a 'buddy broadcast', which is essentially a notification to the trust modules of other servers that it believes the share to have been lost by a given server.

- **Handling buddy broadcasts:** The trust module is responsible for receiving and interpreting buddy broadcasts. These should modify the trust of the subject of the broadcast, based again on current trust of the broadcasting server, the current trust of the subject of the broadcast, and other relevant factors.

5.2 Supported Operations

There are two operations which any publishing service must provide: adding a document to the system, and retrieving a document from the system. Free Haven provides these operations, and also supports a number of behind the scenes operations such as trading, expiration, and rudimentary support for enforcing accountability without sacrificing anonymity.

5.2.1 Storing

When an author (call her Alice) wishes to publish a new file into Free Haven, she must first identify a Free Haven node which is willing to store this document for her. The exact method of identifying and contacting this node is outside the scope of the Free Haven design per se; some possibilities include:

- Alice is running a node herself. In this case, she would presumably be willing to store her own file.
- Some of the nodes are publically contactable, via a website or some other public interface. These nodes are willing to introduce Alice to other private nodes, are willing to store Alice's file, or are willing to sell Alice some space on their systems.
- Some of these nodes have reply blocks (and their associated key) published on some site associated with Free Haven, and might be willing to publish files that Alice delivers through the mixnet.

Alice may choose to deliver her file to the given node (if she herself isn't the node) via an anonymous remailer or other one-way anonymous communications channel, to provide strong author-anonymity as described in Chapter 2. Alice may also choose to encrypt the file before transmitting it – since any reader can retrieve any file if he knows its lookup key (as below), we leave this level of privacy up to Alice's discretion. Alice may also choose to divide the file into new files if it's too long, or pad it if it's too short, based on what sort of file sizes the server she finds is willing to accept. (Prudent choices from the server's perspective are discussed in more depth below under Trading.)

When the publishing server (call him Phil) wants to introduce the new file into the servnet, he breaks the file into shares using Rabin's information dispersal algorithm [84], and then for each share, he finds a machine on the servnet which he trusts and which is willing to make the trade for that share. More precisely, he performs the following steps.

1. Break the file F into shares $f_1, \ldots, f_n$ where any k shares are sufficient to recreate the file.
2. Generate a public/private key pair (PK, SK) to be used for signing each of the shares.
3. For each share, build a data segment as described below under 'Composition of a Share' and sign this segment with SK.
4. Enter the shares into the local server's space.

Really, Alice may perform steps 1 and 2 herself, and thus build the shares herself. A well-behaved server should be able to handle doing the IDA process itself as well as receive shares that have already been split. Note that accepting pre-built shares seems to require more trust of Alice than receiving her file, examining it, and then building a set of shares from it.

The choice of the robustness parameter k is a crucial part of adding a document to the system. 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. This parameter k should probably be chosen based on some compromise between the importance of the file and the size and available space.

A more considered value for k should be available based on the results of modelling the mixnet and servnet to determine how many documents are lost in what sort of situations. The brittleness of documents in the servnet is affected by a variety of factors, including percentage of misbehaving nodes, reliability of the network itself, trading frequency, and the rate at which new servers join the servnet and old servers disappear.

5.2.2 Retrieving

In order to retrieve a file, a client must first know the PK which was used to sign the shares. He learns this from a post to Usenet or some similar external means (or because he was the original author of the document) – see Section 5.9 on how to integrate Directory Services with Free Haven. From here, he must locate a servnet server that is willing to do the query for him. The server can be contacted over its remailer address, but presumably there will be public cgi's available which automate the process of doing the document query. The reader comes up with a key pair (PK, SK) for this transaction, as well as a one-time remailer reply block. The servnet server does a broadcast of "('request', PK_{file}, 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 (since the mixnets are going to introduce some delay as it is, adding a bit more by sending requests perhaps once an hour will not significantly affect the latency).

Each server that receives the query will check to see if it has any shares with the requested PK_{file}, 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, the client recreates the file and is done. If not enough shares arrive, then the client has failed to obtain the file (most likely, it is lost).

The broadcast request can optionally specify which share index is desired. This allows clients who only want one share to be able to query them specifically. For instance, a node wanting to confirm that a given share is still 'alive' might query only for that share. This extension must be implemented with care, though – if a particular operation (such as buddy checking, below) is implemented by always querying for a specific share of a document, then nodes may be able to differentiate between different classes of queries (e.g., querying to retrieve a document and querying to determine if a document is still available) and selectively respond. This would allow them to convince an auditor that the server is obeying its protocol, while at the same time the server does not answer any document requests from actual readers attempting to retrieve those documents. This means that in simple implementations, querying by specific share should be disallowed; in more complex implementations, all operations should support (and have an equal probability distribution for) querying either by entire document or by specific share.

Additionally, the broadcast document request can optionally be signed by the servnet node performing the broadcast. This has a number of ramifications. First of all, it introduces more complexity in the problem described above, namely in trying to ensure that queries cannot be selectively answered by a server attempting to gain trust without serving documents to readers. On the other hand, it allows slightly more accountability on the part of servers: this might be useful in the case of denial of service flooding attacks, wherein an attacker might flood a given node with document requests with the intent either of spamming some helpless victim, or of saturating the bandwidth resources of the server. In this circumstance a server might gain some defense by dropping incoming requests that are not signed by a trusted peer server.

5.3 Expiration

Each share includes an expiration date. 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. This means that if an operator wants to cease providing his node as a Free Haven server, the protocol provides him with a polite way of exiting the system: wait until all the shares he has expire. One way to hasten this would be to trade away for a very large share that expires soon – see below under Trading for more details on this.

Expiration dates should be chosen based on how long the introducing server wants the data to last, compromising based on size of file and likelihood of finding a server willing to make the trade. Servers should be wary of accepting shares that they will not be able to trade away easily, because then they might be committed to keeping that file on their system until it expires.

This last point has strong implications for the stability of shares in the system – shares which are nonstandard either in size or in duration may well be more fragile than shares which are closer to average. One reason for this is that shares with particularly long durations are simply subjected to more chances to be destroyed over the course of their lifetime. A more subtle reason for this is that a very large share may be difficult to successfully trade away; indeed, it may turn out that servers which accept extraordinarily large shares have a greater tendency to be unreliable. (Perhaps, for instance, the server accepts such shares due to negligence on the part of the server operator, which might be indicative of other stability problems.)

5.3.1 Revocation

The ability to revoke or unpublish shares would provide much greater flexibility to the Free Haven system. Specifically, this would allow a much more realistic emulation of an actual read-write filesystem, where published documents could be updated as newer versions became available. Indeed, it also allows political dissidents who publish under their real name to realize their mistake and unpublish the documents.

Revocation could be implemented by allowing the author to come up with a random private value x, and then publishing $H(x)$ inside each share. If the author wanted to unpublish a document, he would broadcast an unpublish request along with his original value x (and also $H(x)$ for the sake of convenience and efficiency), and all servers which were currently holding shares of the document would expire them.

However, there are a number of extra attacks this allows on the system:

- It complicates the buddy system greatly, since we are not sure that the unpublish request would reach the buddy of a given share. Indeed, an adversary might send unpublishing requests to some members of the servnet and not others, in an attempt to cause havoc in the trust system, or even to try to gain insight into the current location of some shares.

- Authors might use the same hash for new shares, and thus 'link' documents. Adversaries might also use the same $H(x)$ even though they are unaware of the value of x: this would cause artificial linking, as observers might conclude that the publisher of the original document also published the later documents.

- The presence of an unpublishing tag $H(x)$ in a share assigns a sort of 'ownership' to a share that is not present otherwise. This may have subtle implications towards publisher and reader anonymity – for instance, a publisher who remembers his x has evidence on his computer that he was associated with that share, thus breaking perfect forward author-anonymity.

In addition, if revocation exists, then a corrupt police force or intelligence agency has an incentive to track down the original author of the document, because chances are good that he still has the value x which would allow them to remove the document from Free Haven. Even if the author immediately destroys his x, the adversary has sufficient reason to suspect that he still has it that it is worthwhile for them to spend resources tracking him.

This problem can be ameliorated by making the unpublishing tag optional. This means that the share itself will make it clear whether that share can be unpublished, so if no unpublishing tag is present, there should be no reason to try to track down the author. If an adversary wishes to create a pretext to hunt down the publisher of a document, however, he can still republish the document *with* a revocation tag, and use that as 'reasonable cause'.

An alternate solution, given that we are willing to accept some amount of linkability in exchange for the ability to perform document revocation, is to simply allow the publisher to remember the key K with which he originally signed each share of the document. Thus instead of generating a separate value x simply for the sake of revocation, we could allow the publisher to revoke his document by transmitting σ_K('revoke'). We can further allow the 'non-revocable' notion on a share by allowing an explicit tag called `<revocable>` which if present would indicate that 'revoke' messages should be honored. Publishers could similarly yield their original key K to some trusted agency. However, this alternate mechanism for revocation is still susceptible to many of the above attacks.

Because the ability to revoke shares potentially puts the original publisher in increased physical danger, as well as increasing the set of attacks on the servnet infrastructure, we chose to leave revocation out of the current design.

5.4 Accountability and Redundancy

Without some sort of server accountability, shares could get swallowed by malicious nodes and nobody would ever notice. This lack of accountability has two ramifications. First of all, over time shares will disappear, eventually causing files to be unrecoverable. Secondly, malicious nodes can continue to be malicious if there are no mechanisms in place for identifying and excising ill-behaved nodes.

One solution to this would be for the publisher of each share to maintain his own local copy, and periodically query the servnet for the share. If the share has disappeared, he will simply drop some of the data currently on his system and insert his backup copy of the share. (At this point, he might stop trusting the server to which he traded that share.) However, this makes the entire system very fragile, as well as making it slow to react to malicious nodes.

A better solution would be to somehow keep track of the current location of each share. Thus a given server would know exactly when a share disappeared and which server was responsible for it at the time. Broadcasting knowledge of bad nodes would quickly update the servnet on the presence and behavior of suspected malicious nodes, limiting the amount of damage that can be done. This solution could be implemented by assigning a fixed and permanent 'shepherd' to each share upon publication (it would be permanent because the share is signed with the shepherd inside); every time a share is traded around the servnet, then both servers involved in that trade would send updates to the shepherd indicating that one of them had relinquished responsibility for the share and the other had taken responsibility for it. These updates would allow the shepherd to maintain a good idea of the current location of its associated share.

However, having a static location for keeping track of location information for each share defeats the purpose of having them scattered throughout the servnet. Specifically, this provides a centralized target for anyone wishing to learn more about (or attack) a given share. (One might argue that as Free Haven scales, some form of directory capability is going to be necessary, and thus maintaining knowledge of location of shares will be necessary anyway. However, this is not the case: a directory service can be implemented without any notion of the location of shares. See Section 5.9 below for more details on Directory Services.)

One possible solution to this is to associate pairs of shares in a given document with each other, and use the 'buddy system'. In this case, each share would be responsible for maintaining information about the location of the other share. When a share moves, it should send notification to the other indicating this move. More correctly, the server sending the share should send notification, and then after the transfer the server receiving the share should send notification. Each notification includes a 'receipt' as described below under Receipts.

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 5.8.

We considered allowing abandoned shares to optionally spawn a new share if their buddy disappears. This would make the service much more robust, since shares that are destroyed can be regenerated. But it is also possible that it could cause an exponential population explosion of shares: if a share is out of touch for a little while but isn't dead, then both shares will end up spawning new copies of themselves. This is a strong argument for not letting shares replicate.

Another more subtle problem with having both shares with non-fixed locations is that the communications channel we have chosen currently has nontrivial latency. This means that a share might have already been traded away by the time notification arrives from the other share. We could solve this by having some sort of locking mechanism on a share, such that they are forced to alternate trading. However, this solution is prone to attacks which 'stagnate' a share or otherwise attack its location.

Instead, we solve this problem by maintaining 'forwarder' addresses after a trade has taken place. Since these forwarder addresses take up very little space, and since servers must retain receipts for the given share anyway, these forwarder addresses are kept until the share's expiration date. (Actually, this forwarder address can be parsed out of the receipt, so servers really don't need to keep the forwarder as a separate value.)

Note that 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 – this means that the node on the other end of the forwarder will also get its own copy of the request.

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 of each other. Robustness can be increased by increasing the number of shares for a document or decreasing the associated k parameter; accountability can be increased by increasing the number of buddies or the frequency of checking.

5.4.1 Buddy Numbering

The choice of how to number buddies during share creation is a complex one. Our original approach was to create two near-identical buddies for each share of the document; these buddies would be the same except one would be labelled as buddy 0 and the other as buddy 1. However, we soon realized that this contradicts the above goal of using buddies solely for accountability and using extra shares solely for robustness. The better alternative is to match pairs of shares during document creation: 0 and 1 are matched, 2 and 3 are matched, etc.

Since shares have information about the location of their buddies, there are a number of attacks that a server can launch to collect both buddies. Once both buddies are located on the same malicious server, that server can delete them without fear of any repercussions. Thus once a given share of a document happens to arrive at a malicious node, that node can do trade requests to try to obtain the buddy. A more subtle approach would be for a conspiring node or set of nodes to do the trade requests instead, lest the other server become suspicious.

An alternative approach to buddy numbering would be to build a chain: share i has $i + 1$ as its buddy, $i + 1$ has $i + 2$ as its buddy, and so on. This might make the system more robust, since the buddies are connected to each other as more than just pairs. On closer inspection, however, this approach makes the document much more vulnerable to deletion: once an adversary obtains even a single share, he can iteratively obtain control over each new share in the 'chain', eventually controlling a sufficient percentage of the document that he can effectively destroy it. Further, after he owns some share i and also $i + 1$, he can delete share i because he controls $i + 1$ – this is the only share which is watching i.

An attempt to salvage this idea might be to build a doubly-linked chain, wherein a given share $i + 1$ watches both i and $i + 2$. However, this approach greatly increases the complexity of share monitoring, and does not give any clear advantage over the simpler schemes – in fact, it may even make the exploit easier for the adversary since he can attack the chain from both ends.

Building an effective accountability system is a complex and challenging problem. The buddy system is a good start, but it will require a lot more thought and engineering genius before it can reliably enforce good behavior for servers.

5.5 Composition of a Share

Using an information dispersal algorithm, documents inserted into Free Haven are split into n pieces, any k of which are required to reconstruct the document. A share contains one of these n pieces along with information about the specific share and about the document. Specifically, the share contains the public key of the document, the share number, the share's buddy number, an expiration date and time in Greenwich Mean Time (GMT), and a signature.

An example of the share format is:

```
<share>
<PKdoc>cec41f889d75697304e89edbdddf243662d8c784</PKdoc>
<sharenum>1</sharenum>
<buddynum>0</buddynum>
<totalshares>100</totalshares>
<sufficientshares>60</sufficientshares>
<expiration>2000-06-11-22:25:24</expiration>
<data>Ascii-armored characters here</data>
<signature>cec41f889d75697304e89edbdddf243662d8c784</signature>
</share>
```

Shares have the following characteristics:

- Share information is completely enclosed inside the outermost $< share >$ and $< /share >$ tags. Data before the $< share >$ and after the $< /share >$ is ignored.
- The public key of the document is placed inside the $< PKdoc >$ tags.
- Shares are numbered 0 through $n - 1$. The number of each share is placed inside the $< sharenum >$ tags. This particular share represents share 1 of the document.
- Each share has a buddy share, which is one of the other shares of this document. The buddy number is placed inside the $< buddynum >$ tags. In this case, the buddy of this share is share number 0. Because buddies are pairs, then share 0's buddy would symmetrically be share 1.
- Each share also includes the total number of shares that were created for this document when this share was created, along with a value which indicates how many shares are sufficient for reconstruction. These values are there for convenience, and may well be superfluous.

- The actual piece of the document as produced by the information dispersal algorithm and represented by the share is placed between the < *data* > and < */data* > tags.
- The string inside of the < *expiration* > tags represents the GMT time when the share is free to be deleted. This share, for example, will expire at 10:25:24 pm on June 11, 2000. Shares are not necessarily deleted at their expiration time. Note that we zero-pad the datestamp – this provides a fixed-width datestamp which may be convenient for reading or automatic processing.
- The information up to and including the < */data* > tag is signed by the cryptography module, and that value is placed inside the < *signature* > tags. Specifically, this signature is done by taking all the tags, removing whitespace between tags, and signing the resulting string.

During the signature operation, the various tags are ordered alphabetically before the external whitespace is stripped. We do this because we want to maintain the flexibility of using XML as our share format. XML specifies that the tags can be in any order, with whitespace between them. On the other hand, our signature must be a signature of a certain set of deterministically ordered bits. Thus we compromise by building a string which comprises each of the tags ordered alphabetically, and then signing that string.

5.6 Trading

Share trading is an integral part of the structure of the Free Haven network. There are a number of reasons why servers trade:

- **Greater anonymity**: if trades are common, then there is no reason to assume that somebody offering a trade is the publisher of a share. Even if the mixnet successfully protects the identity of a servnet operator, nodes could still start rejecting trades from other nodes based on content of shares they previously provided in trades.
- **More dynamic**: frequent trading makes adding and dropping nodes transparent. If shares were just traded once, servnet nodes would have to support extra protocols for dropping out of the network politely (negotiating a new keeper for the share, informing the publisher of the move, etc). 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 some of the longer-lived shares.
- **Can handle longer expiration dates**: long-lasting shares would be difficult to trade away and rely on, if trading them involved finding a server that promised to be up and available for the next several years.

- **Accomodates ethical concerns from servnet operators**: if there's a particular piece of data with which an operator does not wish to be associated and he notices that he is storing a share of that data, frequent trading makes it easy and unsuspicious to trade it away. Operators that do not have this flexibility would end up just dropping data they don't like, or not participating in the servnet.
- **Provides a moving target**: we rely on the security of the mixnet to protect the identity of servnet operators. However, if trading doesn't happen, an organization will learn that a given document lives at the other end of a certain mixnet address, and that it has five years to break it. Encouraging shares to move from node to node through the mixnet means that there is never any specific target to attack.

Trades are considered 'fair' based on the two-dimensional currency of 'time*duration', plus a function of the preferences of the servers involved in the trade. For instance, one server might consider a 1 megabyte share that expires in 2 months to be roughly equivalent to a 2 megabyte share that expires in 1 month. However, another server might be biased towards short-lived files, since it prefers to have the opportunity to leave the servnet at short notice. However, the economics don't have to work this way: it could well be that a given server requires a trade in its advantage before it's willing to complete a transaction, because it doesn't have much trust in the server offering the trade. Alternatively, it could be that a server would be willing to make trades where it ends up with more megabyte-days; such servers might build up a reputation of being easy to trade with.

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 along with some set of hints describing the share that it is interested in receiving in return. If B is interested, it responds with a share λ_k of its own. The negotiation is finalized by each server sending an acknowledgement of the trade (including receipt, as described below) to the other. In the above diagram, there are also two other parties involved: C, which is holding Φ_j, the buddy of Φ_i; and D, which is holding λ_l, the buddy of λ_k. The 'buddy system', plus the accountability which it provides, is described in more detail above, in the Accountability and Robustness section.

If the offered share Φ_i is not acceptable, B can simply not respond, or it can respond with some description of why it refused the trade. This gives some indication of which trades might be accepted in the future. If B chooses not to respond, however, he runs the risk of having A conclude that he is a dead node; see Section 5.8.4 below on Node Expiration.

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

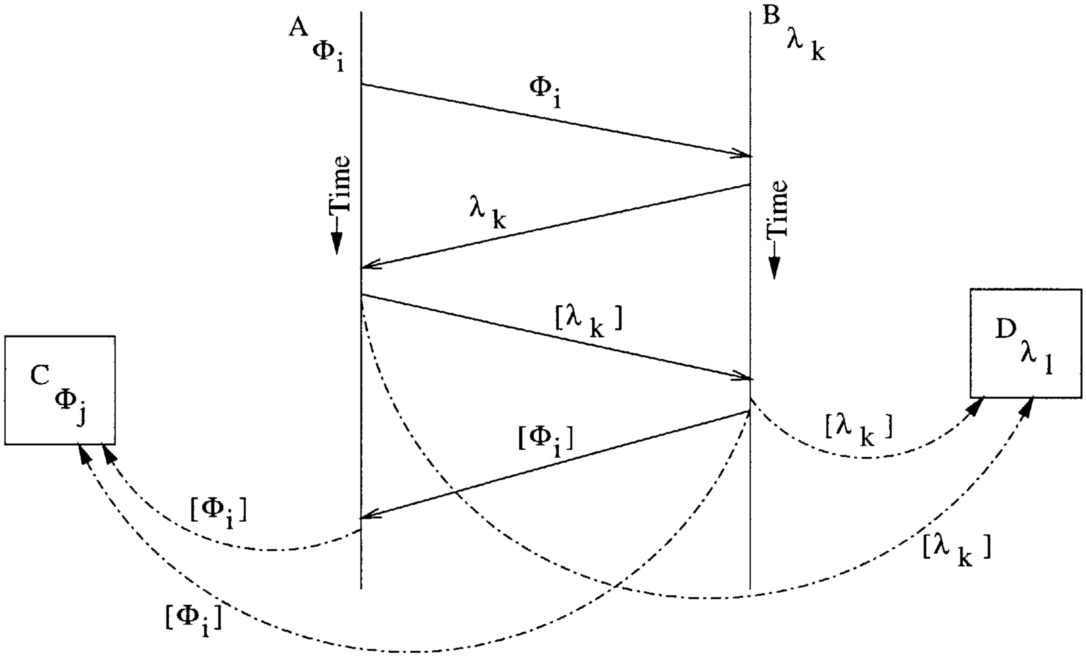

Figure 5-2: Trade handshake timing diagram

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 does its job to cause 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, and 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. This flag will help with accountability.

One of the design ideas that we considered and rejected was to produce variable-sized shares, and distribute these shares proportionally to other servers based on trust, reliability, or response time [68]. This way we would provide more shares to servers who are fast and likely to keep them safely, so document requests would be faster and more reliable. On the other hand, due to the expected frequency of trading, shares will be mixed relatively thoroughly around the servnet – which server the share was first traded to should not make any difference. Indeed, this process also seems to add a lot of complexity to the system: micromanaging the behavior of each servnet node is not conducive to a powerful, flexible, and decentralized network.

5.7 Receipts

Note that 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 server misbehaving is to broadcast a complaint about it, and hope the Trust system handles it correctly.

When server A trades share Φ_i to server B in exchange for share λ_k, then server A will generate a receipt for λ_k with the following entries:

- "I am": A
- "I traded to": B
- "I gave away": $H(\Phi.key), i, \Phi_i.exp$, size
- "I received": $H(\lambda.key), k, \lambda_k.exp$, size
- "Timestamp": τ

This entire set of five elements is signed by server A. This signature means that B is not able to forge this receipt. This means that A really must have created the receipt, so 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.

5.8 Trust Networks

[1] This system is built off some amount of trust in the other nodes of the servnet. In particular, the protocol supports some enforcement of good practice by the other nodes, but there is still plenty of opportunity for nodes to obey the rules until they are sufficiently trusted, and then start breaking the rules in subtle ways. For instance, a node might sometimes fail to provide a receipt to a share's buddy during a trade, introducing confusion as to whether the other side is trying to trick the buddy into believing that the share had been traded away. More destructive would be for a node to never

[1] This section was written in part by Brian Sniffen.

query the buddy shares for any of the shares it possesses, or even wrongly accuse a different node of losing that share. The list goes on and on. However, with careful trust management, each node ought to be able to keep track of which nodes it trusts; with the cushioning provided by Rabin's information dispersal algorithm, only a significant fraction of the nodes turning evil at once will result in actual loss of files.

5.8.1 Protocols for informing neighbors

Each node needs to keep two values describing each other node it knows about: trust and metatrust. The first, trust, signifies a belief that the node in question will obey the dictates of the Free Haven Protocol. The second, metatrust, signifies 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 rating. This serves to represent the "stiffness" of the trust value. For example, a node which has Trust 100 but Confidence 1 will be trusted with a great deal of data, but will lose or gain trust rapidly in response to the utterances of other nodes. Exactly how these values are used is left entirely up to each node.

Some nodes may wish to set all meta-trusts to zero, thus ignoring all outside utterances. Other nodes may wish to set confidence values very high, ignoring not only outside utterances but also direct evidence of wrongdoing.

Nodes should broadcast referrals in several circumstances:

- When they log the honest completion of a trade.
- When they fail to verify that a buddy of a share they hold is safely held.
- When the trust or metatrust in a node otherwise changes substantially.

5.8.2 Getting information

In addition to referrals, the trust system should gain information from the operation of its own node. Such information must be treated as having a very high metatrust — as we're talking to ourselves, why would we lie?

The trust system should log all transactions between its own node and others. It needs this information to determine both when shares have successfully expired and when shares have been deleted before their expiration date.

5.8.3 Introducing new nodes

One of the most important parts of the design of Free Haven is the capacity to seamlessly integrate new servers. These servers need to be able to join and participate in the servnet simply by installing some simple packages and making contact with public servnet nodes. Such nodes (e.g. `freehaven.org` itself) have the option to configure themselves as *introducers*. While nodes configured with the default settings should ignore communications from unknown nodes, introducers respond to such communications by querying the new node for its public key and return address. (Presumably this is some sort of anonymous communications channel, such as a Mixnet.)

Provided with this data, the introducer adds the new node to its database, then broadcasts a referral of that node. Other nodes are welcome to do what they like with this information. It is suggested that the initial trust and metatrust values both be zero, with some small amount of confidence in each. The introducer may also wish to send to the new node a referral for each node it knows about, thus assisting in keeping the network as fully connected as possible.

From this point, it is expected that existing nodes will attempt to offer trades to the new node, and vice versa.

5.8.4 Purging old nodes

Over the course of the operation of Free Haven, many nodes will arrive and many nodes will depart. However, due to the nature of the communications channel we have chosen, there is no way of knowing if a message does not arrive at its destination through the mixnet. Because of this lack of bounces, there is no direct way to determine if a remailer address is no longer in service.

To maintain a set of active servnet nodes, we simply keep track of the number of recent trade requests to a given node which were not answered. After a certain threshold of unanswered trade requests, we mark that node in the node database as 'dormant'. This means that the node is not currently responding to trade requests, and so we should neglect to include that node in our broadcast messages as well. This modification is implemented in the Communications module by simply skipping over dormant nodes during a broadcast. In addition, the trust module should skip over dormant nodes when choosing appropriate nodes for trades.

If a trade request or broadcast arrives from a dormant node, then we can conclude that the node is not actually dormant and resume sending broadcasts and offering trades to this node.

There may be a number of attacks based on the fact that servers might stop sending broadcasts to a silent node. However, it seems that the only way to prevent a node from responding is to either control that node or control an appropriate edge in the mixnet. For an adversary with this level of power over the system, a more thorough denial of service attack would instead be possible.

5.8.5 When to trade

While individual node administrators should be free to change this as they wish, the default Free Haven installation should base its unit of trust currency on the product of share size and storage duration: megabyte-months. Each host should also grant a small amount of leeway. This allows new nodes, for example, to be able to trade small shares despite their 0-trust rating.

A reasonable policy will balance trust-increasing trades with low-confidence nodes against guaranteed-safe trades with high-trust, high-confidence nodes.

5.8.6 Implementation

The Free Haven Trust Module is a library of code accessed by the Haven Module. Because it acts as the information repository for the Haven module, it makes sense to offload its logging and querying needs to a relational database backend. Such a backend can be easily shared with the Communications Module, ensuring coordination of keys and reply blocks. It also makes sense to store metadata about shares in the database, while leaving the shares themselves in a traditional file-system. Appropriate tables for the trust portion can be described in SQL as shown in Figure 5-3.

Of particular note, the `trades.receipt` data is duplicated in the other fields. The receipt itself needs to be there for rebroadcast in case of betrayal, and for its signature. The other fields have been extracted from it for quick and easy access.

Based on user configuration and the above-mentioned database, the trust module supports the following API:

`inform_trustdb(struct tag_t *tag_list)` Takes in a parsed XML referral and adds it to the `referrals` database. See the "Interpreting Referrals" section below for more information.

`trust_find_trade_host(char *target_host, char *desc)` This call gives the trust db great freedom of action: the result should be to find a host we wish to trade with, then write its key into `target_host` and a description of a share it might want into `desc`. This last is obtained from the sharedb once we have selected a host. In the initial implementation, the choice of host is random, but weighted towards those with low confidence.

`trust_find_trade_host_by_share(char *target_host, const char *share)` Given a share description `share`, we find all the hosts which we'd trust to hold it, then select randomly among them, weighted towards those with high confidence.

`trust_find_desc_for_host(char *desc, char *target_host, char *share)` This call writes a share description into `desc` which we are willing to receive from `target_host`. In the initial implementation, this simply describes the limits of our trust of them.

`trust_accept_share_phase_one(struct tag_t *tag_list)` In the initial implementation, this call checks on the trust of the offering node, to see if we're likely to be comfortable trading an equivalent share to them.

```
create table nodes (
    id              integer unique,
    key             varchar(4000),
    replyblock      varchar(8000),
    trust           integer,
    confidence      integer,
    metatrust       integer,
    metaconfidence  integer
)

create table trades (
    source_node     integer references nodes,
    dest_node       integer references nodes,
    receipt         varchar(4000),
    given_keyhash   varchar(512) primary key,
    given_size_k    integer,
    given_exp       date,
    taken_keyhash   varchar(512) primary key,
    taken_size_k    integer,
    taken_exp       date,
    timestamp       date,
    expired         integer
)

create table referrals (
    target_id       integer references nodes,
    referrer_id     integer references nodes,
    trust           integer,
    confidence      integer,
    metatrust       integer,
    metaconfidence  integer
)
```

Figure 5-3: SQL Table Descriptions

`initialize_trust_module()`, `close_trustdb()` These are simple initializers which deal with opening a connection to the database and reading in the configuration file.

The two functions `trust_find_trade_host` and `trust_find_trade_host_by_share` are biased in different directions with respect to confidence. The assumption is that the haven module will call `trust_find_trade_host` when it wishes to do trust-building trades and serve the priorities of the trust system, and call `trust_find_trade_host_by_share` when it wishes to do safe trades. In accordance with this assumption, we trade with low-confidence hosts when offered the opportunity to do so, and high-confidence hosts when told it's important.

Interpreting Referrals

When the Trust Module receives a referral of the form [Trust: T, Confidence: C, Metatrust: M, Metaconf: F] targeted at a node with characteristics [Trust: t, Confidence: c, Metatrust: m, Metaconf: f], it first checks to make sure it has no other referrals with the same target and referrer — if it does, those are backed out of the system and replaced with the new referral.

Then we add $((T - t) \times M \div c)$ to t, and move c one notch in the same direction — that is, if we have increased a positive t or decreased a negative t, we increment c, otherwise we decrement c. Then we add $((M - m) \times M \div f)$ to m, and move f one notch in the same direction, as above.

If we do currently agree with the referral $(T = t, M = m)$, we do not change our trust in the target node at all, but instead increment our metatrust and metaconfidence in the referrer: if that node agrees with us, it must be worth listening to.

Referrals with receipts In the case of a referral with a receipt, we proceed somewhat differently. We ignore metatrust issues, and instead proceed as if we had noticed the lapse ourselves: we decrease our trust in the node which defaulted by the product of the size and the intended duration of the trade (that is, the difference between the timestamp on the trade and the expiration date of the share) and increment our confidence in that trust.

We take no action with regard to the metatrust of the referrer; it was acting based on obvious information, so we have no reason to believe it is particularly wise in other matters..

Gaining Trust Independently

The trust module periodically scans the database, rebuilding trust information. It is in this way that expired shares are noticed. When a share expires without any sign of having disappeared early, we increase our trust in the node we traded it to by the product of the size and the duration of the trade (that is, the difference between the timestamp on the trade and the expiration date of the share), increment our confidence in that trust, and mark that trade receipt as expired.

Losing Trust Independently

When the Haven Module fails to verify that a buddy share still exists, it informs the Trust Module. The respone is the reverse of a successful trade, above: we decrease our trust in the node which defaulted by the product of the size and the intended duration of the trade (that is, the difference between the timestamp on the trade and the expiration date of the share), increment our confidence in that trust, and "squawk."

This "squawk" takes the form of a broadcast referral about this new trust data, including a receipt for the trade which landed that share at the apparantly corrupt host.

Disagreement When scanning the database as mentioned above, the trust module notices when it has referrals for a node which are of the opposite sign from its current trust. Such referrers have their metatrust decreased, to indicate that they are either unwise or untrustworthy.

5.9 Directory Services

One issue we have not yet addressed is the question of directory services. In particular, how do people get a directory of what's listed in the Free Haven? How is this directory maintained? Doesn't it need to be secure, distributed, and anonymous, just like the other documents, so we should put it inside Free Haven?

A document directory has no business being inside Free Haven, for several reasons:

- The nature of the directory is that it's always changing, whereas the data in Free Haven is designed to be immutable.
- Free Haven emphasizes storage, not accessibility. A directory is generally intended to be retrieved frequently. Indeed, if the communications medium we choose has high latency, retrieving a directory will be a very slow process anyway.

We expect that a number of independent directory services will pop up, in many different jurisdictions. These directories can be updated (via remailers) by servers as they put new documents into the Haven, either truly anonymously or signed by a key that a server uses solely for announcing new documents. (This might even develop a sort of economy of trust for how easily a directory service believes a new submission.)

If a given directory service is shut down, then the others will persist.

Servnet operators can build up their own directory services based on what shares pass through their system. If servnet operators help to synchronize and verify shares listed on an external directory service (informing the service of new shares or incorrect current shares), this may make the service more stable and useful.

Indeed, even if there are no directory services at all the system will continue to perform its basic function: as a reliable mechanism for anonymously storing data. Even if only the original publisher of a document knows how to retrieve it, then the Free Haven system is still performing a useful service.

5.10 User Interfaces

5.10.1 Design

[2] Interacting with the core Free Haven process, the user interface has three substantial goals: to provide users with the ability to anonymously insert documents into the Free Haven system, to provide users with the ability to anonymously retrieve documents from the Free Haven system, and to allow internet-literate users to easily perform these two operations.

Upon first inspection, the ability of the user interface to insert and retrieve documents may appear to be a trivial extension of the core trading protocols of Free Haven. The challenges facing the insertion and retrieval of files from the user interface perspective are somewhat different from those facing the designers of the core infrastructure, however. The trading of shares in the servnet requires a two-way transfer and assurance of success, but the user interface – servnet interaction is only concerned with the transfer of documents in one direction. The user interface must by definition, however, provide sufficient anonymity to users of Free Haven.

In the case of retrieving a document from the servnet, a user needs only the ability to make an adequate request of one of the servnet nodes to find shares of the document: a request is considered adequate if the user provides a hash of a public key of a document that matches a hash stored in the system In this request, the user also provides the servnet with a reply address. The servnet nodes with a share of the document, theoretically, simply make a copy of the share and send it off to the reply block included. If the user has the public key of the file, then the servnet assumes that the user's request is valid.

Similarly, in order to insert a document into the network, the user must only find a servnet node willing to accept his document shares. One option is to have servnet nodes who are always willing to accept new documents and publish them into the Free Haven system. This option is currently in use with `http://freehaven.net` being the servnet node willing to accept new documents. Unfortunately, this option does not provide the highest level of anonymity. Another option would be to simply query servnet nodes about their willingness to accept a document. This option would provide slight more anonymity at the expensive of having to iterate through all of the known nodes.

[2] This section was originally drafted by Todd Kamin.

For the first pass user interface, more consideration was given toward creating an interface that was easy to use and relatively simple to build than toward providing maximum anonymity.

5.10.2 Implementation

Currently, the Free Haven user interface must reside on a Free Haven node. The user interface consists of a perl script, called `handler.pl`, which is invoked through CGI. An HTML document contains the appropriate form syntax and information necessary to invoke the user interface on a particular Free Haven node. The remote execution of the user interface perl script is made possible through a web server that allows CGI execution. The Free Haven node must be running a web server configured to allow CGI requests in order for users to access Free Haven.

The two necessary user interface operations are the ability to insert a document into the Free Haven system and the ability to retrieve that document from the system. These two operations require completely different actions to be taken, and thus are implemented somewhat independently of each other.

Inserting a Document

In order to insert a document into the Free Haven system, a user first brings up the appropriate form. An example form can be found at `http://web.mit.edu/www/tkamin/form.html`. Using the bottom of this form, the user selects a document for upload. Note that the current implementation only allows for insertion of text files into Free Haven. The user also has the option of setting the time to expiration of the document. (The default time to expiration is 4 weeks.) By choosing the time to expiration of the document, the user is given a promise that the document will not be deleted for that amount of time. Pressing the submit button causes the time to expiration and the document to be delivered to the CGI program. The current version of this CGI program can be found at `http://freehaven.net/freehaven-cgi/handler.pl`. This CGI program is a perl script running on a Free Haven node; it effectively represents the "user interface."

With the information provided in this "insert" request, the user interface perl script produces shares for insertion into Free Haven. The user interface first proceeds to split the document using Rabin's Information Dispersal Algorithm [84] into a specified number n pieces with k ($k < n$) being required to recreate the document. The values of k and n are specified in the file `cgi.conf` and can be easily changed. The user interface then converts the output from the IDA program into shares according to the "composition of shares" specification. The expiration date is determined by adding the time to expiration and the current GMT time. Once the share is built up to the $< /data >$ tag, the user interface uses the Simplified Wrapper and Interface Generator (SWIG) [32] to invoke a method in the cryptography module that provides a unique signature. The user interface appends the signature and closes the share. Each share is placed in a separate file.

By making the assumption that `handler.pl` will reside on a Free Haven node, the user interface script is able to communicate with the haven module through files placed in a prespecified directory. The share files built by the user interface contain sufficient information to be uploaded directly into Free Haven. The user interface moves the files containing shares to the preconfigured directory. The haven process eventually (more precisely, each time a trade is offered) comes along and uploads these shares into the servnet. Lastly, the user interface returns to the user an HTML page containing the public key of the document and the hash of the public key of the document. (For security and anonymity purposes, we may wish to configure this script later to never notify the user of the actual public key which was used to create his shares.) The user will need to keep the hash of the public key of the document in order to retrieve the document.

Retrieving a Document

The functionality for retrieving a document is provided through the same form and perl script that is used for inserting a file. The form provides input fields for whatever information it may need to retrieve the shares of a document and deliver them to the user. In particular, the user must provide the hash of the public key of the document, some remailer reply block, and optionally some public key with which the shares can be encrypted. This information is then sent using `post` to the user interface.

After receiving the public hash of the file and the remailer reply block from the CGI request, the user interface script builds a message for broadcast to all of the Free Haven servnet nodes. This message includes the hash of the public key of the document and requests that all shares having matching hashes be sent to the reply block. The user interface arranges for the message to be sent to all the nodes through the comm module. The Free Haven nodes that have the requested shares are encouraged, but not required, to send their shares to the included reply block. Because shares of the document (instead of the document itself) are sent to the reply block, Free Haven will provide a script for the user to recombine a document's shares.

Chapter 6

Communication Module and Protocols

[1] This section details the design and implementation of a communications module for Free Haven, serving as an interface between the anonymous publication system and its corresponding anonymous communications channel(s).

6.1 Module Design

An anonymous communications channel is used to carry messages between servnet nodes. In order to provide an abstraction for the specifics of our communication channel, we provide a communication module that interfaces with both the outer anonymous channel and "inner" *haven* module, which provides the actual document publishing, storage, and retrieval functionality.

There are a number of required operations for the communications module – sending messages, broadcasting messages, naming nodes for message delivery, adding nodes, and removing nodes – and desired goals – low latency, delivery- and routing-robust, resistant to attack, and decentralized. While many of these operations and goals are inherently part of the communications channel, the module should provide a flexible and robust interface layer between the two subsystems.

6.1.1 Supported Module Operations

Specifically, the *comm* module must support the following operations or characteristics:

[1] This chapter was originally written by Michael Freedman.

- The *comm* module should send data to *haven* when it becomes available from the communications channel.
- The *comm* module should send data to the communications channel when it becomes available from the *haven* module.
- Messages can arrive simultaneously from the communications channel.
- Messages can arrive from the *haven* module simultaneously with messages arriving from the communications channel.
- Messages on a socket can arrive in a delayed manner.
- The *haven* module can *fail* unexpectedly, but *comm* should still continue normal operation and reconnect to *haven* when it becomes available again.

6.1.2 Module Data Structures

The communications module has several associated data structures:

- **Incoming Feeder Queue:** List of $\{socket, filedesc, filename\}$ tuples that correspond to feeder programs currently being processed by the communications module.
- **Haven Message Queue:** List of messages that have arrived from the communications channel, awaiting transmission to the haven module.
- **Node Database:** Database of information stored for all known servnet nodes.
  Key: Hash of public key
  Value:
  - **Public key**
  - **Waiting message queue:** List of filenames queued for each node in the database, corresponding to outgoing messages from the haven module.
  - **Length of queue**
  - **Cost:** Total "weight" of waiting messages in queue
  - **Communications channel type:** Mixmaster, ZKS, etc.
  - **Address / routing info:** Reply blocks, pseudonyms, etc.
  - **Statistics:** Timeout and availability considerations

6.1.3 System Modularity

The Free Haven design stresses modularity between various pieces of the system – The *haven* module, the *comm* module, the *trust* module, the *crypto* module, the *ui*, and the actual communications channel – providing a strong separation of function. While the initial proof-of-concept implementation of Free Haven will assume a 1:1 *haven* to *comm* module relation, with both processes running on the same machine, this relationship is not necessarily fixed. Indeed, with only slight modification to the actual socket code, our design allows for several *haven* process to share a single network *comm* process interfacing with different communications channels. In order to support this scenario, we should not assume a secure connection across the *comm–haven* socket. Instead, all data across this socket uses both public-key encryption for security and base-64 encoding within data blocks to ensure proper user-formatting of information.

6.2 Module Implementation

In Chapter 2 we discussed the anonymity offered by various anonymous channels; in Chapter 8 we will suggest some other possible channel designs. The current implementation, however, utilizes the Mixmaster remailer [35]. Mixmaster was chosen because of its simple command-line interface, strong anonymity offered through Chaumian mixing, protection from traffic analysis and other attacks due to its high latency, message padding, and packet reordering and buffering within nodes. Furthermore, Mixmaster is freely available, making it quite suitable for an open project such as Free Haven.

The code base allows for the easy incorporation of different mixnets and other anonymous communications channels, by adding simple switching logic to the `do_transmit` function and specifying the proper mixnet type and address in the node database. Indeed, the only primitive that we require is a generalized `send` function. Therefore, a servnet node can actually specify the communications channel on which it wishes to communicate.

The *comm* module implementation also provides database flexibility. We provide an abstraction layer on top of the Node DB, separating any database-specific operations from the user. The GPL-released `gdbm` is currently used in the initial implementation; a relational database of greater functionality is being considered for further development.

6.2.1 Implementation Pseudocode

The communications module provides an "always-on" module to handle incoming and outgoing messages from a servnet node. The basic operation of the module is as follows:

- Create a non-blocking listening tcp socket for incoming communications for feeder programs. These *feeders* are system processes that will pass messages from the communications channel to the Free Haven communications module.

- Loop on the following control structure:

    1. Connect to haven socket. If this socket is not available, continue processing available information and try again upon next iteration.
    2. If data available on *haven* socket, process the outgoing message.
        (a) If message is of type `broadcast`, enumerate the list of non-dormant servnet nodes and `transmit` the message to each node.
        (b) If message is of type `transmit`, enqueue the message within the node's waiting message queue for later processing.
        (c) If message is of type `introduce`, add the corresponding information about a new node to the node database.
    3. If new *feeder* attempts to connect to non-blocking incoming socket, enqueue the new *feeder*.
    4. For all *feeder* sockets on which data is available:
        (a) Read all the information from the socket into the *feeder's* corresponding file.
        (b) If the *feeder* has reached EOF, place the file into the haven message queue and close the *feeder*.
    5. Send incoming messages to the haven module, extracting from haven message queue.
    6. Transmit waiting messages for a node, concatenating and padding to reach a total packet size up to a random or statically-assigned length. Any messages that cannot fit within this buffer remain in the waiting messages queue for later transmission.

/* Communications Handling Functions */
void handle_ports(int incoming_port, int listen_socket, int haven_port);
void create_feeder_entry(int new_feeder_socket);
void process_feeder_entry(struct feeder_t *feeder, struct feeder_t *prev_feeder);
void enqueue_haven_message(char *filename);
void process_freehaven_message(char *filename);
void process_internal_message(char *filename);
void send_file_to_haven(char *filename);
void exit_comm();
/* Transaction Functions */
int transmit(struct tag_t *tag_list);
int do_transmit(char *PK);
int broadcast(struct tag_t *tag_list);
/* Node Database Functions */
void initialize_nodedb(void);
datum construct_gdbm_entry(struct nodedb_entry_t *entry);
struct nodedb_entry_t reconstruct_nodedb_entry(void *noded_value);
int node_change_PK(char *hPK, char *newPK);
int node_add_new_node(char *hPK, char *PK, char *address, char *mixnet, int statistics);
struct message_queue_t* node_get_message_queue(char *hPK);
char* node_get_PK(char *hPK);
char* node_get_mixnet(char *hPK);
char* node_get_address(char *hPK);
char* node_get_busiest_PK();
int node_add_waiting_msg(char *hPK, char *filename);
int node_get_waiting_msgs(char *hPK);
int node_set_statistics(char *hPK, int statistics);
int node_get_statistics(char *hPK);
void free_node_msgs(struct nodedb_entry_t nodedb_entry);
void close_nodedb();

Table 6.1: Communications Module API

6.2.2 Design Discussion

The communications module itself requires access to the crypto module. Communications between the *haven* and *comm* modules should be ASCII-armored with base-64 encoding, as the messages use standard XML format with data delimited by begin $< tag >$ and end $< /tag >$. Using base-64 encoding stops an user from placing < or > symbols into the internal data block, which might confuse message parsing. Furthermore, communications between the *haven* and *comm* are encrypted, to provide for the option of distributed module processes and multiple *haven* processes per *comm* process.

The communications module also needs to perform any necessary message encryption or encoding necessary, based on the type of communications channel used. For the current Mixmaster implementation, the *comm* process performs layered encryption of the message data. This layered encryption method – or "onion routing" – is the standard Chaumian mix-net technique to ensure anonymity. First, the sender chooses a route through the mixnet: Source S → Node A → Node B → Node C → Destination D pseudonym. Second, the sender signs the message with its private key pk_S and possibly encrypts via the destination's public key PK_D. Then, the sender encrypts the message and next-hop information along the mixnet in reverse: encrypt with PK_C, then encrypt with PK_B, and finally encrypt with PK_A. Therefore, only the proper node in the mixnet can decrypt the top layer of the message, exposing next-hop information and the proper onion message to relay. In other words, the encryption layers are unwound (or peeled like an onion) during mixnet transmission, before the message is relayed to its destination. The destination reply-block or pseudonym specifies a path to an explicit servnet node. Once the message arrives at the destination servnet node, the node decrypts the message using pk_D and verifies the sender's signature.

6.2.3 Optimizations

The communications module performs several efficiency and anonymity optimizations. First, haven message queues are built up within the *comm* process, and iteratively dequeued and sent to *haven*, protecting against bursty transfers from the communications channel.

Second, broadcasts can be requested by the haven module by merely calling a `broadcast` primitive, as opposed to querying for a list of available nodes, extracting each node's route and encrypting the message according to that route, then performing the `transmit` operation on each message. A single broadcast call decreases the quantity of socket-level requests from $O(n)$ to $O(1)$, where n is the number of available nodes.

Third, messages are queued during `transmit` for each node. One node is chosen at random (while attempting to ensure semi-fairness among nodes) to transmit its messages along to anonymous communications channel. Messages are concatenated and padded to a specific or random length to protect against message volume attacks. The act of enqueueing and concatenating messages reduces the number of messages to a specific node, possibly adding some protection against traffic analysis that seeks sender/receiver linkability. Furthermore, the random choice of a node during that time iteration adds some latency to haven outgoing messages, especially under high load. This technique may also protect against some form of traffic analysis, especially in light of multi-step Free Haven protocols, such as trading or buddy 'squawking'.

Chapter 7

Adversaries and Attacks

While explaining the motivations behind an anonymous publishing system like Free Haven, we enumerated a number of possible adversaries, diverse in both goals and resources. Several of the types of attacks that may be employed cannot be handled merely through technology: these include social attacks on system security and servnet node operators, political attacks to discourage servnet use, and government and legal attacks to shut down nodes or arrest operators. The success of these attacks will often depend upon the political and jurisdictional clime of servnet nodes' physical location. On the other hand, the success of technical attacks – from individuals, organizations, corporations, or national security agencies – is contingent upon the system's security and robustness to attack.

We have defined anonymity in terms of our ideal anonymous publishing system in section 2. In doing so, we specified a list of protections to provide for system agents and operations. This section describes the various types of attacks an adversary or group of colluding adversaries might use against Free Haven, relating the effect of these attacks on the level of anonymity maintained.

There are three primary modes of attack: on the communcations channel, on the Free Haven servnet, and on individual files. As the security and anonymity of a system are only as strong as its weakest link, we have considered all three of these, and take appropriate countermeasures for many of these attacks.

7.1 Attacks on the Communications Channel

[1] Adversaries operating on the communications channel may seek to weaken one of the five types of communication anonymity: sender-anonymity, receiver-anonymity, unlinkability between sender and

[1] This section was written by Michael Freedman.

receiver, node-anonymity, and carrier-anonymity. We consider both passive and active adversaries, attacking nodes within the communications channel, internal links between nodes within the channel, and endpoint links from the channel to users (i.e., the sender or receiver, which are both servnet nodes in the case of Free Haven).

7.1.1 Communications Nodes

The following attacks assume an active adversary that controls one or more nodes within the communications channel:

- **Denial of Service Attack:** An "evil" node within the communications channel can selectively drop messages/packets that it receives at will.

  Prevention: There is basically nothing a system can do to stop a node from behaving in such a manner; however, users can occasionally "ping" various nodes to determine response time. If a node drops a sufficient number of packets and cannot differentiate between ping and data packets, users will come to realize that the node is not reliable and will stop using it.

  Many sources maintain statistical information on the reliability of mixnet nodes, especially Cypherpunk (Type I) and Mixmaster (Type II) remailers [35]. The most common of these networks use only a small number – a dozen or two – public remailers that are known. Naive denial of service attacks would be noticed.

- **Traceroute Collusion Attack:** A corrupt coalition of nodes within the system collude in order to trace certain messages through the communications channel. An "evil" node receives a message, knowing the IP address of both the last-hop and next-hop. Given the ability to collude with a sufficient number of nodes that have received the same message, the path through the communications channel can be traced, as well as ultimately finding the message sender or receiver.

  Prevention: An ideal anonymous communication system will be distributed, as any corrupted central system risks the exposure of both sender and receiver. For a distributed system, a route traversing k nodes preserves anonymity given a maximum of $k-1$ adversaries along this path. This protection also requires that adversaries cannot track a message across one hop, requiring both that the message changes across every node and adversaries cannot perform effective traffic analysis.

- **Cut-the-Channel Collusion Attack:** Similar to the traceroute attack, adversaries need to control a majority of nodes or bottlenecks within the communications channel. If these evil nodes drop packets and perform denial of service on a large scale, an adversary can watch which connections remain, recognizing more easily the normal communications path used between two users.

Prevention: Similar to other collusion attacks, a distributed system with many independent operators reduces the possibility of a large number of colluding node adversaries. System users can recognize the widespread failure of nodes and stop using the communication channel. Obviously, this presents a system-wide denial of service attack, affecting the users' overall trust in the system.

- **Traffic Mangling Attack:** To perform a traffic mangling attack, an adversary requires control of nodes at the edges of the communications channel. When an entry node receives a message from a sender, it mangles the message such that transmission or routing will occur properly, but an exit node on the other edge of the communications channel can recognize the mangled message. The colluding nodes can communicate outside of the normal channel, and establish sender-receiver linkability and IP correlation. Therefore, this attack does not require a large control over the system, such as the traceroute collusion attack, to link communicating agents.

  Prevention: There are two main defenses against this type of attack. First, message packets should be encrypted or encoded in such a way that any change by a node – the packet mangling itself – will be detected by other nodes, and the packet discarded. This defense relies on the assumption that not all nodes along the message's path are compromised. Many existing systems (e.g., Onion Routing, Freedom) use symmetric key link-layer encryption to counter a traffic mangling attack. Second, strong partial anonymity is a defense mechanism against this attack. Namely, if an "evil" node cannot determine whether it exists at the true edge of the communications channel (i.e., it cannot tell if the agent from which the message arrives is the initial sender), the node can only reveal some k-anonymous set of possible senders or receivers.

Form-based proxy systems (Anonymizer, LPWA) cannot really be analyzed in terms of these various forms of attack. Indeed, these systems basically rely on a single trusted third party: the proxy itself. If an adversary manages to take control of the proxy, both sender-anonymity and receiver-anonymity are lost, and linkability between the two is also established. Carrier- and node-anonymity are only relevant to distributed systems.

7.1.2 Communications Channel Links

This section describes a number of attacks that an adversary may perform on links within the communications channel. The adversary's goal is to determine a message's sender or receiver, or to provide linkability between the two agents. Many of these attacks hold the greatest risk to systems attempting to achieve full n-anonymity, where n is the number of system users. In the case of systems that provide partial anonymity, an adversary may gain information from traffic analysis or a similar attack. Yet, the adversary only reveals a k-anonymous set ($k < n$), describing an probabilistic distribution of anonymity, where k identities are below entropy threshold and thus "exposed." Let us consider the following attacks:

- **Computational Attack:** An adversary can sniff a link within the communications channel, and thus be able to read anything that is sent over that link. Presumably, both the data and transmission path are encrypted. To ensure the anonymity of sender and receiver agents, an adversary should not be able to easily determine this transmission path. Various levels of anonymity result from the security of the path's encryption and encoding. Obviously, this attack can also be performed at nodes within the channel or at its edges.

  Prevention: For naming schemes which rely on computationally-secure reply blocks or pseudonyms, an adversary with sufficient computing power can eventually decrypt the name and determine agent identity. If the communications channel relies on partial anonymity, successfully decrypting the name will only reveal a k-anonymous set of possibilities.

- **Message Coding Attack:** An adversary can trace or link a message that does not change its coding during transmission. If links can be passively monitored, the listener can determine the path that a message takes through the communications channel. Obviously, this attack can also be performed by colluding nodes within the channel, but this attack does not require that much penetration into the system.

  Prevention: An end-to-end encoding or encryption scheme is used by the sender, such that the message changes across each link in the communications channel. In a distributed system, a link-to-link scheme is not sufficient to prevent collusion attacks, as messages need to be different at the edges of the channel to protect sender/receiver anonymity.

- **Message Volume Attack:** An adversary can analyze traffic across a link and examine packet size. If a message of the same or similar length is detected traveling through various communications links, a global observer can determine the transmission path and ultimate sender/receiver.

  Prevention: All messages in the system should be of the same size or of a random size. Messages should therefore be padded with random bits, or concatenated with other messages and padded to the specified size.

- **Traceroute Replay Attack:** Within an anonymous communications channel, messages are transmitted to a given reply block or pseudonym of the receiver. An attacker listening on the link can record messages that pass by, and then attempt to forward traceroute the channel by flooding the system with the replayed message. Similarly, if some reply path or nym is given for the sender, the adversary can try to reverse traceroute the message by flooding the sender. The attacker can then perform traffic analysis to detect which links within the channel, or edges of it, see a rise in traffic. This rise in traffic suggests the message's route.

Prevention: To stop forward traceroute replay attacks, a nonce should be included in each message, and nodes should not resend a message that has already been sent. Nodes would be required to store a "graveyard" of used nonces to lookup. Also, an attacker must not be able to change the included nonce. The system can protect against reverse traceroute replay attacks by only including a way to reach the sender if a reply is necessary, as well as making this information only available to the receiver.

- **Intersection Attack:** An adversary may trace user identities by examining usage patterns over a long period. Similar to distinguishing characteristics of a speaker, users may manifest distinguishable behavior. For example, they may exhibit typical on-line/off-line periods, utilize similar resources over time, contact the same destinations or Internet sites regularly, and so on. If any distinguishing characteristics are transmitted (i.e., pseudonyms, Cookies), an adversary can link past and future communications to the specific user with greater certainty.

  Prevention: We cannot determine any protection against this type of attack. Others have questioned if this problem is even solvable [17].

- **Sniping and Cut-the-Internet-Backbone Attacks:** This attack is similar to the cut-the-channel collusion attack between nodes. An adversary has the ability to snipe specific links with the system for selective denial of service, or bring down large segments of the Internet to destroy inter-node links within the communications channel. The attacker can then see which connections of interest remain and perform traffic analysis.

  Prevention: This attack is beyond the resources of many individuals and organizations. A sniping attack might help an adversary perform traffic analysis to some degree; if the attacker has the ability to cut any link at will, they can eventually expose senders and receivers. However, national intelligence agencies are probably the only organizations able and willing to "Cut-the-Internet-Backbone."

7.1.3 Communications Channel Edges

This section describes a number of attacks that both active or passive adversaries may perform on edges of the communications channel. The edges of the channel correspond to links between a communicating agent (i.e., sender, receiver) and the immediate node within the channel to which it communicates.

- **Timing Attack:** An adversary can attempt to link a specific message between two parties by watching endpoint send and receive actions. Given the would-be transmission time of this message, the attacker can consider the correctness of these two endpoints.

  Prevention: A message can only be protected from a timing attack by hiding the message with others. The linkability between sender and receiver can obviously be established if only one message is transmitted within a certain time period. This protection is established by introducing latency, adding dummy messages, or reordering packets.

  Unless system load is extremely low, a timing attack is likely to expose linkability only with real-time services. Many systems – such as the original Chaumian mix – are based on `sendmail`, already having quite high and variable transmission time. However, systems designed to allow `telnet`, Web browsing, IRC, and other such services risk linkability from timing attack.

  Adding a variable delay decreases the ability to perform timing attacks given a reasonable system load. However, the system has a higher latency than necessary, and the system is still open to traffic analysis attacks, especially if the number of messages across the channel remains small.

  Dummy messages can be transmitted instead to remove this unnecessary latency, adding load to the system. If adversaries are given only a possible transmission duration, this solution increases the difficulty of correlating times to specific messages. Dummy messages are an end-to-end solution, whereas link-level garbage can be recognized and thereafter ignored by an adversary that watches the endpoints.

  Lastly, communication channel nodes can reorder packets when they are received. Nodes store n packets. Upon receiving more packets, the node chooses some k packets at random from this $(n + k)$ pool and sends them out.

- **Trickle Attack:** An adversary has complete active control of all the edges of the communications channel. The adversary stops all incoming messages from entering the channel except one. The next incoming message is not released until the first message is detected along some edge exiting the channel. As only one message is transmitted through the channel at once, the global observer can establish linkability between sender and receiver.

  Alternatively, an adversary can achieve active control over all the links of some internal node within the channel. This type of attack is more easily attained than system-wide control, and allows attacks as described in section 7.1.1.

Prevention: We cannot determine any protection against this attack for communications channels which provide an explicit mapping of names to sender/receiver agents. For systems which provide partial anonymity even after exposure, the "edge" of the communications channel only reveals a k-anonymous set of possible receivers.

- **Identification Flooding Attack:** An adversary can flood with the system with identifiable packets. A message can only remain hidden within the context of other known messages. During normal operation, each system user would send only one message during each time interval, thus producing an independent set of anonymous messages. However, if an attacker floods the system and fills a node's reorder buffer with n packets, it removes the node's defense against timing attacks and allows a certain message to be more identifiable.

  Prevention: A flooding attack is very difficult to defend against for practical Internet systems. Ideally, the system would be able to establish a unique, anonymous identity for each of the n users that send messages during one transmission interval. Performing adequate authentication of message senders while maintaining anonymity is a difficult problem. Possible solutions include the use of pseudonyms for partial anonymous systems, requiring that adversaries cannot control a significant number of pseudonyms. Similarly, some form of blind signature scheme can be used for anonymous authentication.

- **Traffic Flooding Attack:** Similar to the identification flooding attack, an adversary sends a large number of messages into the system to greatly increase traffic along certain paths. The adversary then proceeds to measure a rise in traffic along the communications channel's edges or along internal links. This form of traffic analysis suggests the route taken to reach a specific reply block or pseudonym receiver.

  Prevention: The system should ensure that an equal number of packets are sent between each link during some time interval, by either introducing dummy packets or latency. This protection is similar to that used for timing attacks. However, maintaining steady traffic along the edges of a communications channel is more difficult, given the possibility of very bursty traffic. The system can add dummy packets to the endpoint, but would then require client-side filtering.

- **Pseudonymity Marking Attack:** An adversary masquerades as a normal user and distributes unique names to other distinct agents in the system. These unique names correspond to different reply blocks or pseudonyms, such that the last hop of the transmission path is different for each name. The adversary can then correlate the last hop of the received message to a specific sender. Linking a sender agent to an individual user or IP address remains a separate problem, unless this can be determined during name distribution.

Prevention: This attack is only viable against a communications channel in which the receiver has an explicit transmission path, such as with remailer reply blocks, as opposed to multi-cast or random-walk functionality. Secondly, a sender can defend against this attack by getting the receiver's name from a third party – such as another trusted agent or some specified meeting-place for name distribution.

- **Persistent Identity Attack:** The persistent identity attack is not an attack per se, but rather an inherent anonymity weakness with any persistent naming infrastructure. If an adversary manages to disclose user information with any of the attacks we have enumerated, the adversary can correlate any future use of this name with the exposed individual.

  Prevention: System agents – senders and receivers – should use dynamic naming to provide forward anonymity, or rely on a partial anonymity scheme such that disclosure of agents only reveals a k-anonymous set of possibilities.

- **The Need for "Recipient-Hiding" Public Key Encryption:** We call a public key cryptosystem *recipient-hiding* if it is infeasible to determine, given a ciphertext, the public key used to create that ciphertext. The recipient-hiding property is *not* implied by the standard definition of semantic security (even with respect to adaptive chosen ciphertext attack). Moreover, it is not even achieved in common practical constructions. This has implications for mixnets which use reply blocks that are separate from the body of the message.

  To see that semantic security and recipient hiding are independent, consider any semantically secure cryptosystem C. Construct the cryptosystem C' which is just like C in every way, except that it appends the public key used to encrypt to every ciphertext. All messages produced by C' with the same public key are indistinguishable from each other if the messages produced by C are, and so C' is semantically secure – but it is the very opposite of recipient hiding.

  In practice, mail programs such as PGP tend to include the recipient's identity in their header information. Even if headers are stripped, David Hopwood has pointed out in the case of RSA that because different RSA public keys have different moduli, a stream of ciphertext taken modulo the "wrong" modulus will tend to have a distribution markedly different from the same stream taken modulo the "right" modulus. This allows an adversary to search through a set of possible public keys to find the one which is the best fit for any ciphertext, even if OAEP or similar padding is used.

  In mixnets which provide reply blocks, the reply block is often treated as opaque(for example Babel [41]) and prepended to the message to be sent. This means that the message is available for inspection by each intermediate hop with no processing at each hop; for this reason the message is often encrypted with the public key of the recipient. The point of a reply block is to provide a chain of mix nodes between the sender and the recipient, in which intermediate nodes are supposed to know neither the sender nor the receiver. If the recipient can be identified by simply inspecting the message, then every single intermediate node knows the destination, and knows approximately where it is in the reply block. This may leak an undesirable amount of information about the mixnet.

Prevention: Rivest suggested that randomized cryptosystems (such as the Goldwasser-Micali cryptosystem) might possess this property. Independently, Lysyanskaya and Wagner proposed a version of ElGamal in which all parties share the same modulus as a concrete example. There is a formal definition of recipient-hiding proposed on sci.crypt by Hopwood [43], with application to showing that a variant of Bellare, Abdalla, and Rogaway's DHAES scheme [1] achieves the recipient-hiding property, along with a variant of RSA in which moduli are generated to be close together. It seems that recipient-hiding cryptosystems may not be that hard to construct, once the requirement is recognized. The problem is that because previous systems were not designed with anonymity in mind, commonly deployed cryptosystems may not be recipient-hiding.

7.2 Attacks on the Infrastructure and Documents

7.2.1 Attacks on Documents or the Servnet

- Attack the time-synchronization protocol to make files expire earlier than expected.

  Prevention: We rely on the ability of servnet node operators to maintain accurate or near-accurate time on their systems. Presumably if an adversary has the capacity to successfully attack a system's time server or the link to the time server, then the adversary can do other attacks as well. This is something for servnet node operators to bear in mind, though, since some additional checks to make sure the time cannot be changed by very much delta per time period could well make Free Haven more robust.

- Go 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, we hope that the servnet operator will be able to claim plausible deniability over knowledge of the data stored on his computer.

- Physically destroy a servnet node, to attack the integrity of the data in the network.

  Prevention: Because we are breaking documents into shares and only k of n shares are required to reconstruct the document, losing some fraction of the servnet nodes should not affect availability of documents in the system.

- Claim that the servnet or mixnet concept is patented or otherwise illegal. Sue the Free Haven Project and any known node administrators.

  Prevention: We rely on the notion of jurisdictional arbitrage to maintain the integrity of the servnet in the face of loss of some parts of it due to legal or government attacks. Information illegal in one place is frequently legal in others. Global oppression of a piece of information is relatively rare. The 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.

- Attack the generosity of individuals: increase the personal cost of running a servnet or mixnet node, either by adding a monetary cost to moving large quantities of data around, or by adding a bad reputation such as "harboring terrorist data and kiddie porn".

  Prevention: Owning a node of this service is going to put an administrator in a potentially tricky situation. We rely on the Hacker ethic and a commitment to free information flow to provide volunteers who believe these risks are worthwhile.

- Denial of service attack on the servnet: continued flooding of queries for data or requests to join the servnet may use up all available bandwidth and processing power for a node.

  Prevention: In short, we must assume that our communications channel has adequate protection and buffering against this attack. Most communications channels we are likely to choose will not protect against this attack. This is a real problem.

- Trade until a sufficient fraction of an objectionable document is controlled by a group of collaborating servers, and then destroy this document.

  Prevention: We rely on the overall size of the servnet to make it statistically unlikely for any given server or group of collaborating servers to obtain a sufficient fraction of the shares of any given document. We really on the accountability from the buddy system to make it unprofitable to destroy a share without also destroying its buddy. This attack is actually more complicated than just hoping to possess enough shares of a document at a given instant in time: adversaries can obtain control over certain shares and then refuse to trade those shares away. This means that an adversary might over time increase the fraction of the document that he controls. The timing and frequency of trades must be modelled, based on the expected size of the servnet, to choose parameters that prevent this attack.

- Conspire to make a cause "unpopular". Convince servnet node administrators that they don't want to be hosting data for these unpopular causes, and that they should manually prune their data.

Prevention: We rely on the judgment of servnet administrators to choose to support any and all content, if they can get away with it in their jurisdiction. We 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.

- Insert false shares of a file into the servnet.

  This is not really an attack per se, because there is no such thing as a false share. Any set of bits at all is an acceptable share, if you can convince another node to accept a trade for it and provide a receipt. On the other hand, trading away a share implies a contract to store some other share. Thus the ability to insert shares, whether 'false' or valid, is limited by the ability of that server to provide space for the share it receives in return.

  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.

7.2.2 Attacks on Anonymity

In addition to the above attacks which focus on reducing availability of documents within Free Haven, there are also a number of attacks which focus on increasing knowledge of the identity of one of the agents in Free Haven.

- Attacks to determine the identity of a reader include:

  - Spread a Trojan horse, worm, or virus and look for signs that somebody has been infected.
  - Develop a customized virus which automatically contacts a given host upon execution.
  - Become a server, and provide extra information for document responses. We hope the mixnet will protect against most attacks of these sort, but the mixnet cannot protect against end-to-end attacks.
  - 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.
  - Include mime-encoded URLs in a document, and exploit reader software to automatically load these URLs.
  - Offer a large sum of money for information leading to readers of a given document.
  - Attack to find people interested in a particular document: claim to have one, and see who requests it.
  - Become a server, and simply monitor queries and record the source of each query.

  - Correlate readers based on the material they download; try to build statistical profiles and match them to people (outside Free Haven) based on activity and preferences. This would develop into a directed marketing campaign similar to Amazon's: "People who have downloaded this share may also like the following shares." This last attack is perhaps the most insidious one, since corporations with a lot of resources might want to take advantage of internet publication services to gain more information about users; we prevent this attack by using each reply block for only one transaction.

- Attacks to determine the identity of a server include:
  - 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.
  - Spread a Trojan Horse or worm which looks for Free Haven servers and examines or reports which shares they are currently storing.
  - Become a servnet node, and collect information as other nodes send us mail or lists of nodes.
  - Attempt to map servnet topology, and correlate nodes that are 'close' in the servnet with nodes that are 'close' geographically.
  - Become a node on the mixnet, and attempt to correlate message timing with trade requests or trust broadcasts.
  - Offer a large sum of money for information leading to the current location of a given document or share in the Servnet.
- Attacks to determine the identity of a publisher include:
  - Become a server and log publishing acts. Correlate source or timing.
  - Look at servers who might recently have published a document, and try to determine who has been communicating with them recently.
  - Offer a large sum of money for information leading to the current location of a given document or share in the Servnet.

We avoid or reduce the threat of many of these attacks by using a mixnet for our communications. This prevents most or all adversaries from being able to determine the source or destination of a given message, or correlate either endpoint of a set of messages. Other attacks, including social attacks, are much more difficult to anticipate and protect against. Agents and users who follow the protocol and use basic common sense will be more likely to maintain their anonymity.

7.3 Attacks on the Trust System

[2] There are a variety of attacks which are possible on the Free Haven system. Many of these attacks are far outside the scope of the Trust Module: social attacks on system security and servnet node operators, political attacks to discourage servnet use, government and legal attacks to shut down nodes or arrest operators, denial of service attacks on the communications anonymous channel, and attacks on the infrastructure of the server network.

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 is not 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.

7.3.1 Simple Betrayal

The simplest attack is this: Become part of the Servnet, earn trust, then betray it by deleting files before their expiration dates. The trust economy is designed to make this as unprofitable as possible. The size-time currency means that a corrupt node has to donate at least as much to the Free Haven as it removes. This 50% useful work ratio is a rather loose lower bound — it requires duping a great number of high-metatrust nodes into recommending you.

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

7.3.2 Buddy Coopting

It is possible for a corrupt node to gain control of both a share and its buddy; at this point it can delete one of them without repercussions. This means that corrupt nodes can defeat the buddy system by capturing both buddies, then deleting them.

A possible work-around to this attack involved separating the contact addresses for trading and for buddy checking, preventing corrupt nodes from acquiring the buddies of the shares they already have. Such an approach adds a great deal of complexity, and opens other attack avenues.

7.3.3 Trading Receipt Games

The receipts used in trading are a complicated mechanism, and we have no formal system for talking about how they interact. While we believe that the signed timestamp makes it clear who did what

[2] This section was written by Brian Sniffen.

and when, it is possible that some attacks exist, likely involving multi-node adversaries engaging in coordinated bait-and-switch games with target nodes.

7.3.4 Pollution

An adversary can join the server network, then trade away garbage for valuable data. A sufficiently wealthy adversary could even purchase a series of very large drives, then trade away enough garbage to have the majority of the data in the server network on his drives, subject to deletion.

We have no defense against this attack. However, any adversary capable of perpetrating the above attack against a widely-used Free Haven is equally capable of many cheaper, easier, non-technical attacks.

7.3.5 False Referrals

An adversary can broadcast false referrals, or direct them to specific hosts. The metaconfidence system combined with the single-reporting policy provide somewhat of a guard against this. Based on field tests of Free Haven, we may need to switch to a policy of ignoring referrals which do not have receipts.

7.3.6 Entrapment

There are several ways in which an adversary can appear to violate the protocols. When someone points this out, the adversary can present receipts which show him wrong and accuse him of the above attack.

There is no defense in the present implementation against this attack; a more thorough system of attestations and protests is necessary.

Chapter 8

Future Works

8.1 Communications Channels

[1] A "standard" design for an anonymous communications channel is very much an open question. In section 2, we specified the requirements for an ideal anonymous communications channel, and considered how current works fulfill these requirements. In general, there are various considerations when designing an anonymous communications channel:

- Low latency
- Delivery robustness
- Resistance to traffic analysis and similar attacks
- Types of anonymity provided:
    - Anonymity vs. pseudonymity
    - Full vs. partial anonymity
    - Computational vs. information-theoretic anonymity
    - Perfect forward anonymity

Some of these goals are conflicting in nature. Systems which provide low latency and strong delivery robustness are generally more open to traffic analysis and other such attacks, as messages are routed quickly and sometimes repetitively through the channel. Still, while Free Haven stresses anonymity over availability, we would prefer a design which provides latency in the realm of seconds or minutes, as opposed to hours or days. If a high latency were to endure, Free Haven usage would

[1] This section was written by Michael Freedman.

be constrained to publishers and readers which specifically require our strong notions of anonymity. Similarly, lossyness within the channel degrades system performance. Free Haven can be designed to handle communications lossyness for file reconstruction by a robust information dispersal algorithm. Still, this condition has possible effects on the trust network and trading/buddy protocols, when the loss is not due to server failure, but rather to an unreliable communications channel. In this section, we present some alternative designs for an anonymous communications channel.

8.1.1 Garlic Routing

The concept behind Garlic Routing is similar to a mix-net. A sender encodes routing information in a series of layered encryptions, forming an "onion" of encrypted information. Each node along the route decrypts the outer layer of the onion, exposing the next layer and determining the location of the next hop. Eventually, the entire onion is peeled, and the message reaches its destination. A garlic packet looks similar to an onion packet, until it is unwrapped. A node then finds several garlic bulbs to transmit, instead of the normal single onion. Each bulb is a viable path-to-destination from that intermediate node, therefore providing several routes. Earlier intermediate nodes would have no knowledge of the path or existance of these newly exposed routes.

Garlic routing provides a few benefits. Delivery reliability and robustness is therefore increased through path redundancy. Reply-block encoding can be implemented efficiently in terms of size, as reply blocks will only grow linearly with the total number of nodes in onion and garlic routes. The encryption of header information can be performed using a hybrid scheme: all garlic bulbs within a layer are encrypted with the same symmetric key, which is then encrypted with each garlic node's public key. Therefore, the size of a garlic packet containing n bulbs is only $L + k * n$, where L is the size of the normal onion layer and k is the symmetric key length. A similar hybrid encryption scheme is used by PGP.

Based on the concept of a Chaumian mix-net, a garlic-routing mix-net provides computational anonymity based on the strength of encryption used on the garlic. Reply-blocks for the channel provide a route to an explicit destination; thus disclosure of the information will specify a receiver. This differs from our definition of partial anonymity schemes, where exposure of information will only yield a k-anonymous set of possibilities.

8.1.2 Iterative Exposure

The concept of Iterative Exposure is similar to that of a mix-net: each path node only knows the last hop from where it received the packet and can only expose the next hop to where it should send the packet. However, we do not use an layered encryption scheme. Header information is a list of entries, each one encrypted to a specific node, containing a simple message such as "Node A: send to node B." As with Garlic Routing, we can help provide delivery robustness through redundancy with a simple change: "Node A: send to nodes B, C, D." Entries in the routing list are randomly ordered. A node iterates through the list, and attempts to decrypt each entry and check if the entry corresponds to itself. Once determining the proper next hop information, the node can reorder the routing list to protect against message coding attacks. To adequately protect against this type of attack, some further change to the message itself would be required at each hop.

A destination can either provide a reply-block for a specific source, in which the source node has an encrypted element in the list, or a generic public reply-block. For the latter, one element needs to be publically readable so that a source knows where to "pick up" the path: a source can either send a message to this node as the first hop, or the source can provide its only anonymous path to this node by adding reply list entries. Admittedly, this node will lose carrier-anonymity to adversaries. However, such a generic reply block can protect against pseudonymity marking attacks, and it allows both the sender and receiver to specify paths accordingly to their own anonymity requirements.

8.1.3 Alien Conspiracy Net

The Alien Conspiracy Net seeks to make computational and traffic analysis attacks more difficult by specifying a naming scheme that does not yield an explicit destination. The advantage of this scheme is predicated on the benefits of partial anonymity: an adversary can only determine a k-anonymous set of destination possibilities.

Joining an existing net

A node generates a series of n identical tokens, each of b randomly-chosen bits. This sequence of tokens is the node's address, and is kept private. To join an existing network, a node connects to a few existing public nodes and offers to trade one token with each of its neighbors. The node removes the token traded away from its address and replaces it with the neighbor's token. The node also adds the token traded away and the one received as its approximation of the neighbor's address.

Node Behavior

Nodes will occasionally receive messages, most likely in the form of encrypted data. Whether or not the node can read the data, it should always forward it properly to neighboring nodes to protect against traffic analysis of channel edges.

At the top of the message will be a series of tokens. A node compares these tokens to the approximation of the address of each neighbor, calculating the number of tokens in common with each. The node picks, at random, some subset of the nodes with the greater commonality and sends them the message.

Partial-Anonymous Naming of Nodes

In order for the protocol to work, tokens need to continue to spread throughout the network. Thus, each node should offer a trade to each of its neighbors within some user-set delay. The token chosen to trade is selected at random from the node's address, so it may not be one of the node's own tokens.

Eventually, the above protocol would lead to a completely homogenous set of tokens. This would make message delivery problematic. Thus each node has a second user-set delay between mutations. When that delay has passed, the node replaces portions of its address with its own tokens.

A node publicizes its address by exposing some $\log n$ of its address tokens. A sender addresses a message to this $\log n$ address. Therefore, a node's public address yields partial anonymity: many such combinations result from the node's actual address, and many addresses can yield this set of $\log n$ address tokens. As destinations forward their own messages as well, an adversary cannot determine the actual destination only given the message's destination address. Obviously, messages are given a TTL (time-to-live).

Network Topology

This protocol relies on token gradients across the network. The message travels from regions of low potential to regions of high potential. For this to happen, the network probably requires a properly-connected graph. If nodes connect to large numbers of other public nodes, or edges between nodes are randomly distributed, the resulting network may not have clear token gradients. We have not yet determined what type of network topology is actually required for adequate message delivery, nor considered protocols to yield a desirable topology via distributed communications and control.

8.1.4 Zones

The Crowds system organizes users into collections of nodes called crowds, achieving partial-anonymity for non-local adversaries and sender-anonymity within each crowd. Crowds, as proposed by AT&T Research, focuses on anonymous Web browsing, in which users communicate with specific end servers. Built onto the Crowds model, the Zones system seeks to achieve anonymous communications between two users instead.

Communications from a sender are routed through the sender's zone as specified by the Crowds protocol. When the message is dispatched from a crowd node, however, it is not sent directly to an end server. Instead, the message is dispatched to another zone. Once within the destination zone, the message is either multi-casted to all nodes or forwarded around via some random walk. Each node attempts to decrypt each message received to determine if it is the proper receiver. If random-walking is used, the proper receiver still wants to forward the message to protect against zone traffic analysis. This being so, a message's TTL has to be sufficiently high such that a random-walk will probabilistically find the proper receiver. This type of distribution strategy provides k-anonymous receiver-anonymity, where k is the size of the receiver's zone.

8.1.5 Small Worlds Model

Social networks display two characteristics which initially may appear to be contradictory. Firstly, social connections display clustering, whereby friends are likely to share the same group of friends. Secondly, they exhibit what has been termed by Stanley Milgram as the "small worlds effect" [65]. Namely, any two people can establish contact by going through only a short chain of intermediate acquaintances. Milgram proposed that all people in the world are separated by six intermediaries on average; this effect is better known as "Six Degrees of Separation" or the "Kevin Bacon Game."

Network Construction and Transmission

Mathematicians have begun studying sparse networks to prove the small worlds effect [101]. Using these models for network topology, we can construct an anonymous communications channel built on this network routing principle. Crowds or zones mimic small worlds of friends, without the necessary long-range connections that provide the small worlds effect. Therefore, we can achieve this effect by constructing zones wherein people also specify a few connections to users in other zones.

A node sends a message to all of its friends, who in turn propagate the message to the rest of their friends. While friends share many connections due to the clustering effect – explicitly formed by the creation of zones – the long-range connections allow the propagation of messages through the network. This model places a large load on the system, especially given the high degree of connectivity in most networks. We have also considered an alternative "directed" propagation of messages according to some greedy heuristic, such as a Hamming distance or the protocol we describe in the Alien Conspiracy Net.

Network Anonymity

The anonymity of this communications channel is based on the presumed innocence of the Crowds system. An adversary within the network cannot determine whether a message received originated from its last hop, or was merely forwarded by that node. As the number of nodes that may be involved in the message's path increases, the innocence of the last hop becomes probabilistically greater [85]. The system does not protect the anonymity of senders from a global observer.

K-anonymous receiver-anonymity can be achieved if receivers forward the message along. For simple broadcast networks, k is equal to the number of nodes in the entire network; for directed multi-cast networks, k is the number of nodes that received the message. With large values of k

and a high propagation of message transmissions, adversaries have a difficult time performing traffic analysis.

Messages should be encrypted to the receiver such that only the receiver can tell if the message is meant for her, such that an adversary cannot simply compare some naming scheme to link messages to receivers. A small worlds model likely requires a low-latency communications channel, such as that built directly on TCP/IP, due to the high bandwidth required. Delivery robustness is assured also by the redundancy of paths to any destination. To protect against message coding and message volume attacks, messages can be point-to-point encrypted and padded to the same or some random size. Reordering messages within a given node helps protect against timing attacks similarly to mix-nets. Like other schemes, we still witness the trade-off between a low latency channel and a resistance to traffic analysis.

Gnutella uses a similar six-degrees of separation model for network communication. Nodes broadcast a received messsage to all their friends, and messages include an explicit TTL to stop infinite looping. Messages are not encrypted or padded.

8.2 Publication Systems

Along with the need for more research on communications channels, there are a number of other issues we should address more thoroughly. These include:

Formalizing Anonymity The notions that we present in Chapter 2 are an excellent beginning for talking about the amount of anonymity a publishing system provides. However, we still have no real formal definitions for these concepts. By thoroughly enumerating the capabilities of potential adversaries, along with describing what their goals might be and what it means for them to achieve each goal, we could gain much more insight into the nature of anonymity and how it relates to other notions such as confidentiality.

Ideal Mixnet Many of the aspects of anonymity that we hope to achieve are going to be very difficult without a more anonymous communications channel. In order to provide a more anonymous publication service, we need to solve the mixnet design problem.

Deployed Mixnet Apart from developing the design for a better mixnet, we need to actually get a functional communications channel out on the internet. Without a large number of nodes running and available in a free environment, our anonymity requirements cannot be achieved. It would probably be very effective to distribute binary packages (e.g., in `RPM` format) to ease installation.

Deploying, Testing, and Evaluating Free Haven is only a set of designs for an anonymous publishing service. We discovered a number of important ideas while implementing our proof of concept; however, actually finishing and testing the implementation would teach us a lot more about how stable and reliable Free Haven will be.

The buddy system Free Haven provides a much stronger level of anonymity than related works; however, this amount of anonymity requires us to include some method of accountability to ensure that servers are behaving according to protocol. The current buddy system is insufficient for a number of reasons. We must find a better accountability mechanism.

Flooding protection Because we are using a mixnet which emphasizes anonymity, there are currently no flooding protection mechanisms in place. We need to devise some intelligent ways of bandwidth-throttling, to encourage more people to be willing to run mixnet or servnet nodes.

Modelling to determine optimal parameters We should describe some mechanisms for modelling Free Haven based on various parameters of k, n, number of documents, frequency of trading, percentage of malicious nodes, etc. Building a good model for Free Haven that is extensible enough to cover other anonymous publishing systems such as Freenet would benefit the community as well.

Alternate Payment Schemes We currently avoid the requirement for payments – instead, our 'economy of trust' rewards successfully storing data with the offer to store data in return. We should investigate alternate mechanisms for decentralized currency and payment.

Education and legal awareness Although there are a few papers (e.g. [46]) which succinctly and accurately summarize case law and current legislative opinion about anonymity on the internet, most programmers and other enthusiasts are disturbingly ignorant on this topic. Collecting and coordinating an easily readable archive of such documents would be a great benefit to the community.

8.3 Trust

[2] There are several areas in which this Trust Module falls short. The Free Haven group hopes to resolve most of them over the next few months, as we work towards our first public release. Others require advances in the theory of cryptographic anonymity:

Trust Algebra There is no good system for formally and rigorously talking about trust and approval. We need one in order to be certain that the trust system does anything like what we

[2] This section was written by Brian Sniffen.

are hoping for.

Entrapment Defense The current belief in referrals is somewhat naïve. We need better support for logging various referrals and integrating their results. The best approach is beginning to look like some form of linear algebra to deal with the loops of trust and metatrust.

Deployment Many of the questions we have about Free Haven and the Trust Module cannot be answered with thought experiments. We need to get the system deployed and working in a test environment to see where it is attacked and where it breaks. This test system needs to be clearly marked as potentially unsafe.

Chapter 9

Conclusions

Free Haven provides a service that is not currently available from any other project or application. Web pages available from the internet have their source easily evident. Usenet articles do not reliably reach all readers, and are subject to unpredictable expiration from disapproving administrators or simply due to space constraints.

By providing a stable and distributed service for anonymous publishing and anonymous reading, we provide dissidents with more powerful tools for both communication and publication. We believe in – and provide – a stronger notion of free speech than simply the ability to make government-sanctioned statements. Overall, we believe that providing individuals with the power to speak in a free, persistent, and untraceable manner is well worth the risk that the system could also be used for malicious or otherwise immoral activities.

Designing a good system for robust anonymous publication is hard. Providing sufficient accountability without sacrificing this anonymity is hard. Designing a good communications channel is hard. Designing a good trust system is hard. Modelling and anticipating the capabilities of future adversaries is hard. Solving all of these problems for a specific situation and then integrating the solutions into a robust functional implementation is much harder still.

We have provided a good foundation for a solution to each of these problems. By deriving basic definitions and notions for anonymity in a decentralized publishing system, and enumerating and addressing the capabilities and attacks that an adversary might employ in trying to break the system, we pave the road for a more thorough and broad approach to analyzing the success and protection of various related works, including Gnutella, Freenet, and Publius.

The Free Haven design is an important step toward answering a problem which is not currently being addressed: namely the creation of a strongly anonymous content-neutral decentralized publication system. Whereas related projects currently provide at best incrementally better anonymity relative to services like Napster, Free Haven provides computational anonymity for all three agents of the system: publishers, servers, and readers. We do not expect Free Haven to be as popular or immediately useful as services like Freenet, where losing anonymity might at worst spark a legal battle. However, we do believe that a service such as Free Haven is vital for foreign political dissidents and other activists who do not have the luxury of failure.

Appendix A

Acknowledgements

Quite a few people discussed the ideas and designs contained herein. They include:

- Michael Freedman for his brilliant and dedicated work on the communications channels research, as well as his contributions to the Free Haven design discussions.

- David Molnar for his unique ability to always know the paper to look for, or who wrote which articles about a given topic. David was the voice of theory in this group, which says a lot for him since I'm a theory grad student. He was also responsible for the enormous appendix enumerating the timeline of Communications Channels works.

- Brian Sniffen was very helpful in providing different perspectives on a number of the topics that we covered. He provided the design and background for the trust module and the original draft of the infrastructure and trust attacks.

- Joseph Sokol-Margolis for volunteering his time to helping keep our design sane. Joseph was very useful for considering attacks and other 'unintended' uses of the system.

- Todd Kamin took our notion of how to build a user interface for Free Haven, and fleshed it out into an actual design.

- Nathan Mahn during several different meetings saved the notion of the buddy system from derailing and completely collapsing. I'm pretty sure this is a good thing.

- Susan Born for her unending work at editing the various documents and subdocuments that comprise this thesis. Without Susan, we would still be missing a section or two.

- Ross Anderson for providing the initial inspiration for Free Haven, and also for providing some comments and responses to the overall design.

- Ian Brown (of the cypherspace datahaven design), Ian Clarke (of FreeNet), Ian Goldberg (of Zero Knowledge), Ian Marsh (of Jetfile), and Ian Hall-Beyer (of Gnutella) for providing the mystery of the 'Ian Conspiracy': why are the leaders of all these projects named Ian?

- Professor Rivest in his role as my thesis advisor for helping me to chip away many of the extra layers of confusion over my ideas. Rivest was extremely useful at providing perspective for why I choose certain design decisions, and which decisions are most important.

Appendix B

Anonymous Communications Channels

[1] We earlier described several major implementations of anonymous communications channels. This appendix serves to give a more detailed survey of research and development in the area of anonymous communications. Some of these projects are not implemented; some exist more as a proof-of-concept by their respective designers; and still others repeat design and functionality provided by like systems.

We review three main types of design: proxy-servers, mix-nets, and other anonymous communications channels.

B.1 Proxy Services

Proxy services provide one of the most basic forms of anonymity, inserting a third party between the sender and recipient of a given message. Proxy services are characterized as having only one centralized layer of separation between message sender and recipient. The proxy serves as a "trusted third party," responsible for sufficiently stripping headers and other distinguishing information from sender requests.

Proxies only provide unlinkability between sender and receiver, given that the proxy itself remains uncompromised. This unlinkability does not have the quality of perfect forward anonymity, as proxy users often connect from the same IP address. Therefore, any future information used to gain linkability between sender and receiver (i.e., intersection attacks, traffic analysis) can be used against previously recorded communications.

Sender and receiver anonymity is lost to an adversary that may monitor incoming traffic to the

[1] This appendix was written by David Molnar and Michael Freedman.

proxy. While the actual contents of the message might still be computationally secure via encryption, the adversary can correlate the message to a sender/receiver agent.

This loss of sender/receiver anonymity plagues all systems which include external clients which interact through a separate communications channel – that is, we can define some distinct edge of the channel. If an adversary can monitor this edge link or the first-hop node within the channel, this observer gains agent-message correlation. Obviously, the ability to monitor this link or node depends on the adversary's resources and the number of links and nodes which exist. In a proxy system, this number is small. In a globally-distributed mixnet, this number could be very large. The adversary's ability also depends on her focus: whether she is observing messages and agents at random, or if she is monitored specific senders/receivers on purpose.

B.1.1 Anonymizer.com

The Anonymizer was one of the first examples of a *form-based web proxy* [7]. Users point their browsers at the Anonymizer page at `www.anonymizer.com`. Once there, they enter their destination URL into a form displayed on that page. The Anonymizer then acts as an `http` proxy for these users, stripping off all identifying information from `http` requests and forwarding them on to the destination URL.

The functionality is limited. Only `http` requests are proxied, and the Anonymizer does not handle cgi scripts. In addition, unless the user chains several proxies together, he or she may be vulnerable to an adversary which tries to correlate incoming and outgoing `http` requests. Only the data stream is anonymized, not the connection itself. Therefore, the proxy does not prevent traffic analysis attacks like tracking data as it moves through the network.

B.1.2 Lucent's Proxymate

Chaining multiple proxies together by hand is a tedious business, requiring many preliminaries before the first web page is reached. Lucent's Proxymate software automates the process[59]. The software looks like a proxy sitting on the user's computer. By setting software to use the Proxymate proxy, the user causes the software's requests and traffic to go to the software, which then automatically negotiates a chain of proxies for each connection.

B.1.3 Proxomitron

Another piece of software which helps manage many distinct proxies in a transparent manner is Proxomitron[83]. In addition to basic listing and chaining of proxies, Proxomitron allows users to write filter scripts. These filters can then be applied to incoming and outgoing traffic to do everything from detecting a request for the user's e-mail address by a web site to automatically changing colors on incoming web pages.

B.2 Chaumian Mix-nets

The project of anonymity on the Internet was kicked off by David Chaum in 1981 with a paper in Communications of the ACM describing a system called a "Mix-net." This system uses a very simple technique to provide anonymity: a sender and receiver are linked by a chain of servers called Mixes. Each Mix in the chain strips off the identifying marks on incoming messages and then sends the message to the next Mix, based on routing instructions which encrypted with its public key. Comparatively simple to understand and implement, this Mix-net (or "mix-net" or "mixnet") design is used in almost all of today's practical anonymous channels.

B.2.1 Chaum's Digital Mix

Chaum's original paper introduced the basic concept of a Mix as a sort of "permutation box." On the incoming side is a list of messages representing the messages which have arrived at the Mix server, each of which is identified with a particular sender. On the outgoing side is a randomly permuted list of messages, which have lost their identification with the sender. The assumption is that if the Mix works correctly, no adversary can do better than guessing to link an incoming message with an outgoing message.

B.2.2 ISDN Mixes

Chaum's original Digital Mix was described in terms of a series of Mix nodes which passed idealized messages over a network. The first proposal for the practical application of mixes came from Pfitzmann et. al. [80], who showed how a mix-net could be used with ISDN lines to anonymize a telephone user's real location. Their motivation was to protect the privacy of the user in the face of a telephone network owned by a state telephone monopoly.

Their paper introduced a distinction between *explicit* and *implicit* addresses. An explicit address is something about a message which clearly and unambiguously links it to a recipient and can be read by everyone, such as a To: header. An implicit address is an attribute of a message which links it to a recipient and can only be determined by that recipient. For example, being encrypted with the recipient's public key in a recipient-hiding public key is an implicit address.

B.3 Remailers: SMTP Mix-nets

Until the rise of proxy-based and TCP/IP-based systems, the most popular form of anonymous communication was the *anonymous remailer*: a form of mix which works for e-mail sent over SMTP. Remailers are informally divided into three categories, called Type 0, Type 1, and Type 2.

B.3.1 Type 0: anon.penet.fi

One of the first and most popular remailers was anon.penet.fi, run by Johan Helsingius. This remailer was very simple to use. A user simply added an extra header to e-mail indicating the final destination, which could be either an e-mail address or a Usenet newsgroup. This e-mail was sent to the anon.penet.fi server, which stripped off the return address and forwarded it along. In addition, the server provided for return addresses of the form "anXXXX@anon.penet.fi"; mail sent to such an address would automatically be forwarded to another e-mail address. These pseudonyms could be set up with a single e-mail to the remailer; the machine simply sent back a reply with the user's new pseudonym.

The anon.penet.fi remailer is referred to as a Type 0 remailer for two reasons. First, it was the original "anonymous remailer." More people used anon.penet.fi than are known to have used any following type of remailer. Exact statistics are hard to come by, but X number of accounts were registered at penet.fi, and only Y are currently registered at nym.alias.net.

Second, anon.penet.fi did not provide some of the features which motivated the development of "Type I" and "Type II" remailers. In particular, it provided a single point of failure and the remailer administrator had access to each user's "real" e-mail address. In general, any remailer system which consists of a single hop is considered Type 0.

This last feature proved to be the service's undoing. The Church of Scientology, a group founded by the science fiction writer L. Ron Hubbard, sued a penet.fi pseudonym for distributing materials reserved for high initiates to a Usenet newsgroup. Scientology claimed that the material was copyrighted "technology." The poster claimed it was a fraud used to extort money from gullible and desperate fools. Scientology won a court judgment requiring the anon.penet.fi remailer to give up the true name of the pseudonymous poster, which the operator eventually did. This incident, plus several allegations of traffic in child pornography, eventually convinced Johan Helsingius to close the service in 1995[42].

Services similar to Type 0 remailers now exist in the form of "free e-mail" services such as Hotmail, Hushmail, and ZipLip, which allow anyone to set up an account via a web form. Hushmail and ZipLip even keep e-mail in encrypted form on their server. Unfortunately, these services are not sufficient by themselves, as an eavesdropping adversary can determine which account corresponds to a user simply by watching him or her login.

B.3.2 Type 1: Cypherpunks Remailers

The drawbacks of anon.penet.fi spurred the development of "cypherpunks" or "Type 1" remailers, so named because their design took place on the cypherpunks mailing list. This generation of remailers addressed the the two major problems with `anon.penet.fi`: first, the single point of failure, and second, the vast amount of information about users of the service collected at that point of failure. Several remailers exist; a current list can be found at the Electronic Frontiers Georgia site [35] or on the newsgroup alt.privacy.anon-server.

Each cypherpunk remailer has a public key and uses PGP for encryption. Mail can be sent to each remailer encrypted with its key, preventing an eavesdropper from seeing it in transit. A message sent to a remailer can consist of a request to remail to another remailer and a message encrypted with the second remailer's public key. In this way a chain of remailers can be built, such that the first remailer in the chain knows the sender, the last remailer knows the recipient, and the middle remailers know neither.

Cypherpunk remailers also allow for *reply blocks.* These consist of a series of routing instructions for a chain of remailers which define a route through the remailer net to an address. Reply blocks allow users to create and maintain pseudonyms which receive e-mail. By prepending the reply block to a message and sending the two together to the first remailer in the chain, a message can be sent to a party without knowing his or her real e-mail address.

B.3.3 Type 2: Cottrell's Mixmaster

While Cypherpunk remailers represented a major advance over anon.penet.fi, they fell short of the anonymity provided by the ideal mix. In 1995, Lance Cottrell outlined some of the problems with "Type I" remailers [35]:

- **Traffic Analysis:** Cypherpunk remailers tend to send messages as soon as they arrive, or after some specified amount of delay. The first option makes it easy for an adversary to correlate messages across the mix-net. It's not clear how much delay helps protect against this attack.

- **Does Not Hide Length:** The length of messages is not hidden by the encryption used by cypherpunk remailers [2]. This allows an adversary to track a message as it passes through the mixnet by looking for messages of approximately the same length.

Cottrell wrote the Mixmaster, or "Type II", remailer to address these problems. Instead of using PGP, Mixmaster uses its own client software (which is also the server software), which understands a special Mixmaster packet format. All Mixmaster packets are the same length. Every message is

[2] note that the definitions of semantic security and non-malleability do not seem to imply "length-hiding" either

encrypted with a separate 3DES key for each mix node in a chain between the sender and receiver; these 3DES keys are in turn encrypted with the RSA public keys of each mix node.

When a message reaches a mix node, it decrypts the header, decrypts the body of the message, and then places the message in a "message pool." Once enough messages have been placed in the pool, the node picks a random message to forward.

As of this writing, Mixmaster is in version 2.9b22[67]. Discussion of the project can be found on the mix-l mailing list[66]. A Mixmaster version 3 is planned in which nodes will communicate with each other via TCP/IP connections. All traffic will be encrypted with a key derived by a Diffie-Hellman key exchange and then destroyed immediately after the transaction is ended, thereby providing perfect forward secrecy. Unfortunately, the prototype specification for this protocol is only available in German and is not finished.

B.3.4 Nymservers and nym.alias.net

The reply blocks used by cypherpunks remailers are important for providing for return traffic, but they must be sent to every correspondent individually. In addition, using a reply block requires that a correspondent be familiar with the use of specialized software. This problem is addressed by *nymservers*, which act as holding and processing centers for reply blocks.

To use a nymserver, a user simply registers an e-mail address of the form "nym@nymserver.net" and associates a reply block with it. This association can be carried out via anonymous e-mail. Then whenever a message is sent to "nym@nymserver.net," the nymserver automatically prepends the associated reply block, encrypts the aggregate, and sends it off to the appropriate anonymous remailer.

The most popular nymserver may be the one run at nym.alias.net, which is hosted at MIT's Lab for Computer Science. A recent report by Mazieres and Kaashoek details the technical and social details of running the nymserver, including problems of abuse[62].

B.3.5 Remailer User Interfaces

The major reason for the massive popularity of `anon.penet.fi` was that it was extremely easy to use. Anyone who could type "Request-Remailing-To:" at the top of an e-mail message could send anonymous e-mail. With the advent of remailers which required the use of PGP or the Mixmaster software, the difficulty of using remailers increased. This difficulty was aggravated by the fact that for years, both PGP and Mixmaster were only available as command-line applications with a bewildering array of options.

B.4 Recent Mix-Net Designs

B.4.1 TAZ / Rewebber

Goldberg and Wagner applied Mixes to the task of designing an anonymous publishing network called Rewebber[38]. Rewebber uses URLs which contain the name of a Rewebber server and a packet of encrypted information. When typed into a web browser, the URL sends the browser to the Rewebber server, whch decrypts the associated packet to find the address of either another Rewebber server or a legitimate web site. In this way, web sites can publish content without revealing their location.

Mapping between intelligible names and Rewebber URLs is performed by a name server called the Temporary Autonomous Zone(TAZ), named after a novel by Hakim Bey. The point of the "Temporary" in the name of the nameserver (and the novel) is that static structures are vulnerable to attack. Continually refreshing the Rewebber URL makes it harder for an adversary to gain information about the server to which it refers.

B.4.2 Babel

Contemporary with Cotrell's Mixmaster is an effort by Gulcu and Tsudik called "Babel"[41]. Babel uses a modified version of PGP as its underlying encryption engine. This modified version does not include normal headers, which would include the identity of the receiver, the PGP version number, and other identifying information.

The Babel paper defines quantities called the "guess factor" and the "mix factor" which model the ability of an adversary to match messages passing through the mix with their original senders. Then several attacks are presented, including the trickle and flooding attack, along with some countermeasures. The paper is noteworthy in that it attempts to give an analysis of just how much the practice of batching messages helps the untraceability of a mix-net node.

B.4.3 Stop and Go Mixes

The next step in probabilistic analysis for mixnets comes in the work of Kesdogan, Egner, and Buschkes [53], who proposed the "Stop and Go Mix." They divide networks into two kinds: "closed" networks, in which the number of users is small, known in advance, and all users can be made distinct, and "open" networks like the Internet with extremely large numbers of users. They claim that perfect anonymity cannot be achieved in these open networks, because there is no guarantee that every single client of the mix node is not the same person coming under different names.

Instead, they define and consider a notion of *probabilistic anonymity*: given that the adversary controls some percentage of the clients, some other set of mix servers, and is watching a Mix, can the probability of correlating messages be quantified in terms of some security parameter? They consider queueing theory as an inspiration for a statistical model and manage to prove theorems about the adversary's knowledge in this model.

B.4.4 Variable Implicit Addresses

Later, Kesdogan et. al. applied Mixes to the GSM mobile telephone setting[52]. Here, the point is to allow for GSM roaming from cell to cell while still protecting the user's real location from discovery by the phone company or an outside intruder. This is done by the use of *variable implicit addresses*, which work as follows : each roaming area has a publically known and static explicit address. When the client GSM phone comes online or crosses the boundaries of a cell, it queries the surrounding cells and downloads these addresses. Then it creates a new address for itself which combines the addresses of its surrounding cells.

Then, instead of sending the entirety of the new address, the phone sends only some characters, say $log n$, of the address to the network to identify itself. The network then directs traffic intended for the phone to any cell which has those $log n$ characters in its address. A refinement process then takes place in which the phone gives out slightly more information to the system to improve performance by sending information to fewer cells, but not so much as to allow its location to be restricted to only one cell.

B.4.5 Jakobsson's Practical Mix

At EUROCRYPT '98, Jakobsson proposed a mixnet which was both practical and could be proved to mix correctly as long as less than 1/2 of the servers were corrupted[48]. The crucial idea is to treat the mixing as a secure multiparty computation in which each party is collaborating to make the collective mix look like a "random enough" permutation on a batch of messages. Then techniques of zero-knowledge proof are used by which each server can prove to all other servers that they are in fact conforming to the mix protocol. Deviating servers cannot produce valid proofs, and so can be caught and excluded from future mixing. Jakobsson's original protocol requires in the neighborhood of 160 modular exponentiations per message per server.

At PODC '99, Jakobsson showed how the use of precomputation could reduce the cost even further[47]. This new "flash mix" required only around 160 modular *multiplications* per message per server. This level of efficiency makes flash mixing competitive with the encryption used in anonymous remailers, and a serious candidate for low-latency mixing.

B.4.6 Universally Verifiable Mix-nets

With Jakobsson's design, the correctness of a mix-net can only be verified by the mix servers themselves. When more than a threshold of servers is corrupt, the verification fails. Because a user of the mix-net may not be aware of the corruption, this failure may be silent and therefore dangerous. One solution to this problem is a *universally verifiable* mix-net – a mix-net whose correctness can be verified by anyone, regardless of their status as server or user.

The concept was introduced by Killian [88], and recently a design of this type was proposed at EUROCRYPT '98 by Abe [2]. This design works along the similar broad lines as the Jakobsson design; each mix server uses zero-knowledge proofs to prove that it is acting in accordance with some protocol to randomly mix messages. The difference here is that these proofs are posted publically by the mix nodes instead of being multicast only to other mix nodes. The novel feature of Abe's design is that the work necessary to verify these proofs grows in a fashion independent of the number of servers. Unfortunately, verifying these proofs requires on the order of 1600 modular exponentiations per message.

B.4.7 Onion Routing

The Onion Routing system designed by Syverson, et. al. creates a mix-net for TCP/IP connections [95, 77]. In the Onion Routing system, a mixnet packet, or "onion", is created by successively encrypting a packet with the public keys of several mix servers, or "onion routers."

When a user places a message into the system, an "onion proxy" determines a route through the anonymous network and onion encrypts the message accordingly. Each onion router which receives the message peels the topmost layer, as normal, then adds some key seed material to be used to generate keys for the anonymous communication. As usual, the changing nature of the onion – the "peeling" process – stops message coding attacks. Onions are numbered and have expire times, to stop replay attacks. Onion routers maintain network topology by communicating with neighbors, using this information to initially build routes when messages are funneled into the system. By this process, routers also establish shared DES keys for link encryption.

The routing is performed on the application layer of onion proxies, the path between proxies dependent upon the underlying IP network. Therefore, this type of system is comparable to loose source routing.

Onion Routing is mainly used for sender-anonymous communications with non-anonymous receivers. Users may wish to Web browse, send email, or use applications such as `rlogin`. In most of these real-time applications, the user supplies the destination hostname/port or IP address/port. Therefore, this system only provides receiver-anonymity from a third-party, not from the sender.

Furthermore, Onion Routing makes no attempt to stop timing attacks using traffic analysis at the network endpoints. They assume that the routing infrastructure is uniformly busy, thus making passive intra-network timing difficult. However, the network might not be statistically uniformly busy, and attackers can tell if two parties are communicating via increased traffic at their respective endpoints. This endpoint-linkable timing attack remains a difficulty for all low-latency networks.

B.4.8 Zero Knowledge Systems

Recently, the Canadian company Zero Knowledge Systems has begun the process of building the first mix-net operated for profit, known as Freedom [102]. They have deployed two major systems, one for e-mail and another for TCP/IP. The e-mail system is broadly similar to Mixmaster, and the TCP/IP system similar to Onion Routing.

ZKS's "Freedom 1.0" application is designed to allow users to use a nym to anonymously access web pages, use IRC, etc. The anonymity comes from two aspects: first of all, ZKS maintains what it calls the Freedom Network, which is a series of nodes which route traffic amongst themselves in order to hide the origin and destination of packets, using the normal layered encryption mixnet mechanism. All packets are of the same size. The second aspect of anonymity comes from the fact that clients purchase "tokens" from ZKS, and exchange these token for nyms – supposedly even ZKS isn't able to correlate identities with their use of their nyms.

The Freedom Network looks like it does a good job of actually demonstrating an anonymous mixnet that functions in real-time. The system differs from Onion Routing in several ways.

First of all, the system maintains Network Information Query and Status Servers, which are databases which provide network topology, status, and ratings information. Nodes also query the key servers every hour to maintain fresh public keys for other nodes, then undergo authenticated Diffie-Hellman key exchange to allow link encryption. This system differs from online inter-node querying that occurs with Onion Routing. Combined with centralized nym servers, time synchronization, and key update/query servers, the Freedom Network is not fully decentralized [37].

Second, the system does not assume uniform traffic distribution, but instead uses a basic "heart-beat" function that limits the amount of inter-node communication. Link padding, cover traffic, and a more robust traffic-shaping algorithm have been planned and discussed, but are currently disabled due to engineering difficulty and load on the servers. ZKS recognizes that statistical traffic analysis is possible [91].

Third, Freedom loses anonymity for the primary reason that it is a commercial network operated for profit. Users must purchase the nyms used in pseudonymous communications. Purchasing is performed out-of-band via an online Web store, through credit-card or cash payments. ZKS uses a protocol of issuing serial numbers, which are reclaimed for nym tokens, which in turn are used to anonymously purchase nyms. However, this system relies on "trusted third party" security: the user must trust that ZKS is not logging IP information or recording serial-token exchanges that would allow them to correlate nyms to users [89].

B.4.9 Web Mixes

Another more recent effort to apply a Mix network to web browsing is due to Federrath et. al.[16] who call their system, appropriately enough, "Web Mixes." From Chaum's mix model, similar to other real-time systems, they use: layered public-key encryption, prevention of replay, constant message length within a certain time period, and reordering outgoing messages.

The Web Mixes system incorporates several new concepts. First, they use an adaptive "chop-and-slice" algorithm that adjusts the length used for all messages between time periods according to the amount of network traffic. Second, dummy messages are sent from user clients as long as the clients are connected to the Mix network. This cover traffic makes it harder for an adversary to perform traffic analysis and determine when a user sends an anonymous message, although the adversary can still tell when a client is connected to the mixnet. Third, Web Mixes attempt to restrict insider and outsider flooding attacks by limited either available bandwidth or the number of used time slices for each user. To do this, users are issued a set number of blind signature tickets for each time slice, which are spent to send anonymous messages. Lastly, this effort includes an attempt to build a statistical model which characterizes the knowledge of an adversary attempting to perform traffic analysis.

B.5 Other Anonymous Channels

B.5.1 The Dining Cryptographers

The Dining Cryptographers protocol was introduced by David Chaum[21] and later improved by Pfitzmann and Waidner[] as a means of guaranteeing untraceability for the sender and receiver of a message, even against a computationally all-powerful adversary. The protocol converts any broadcast channel into an anonymous broadcast channel. In the context of Free Haven, however, we have a problem : the participants in the protocol are identified, even though the sender and receiver of any given message is not. If the only long-term participants in the protocol are likely to be Free Haven servnet nodes, then we do not achieve the server-anonymity we desire. Less serious, but still important, problems are the efficiency of the protocol and the difficulty of correct implementation.

Therefore we have not seriously considered using the dining cryptographers protocol to provide Free Haven's anonymous channel. If we were to do so, we might consider running a dining cryptographer protocol using Mixes to hide the legal identity of each participant. In that case, while a failure of the Mix would reveal a participant's identity, the anonymous broadcast would prevent him or her from being linked to any particular message.

B.5.2 Crowds

The Crowds system was proposed and implemented by AT&T Research, named for collections of users that are used to achieve partial anonymity for Web browsing [85]. A user initially joins some crowd and her system begins acting as a node, or anonymous *jondo*, within that crowd. In order to instantiate communications, the user creates some path through the crowd by a random-walk of *jondos*, in which each *jondo* has some small probability of sending the actual `http` request to the end server. Once established, this path remains static as long as the user remains a member of that crowd. The Crowds system does not use dynamic path creation so that colluding crowd eavesdroppers are not able to probabilistically determine the initiator (i.e., the actual sender) of requests, given repeated requests through a crowd. The *jondos* in a given path also share a secret *path key*, such that local listeners, not part of the path, only see an encrypted end server address until the request is finally sent off. The Crowds system also includes some optimizations to handle timing attacks against repeated requests, as certain HTML tags cause browsers to automatically issue re-requests.

Similar to other real-time anonymous communication channels (Onion Routing, the Freedom Network, Web Mixes), Crowds is used for senders to communicate with a known destination. The system attempts to achieve sender-anonymity from the receiver and a third-party adversary. Receiver-anonymity is only meant to be kept from adversaries, not from the sender herself.

The Crowds system serves primarily to achieve sender and receiver anonymity from an attacker, not provide unlinkability between the two agents. Due to high availibility of data – real-time access is faster that mix-nets as Crowds does not use public key encryption – an adversary can more easily use traffic analysis or timing attacks. However, Crowds differs from all other systems we have discussed, as users are *members* of the communications channel, rather than merely communicating *through* it. Sender-anonymity is still lost to a local eavesdropper that can observe all communications to and from a node. However, other colluding *jondos* along the sender's path – even the first-hop – cannot expose the sender as originated the message. Reiter and Rubin show that as the number of crowd members goes to infinity, the probable innocence of the last-hop being the sender approaches one.

B.5.3 Ostrovsky's Anonymous Broadcast via XOR-Trees

In CRYPTO '97, Ostrovsky considered a slightly different model of anonymous broadcast[]. In this model, there are n servers broadcasting into a shared broadcast channel. One of the servers is a special "Command and Control" server; the rest are broadcasting dummy traffic. Then there is an adversary who has control of some of the servers and wants to know which server is the

“Command and Control.” Ostrovsky shows how to use correlated pseudo-random number generators whose output reveals a certain message when XORed together to create a protocol which prevents the adversary from discovering which server is the correct one, even if he can eavesdrop on all communications and corrupt up to k servers, where k is a security parameter which affects the efficiency of the protocol.

Bibliography

[1] M. Abdalla, M. Bellare, and P. Rogaway. DHAES. Submission to IEEE P1363.

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

[3] Abrams v. U.S. , 250 u.s. 616, 1919. http://caselaw.findlaw.com/scripts/getcase.pl?navby=search&court=US&case=/us/250/616.html.

[4] ACLU of Georgia v. Miller, 1997. http://www.aclu.org/court/aclugavmiller.html.

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

[6] Anonymity on the internet FAQ. http://www.eff.org/pub/Privacy/Anonymity/net_anonymity.faq.

[7] The Anonymizer. http://www.anonymizer.com.

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

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

[10] John Perry Barlow. The Economy of Ideas. http://www-swiss.ai.mit.edu/6805/articles/int-prop/barlow-economy-of-ideas.html.

[11] Geremie R. Barme and Sang Ye. The Great Firewall of China. http://www.wired.com/wired/5.06/china.html.

[12] Douglas Barnes. The coming jurisdictional swamp of global internetworking (or, how i learned to stop worrying and love anonymity. http://www.communities.com/paper/swamp.html.

[13] Tonda Benes. The Eternity Service. http://www.kolej.mff.cuni.cz/~eternity/.

[14] Tonda Benes. The Eternity Service. http://www.kolej.mff.cuni.cz/~eternity/Doc/TondaBenes/Thesis/ps/thesis.ps.

[15] Oliver Berthold, Hannes Federrath, and Marit Kohntopp. Project "anonymity and unobservability in the internet.". In *Workshop on Freedom and Privacy by Design / CFP2000.*

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

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

[18] Patricia Brennan. Timeline of copyright law in the United States. http://arl.cni.org/info/frn/copy/timeline.html.

[19] Anita Susan Brenner and B. Metson. Paul and Karla hit the net. http://www.cs.indiana.edu/canada/wired.

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

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

[22] 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.

[23] American Civil Liberties Union. ACLU wins first-ever challenge to a state internet censorship law in Georgia. http://www.aclu.org/news/n062097b.html.

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

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

[26] Cubby v. Compuserve, 1991. http://www.loundy.com/CASES/Cubby_v_Compuserve.html.

[27] John Curran and Leslie Daigle. Uniform Resource Names (urn) Charter. http://www.ietf.org/html.charters/urn-charter.html.

[28] TrakWeb Development. Welcome to MobilTrak. http://www.mobiltrak.com/.

[29] Matt Dietrich. Fcc ruling won't affect low-power radio pioneer. http://www.infoshop.org/news5/kantako.html, January 2000.

[30] EFF. Canadian law supporting right to anonymity. http://www.eff.org/pub/Privacy/Anonymity/can_anonymity.law.

[31] Elrod v. Burns, 427 U.S. 347, 373, 1976.

[32] Dave Beazley et al. Simplified wrapper and interface generator. http://www.swig.org/.

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

[34] Center for National Security Studies. The FBI's Domestic Counterterrorism Program. http://www.cdt.org/policy/terrorism/cnss.FBI.auth.html.

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

[36] Andrew V. Goldberg and Peter N. Yianilos. Towards and archival intermemory. In *Proc. IEEE International Forum on Research and Technology Advances in Digital Libraries (ADL'98)*, pages 147–156. IEEE Computer Society, April 1998.

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

[38] Ian Goldberg and David Wagner. Rewebber. *First Monday.*

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

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

[41] 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.

[42] Johan Helsingius. press release announcing closure of anon.penet.fi. http://www.penet.fi/press-english.html.

[43] David Hopwood. Definition of recipient-hiding cryptosystem. sci.crypt usenet post.

[44] Infoshop.org. How does capitalism affect liberty? http://www.infoshop.org/faq/secB4.html#secb41.

[45] Infoshop.org. Social struggle. http://www.infoshop.org/faq/secJ4.html#secj41.

[46] The Cato Institute. Anonymity on the Internet. http://www.cato.org/pubs/briefs/bp-054es.html.

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

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

[49] Thomas Jefferson. Declaration of Independence, as originally drafted, 1776.

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

[51] Gene Kan. Gnutella protocol. http://capnbry.dyndns.org/gnutella/protocol.html.

[52] D. Kesdogan, A. Trofimov, and D. Trossen. Minimizing the average cost of paging on the air interface. In *KuVS Springer-Verlag*, 1999.

[53] Dogan Kesdogan and ... Stop and go mixes : Providing probabilistic anonymity in an open system. In *1998 Information Hiding Workshop.*

[54] Kolender v. Lawson, 461 U.S. 352, 357, 1983.

[55] Jason Kroll. Crackers and crackdowns. http://www2.linuxjournal.com/articles/culture/007.html.

[56] Lamont v. Postmaster General, 381 U.S. 301, 1965. http://caselaw.findlaw.com/scripts/getcase.pl?court=US&vol=381&invol=301.

[57] Jon Lebkowsky. Interview with Eric Hughes in the Electronic Frontiers forum. http://hotwired.lycos.com/talk/club/special/transcripts/96-07-11.hughes.html.

[58] Brian Livingston. AOL's 'youth filters' protect kids from Democrats. http://www.news.com/Perspectives/Column/0,176,421,00.html.

[59] Lucent personalised web assistant. http://www.lpwa.com.

[60] Tal Malkin. *Private Information Retrieval.* PhD thesis, MIT. see http://www.toc.lcs.mit.edu/ tal/.

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

[62] 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.

[63] Mcintyre v. Ohio Elections commission, 1995. http://cpsr.org/cpsr/free_speech/mcintyre.txt.

[64] Steve Miale. Paul Teale/Karla Homolka information site. http://www.cs.indiana.edu/canada/karla.html.

[65] Stanley Milgram. The Small World Problem. *Psychology Today*, 2:60–67, 1967.

[66] Mix-l mixmaster discussion list. mix-l-subscribe@egroups.com.

[67] Ulf Möller and Company. Mixmaster 2.9b source code. http://mixmaster.anonymizer.com.

[68] David Molnar. Free Haven: Some possible weaknesses? http://freehaven.net/archives/freehaven/dev/Feb-2000/msg00001.html.

[69] Andrew Musgrave. Liability of internet service providers for copyright infringement the impact of the copyright amendment (digital agenda) bill 1999. http://www.lawnow.com.au/LegalRegArticles/authors/musgrave/musgravetxt.htm.

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

[71] No Electronic Theft act. ftp://ftp.loc.gov/pub/thomas/c105/h2265.ih.txt.

[72] New York Times Co. v. Sullivan, 376 U.S. 254, 1964.

[73] A bad gag order in Canada. http://www.cs.indiana.edu/canada/n.y.times-editorial.

[74] British House of Commons. Regulation of investigatory powers bill. http://www.parliament.the-stationery-office.co.uk/pa/cm199900/cmbills/064/2000064.htm.

[75] U.S. Library of Congress. About the Federalist Papers. http://lcweb2.loc.gov/const/fed/abt_fedpapers.html.

[76] 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.

[77] Onion router. http://www.onion-router.net/.

[78] DeCSS lawsuit overview. http://www.lemuria.org/DeCSS/cca.html.

[79] Thomas Paine. Common sense. http://libertyonline.hypermall.com/Paine/CS-Body.html.

[80] 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.

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

[82] Associated Press. German court: AOL liable for music piracy. http://www.usatoday.com/life/cyber/tech/review/crh053.htm.

[83] The proxomitron. http://www.proxomitron.cjb.net/.

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

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

[86] Karina Rigby. Anonymity on the internet must be protected. http://www-swiss.ai.mit.edu/6095/student-papers/fall95-papers/rigby-anonymity.html.

[87] R. Rivest. Graduated mirroring. http://freehaven.net/archives/freehaven/dev/Jan-2000/msg00001.html.

[88] K. Sako and J. Killian. Receipt-free mix-type voting scheme. In *Advances in Cryptology – EUROCRYPT '95*, pages 393–403.

[89] Russell Samuels. Untraceable nym creation on the freedom network, November 1999.

[90] Bruce Schneier and John Kelsey. The street performer protocol. http://www.counterpane.com/street_performer.html.

[91] Adam Shostack and Ian Goldberg. Freedom 1.0 security issues and analysis, November 1999.

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

[93] Stratton v. Prodigy, 1995. http://www.epic.org/free_speech/stratton_v_prodigy_1995.txt.

[94] Anne Swardson. Unspeakable crimes: This story can't be told in Canada. and so all Canada is talking about it... http://www.cs.indiana.edu/canada/WashingtonPost.

[95] 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.

[96] Talley v. California, 362 U.S. 60, 64, 1960.

[97] United States v. Aluminum Company of America. 148 F. 2d 416, 1945.

[98] 22 U.S.C. 611 (j).

[99] 39 U.S.C. 4008 (b).

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

[101] Duncan J. Watts and Steven H Strogatz. Collective dynamics of 'small-world' networks. *Nature*, page 393, 1998.

[102] Zero Knowledge Systems. http://www.freedom.net/.