Dagster: Censorship-Resistant Publishing Without Replication

Adam Stubblefield
Dan S. Wallach
Department of Computer Science, Rice University

Abstract

In this paper we present Dagster, a new censorship-resistant publishing scheme. Unlike previous censorship-resistant schemes, Dagster does not rely on the widespread replication of data and can even be used in a single server setting. It accomplishes this by "intertwining" legitimate and illegitimate data, so that a censor can not remove objectionable content without simultaneously removing legally protected content. The Dagster system was designed to be as simple and efficient as possible. It increases required network traffic by a contant (but tunable) factor, but otherwise has a very low cost for both clients and servers, making it easy to scale.

1 Introduction

In recent years, censorship has become an issue with online publication. The Church of Scientology has attempted to stop open publication of its "secret" religious materials. The Motion Picture Association of America and others have attempted to stop publication of DeCSS, a small program that can defeat the encryption technique used on commercial DVD movies. Censorship can occur in many forms, whether it be a government or powerful corporation that objects to any particular online document. With some traditional systems, such as Usenet news postings, anyone who sees a message can post a "cancel" message to delete it. With other systems such as Web servers, the owner of the Web server can be coerced, often with some kind of legal threat, to delete an objectionable document.

Of course, any document worthy of censorship by one party is often quite interesting to another party. Furthermore, what may be legal to censor in one country may not be legal to censor in another. A complete legal analysis of what may or may not be legally censorable is beyond the scope of this paper, but we assert that there exist documents worthy of publication that deserve strong protection against all forms of censorship.

Current censorship-resistant techniques gain this property by way of widely replicating their data. If a document is copied to servers in every country across the globe, surely censorship of the document must be beyond the laws of *some* of the countries. However, with global treaties on intellectual property, such a model could well be legislated out of existence.

Instead of taking advantage of geographic distribution for censorship-resistance, we would prefer to take advantage of free-speech rights within a specific country. If we assume that most documents are censorship-proof, then we would like our system to *intertwine* all of its documents together in such a fashion that deleting any one document would cause other documents to become unreadable. This creates a legal conundrum: any action to delete a censorable document would, with high probability, have the effect of deleting an unknown number of legitimate documents. In a legal system where freedom of speech for a legitimate document has a higher value than a right of censorship on an illegitimate document, our system is able to protect the illegitimate document using the legitimate one. Furthermore, we can achieve this property without requiring a geographically distributed system. The whole system can be implemented on a single server and preserve the same anti-censorship guarantees. Replication can still be added, of course, to increase the system's reliability and performance, but it's unnecessary for anti-censorship reasons.

In order to intertwine documents together, we present a model where publishing a document requires the publisher to first download a number of pre-existing documents and then to mix them with the document to be published. In this way, new documents *depend* on older documents in order to be later decoded. Since the graph of these dependencies is a directed acyclic graph, we refer to our system as "Dagster."

The rest of this paper is organized as follows: Section 2 discusses related work, Section 3 contains our goals and an outline of the operation of the system, Section 4 provides a detailed look at the Dagster protocols, Section 5 explores the properties these protocols provide, Section 6 discusses how Dagster could work in a distributed system, Section 7 presents future work, and Section 8 concludes.

2 Related Work

In this section we will outline how Dagster relates to a number of previous Internet anonymity schemes. Some of these schemes, like Dagster, aim to provide anonymous and censorship-resistant publishing, while others provide the connection anonymity that is used by Dagster to provide publisher anonymity. We will also outline the ideas that we borrow from the cryptographic literature.

2.1 Anonymous and Censorship Resistant Publishing

The first major work on anonymous Internet publishing was Ross Anderson's Eternity Service [1]. This paper provides the vision for a network of distributed data havens that are highly censorship resistant. Using electronic cash, a user would be able to submit some data to be published, at which point it would be replicated to servers all over the world. To retrieve data, a request would be broadcast to all of the servers in the network, and relayed to the requester through an anonymizing remailer. There have since been a number of interesting implementations based on this basic model [2, 3, 7]. Unlike Dagster, these systems provide censorship-resistance by making it logistically difficult for a censor to remove a document from all of the servers that are hosting it.

A more recent censorship-resistant publishing system is Publius [10]. Encrypted content is stored on a subset of the Publius servers. Then, Shamir secret sharing [14] is used to spread the key across multiple servers. A retriever downloads the encrypted document from one server and the key shares from multiple servers and can thus reconstruct the original document. Publius also allows for the original content publisher to update and delete data from the system. The biggest problem with Publius is it's scalability: the list of Publius servers is fixed, so it is quite problematic to add new servers to the system or to remove a corrupt server.

Gnutella[1] is a peer-to-peer system whose main aim to provide high availability of data. The protocol does attempt to provide a degree of anonymity, though for an actual document transfer to take place the requester and provider must know each other's IP addresses, since document delivery is point to point. Also, Gnutella is non-persistent; when a user logs out of the network, any data he was sharing disappears as well.

Another peer-to-peer system is Freenet [5], which, unlike Gnutella, replicates popular data across multiple hosts with the goal of providing high availability. Freenet does a somewhat better job of providing anonymity than Gnutella, as the true requester's IP address is not required to propagate all the way across the network. It's replication and caching mechanisms also provide much higher availability of data.

The Rewebber [6] is a URL rewriting service. It acts as an intermediary between a content author and requester, so that neither can obtain the identity of the other. Goldberg and Wagner have extended the idea to include a network of

[1] `http://gnutella.wego.com`

rewebbers [9].

The Tangler [17] system, which is also based in the idea of intertwining data, provides essentially the same basic services as Dagster. Besides a simple block storage scheme, the Tangler paper also provides an elegant method of embedding metadata, which makes the system function much like a UNIX filesystem. We note that much of this metadata discussion could be directly applied to Dagster blocks. Unlike Dagster, Tangler's intertwining scheme, based on polynomial interpolation, seems quite expensive and does not scale well when linking to an arbitrary number of pre-existing blocks.

2.2 Anonymous Connections

One of the simplest anonymous connection providers is the Anonymizer[2]. Instead of requesting a web page directly, a user sends the request to the Anonymizer which forwards the request appropriately. The content is then delivered to the Anonymizer which returns it to the requesting user. The Anonymizer can only be used to retrieve web content and the user is required to trust the Anonymizer's operators not to reveal her identity.

To avoid the single point of trust issue, mix-nets [4] can be used. In a mix-net, data is routed through multiple hosts, none which are able to determine both it's true source or destination. Onion Routing [16] and Crowds [13] are mix-net style systems for web browsing. The Freedom Network [15] is a mix-net at the IP level.

2.3 Cryptographic Primitives

A good summary of cryptographic hash functions can be found in chapter 9 of Menezes, *et al.* [11]. The idea of hash trees is due to Merkle [12].

3 Overview

We will now provide our goals for Dagster, as well as an overview of how the system works.

3.1 Goals

Before examining how Dagster works, we first present our design goals.

1. **Censorship Resistant** It should be difficult for a censor to claim that any given block of data that a server is storing is "censorable". That is, it should

[2] `http://www.anonymizer.com`

be impossible to censor a document without, in the process, censoring a large number of legitimate documents.

2. **Publisher and Retriever Anonymous** There should be no way to link the identity of the publisher or retriever of a block of data to the data itself.

3. **Server Deniable** The published data that a server stores should be indistinguishable from random data.

4. **Tamper Evident** The system should allow users to determine if a server has tampered with published data.

5. **Server Independent** The system should not be tied to a particular networking architecture or protocol.

This set of goals is built on the idea that a legal system wherein only certain information can be censored exists in the jurisdiction (or jurisdictions) that govern the Dagster service. This corresponds roughly to the current legal systems in many western countries. Since only illegitimate data can be censored in this model, the Dagster system tries to completely "intertwine" legitimate and illegitimate data. In this way, the act of removing a piece of illegitimate data will cause legitimate data to be removed as well, something that the law should not allow. A true legal analysis of the implications of Dagster is beyond the scope of this paper.

3.2 Architecture

The actual Dagster system is split into three parts: an anonymous channel between publishers or requesters and servers, an out-of-band channel for announcing that a document has been published, and the publishing server itself. The anonymous channel can be realized using the techniques discussed in section 2.2 of the related work. The out-of-band channel for announcing the publication of a document could be anything from an IRC channel to a newsgroup to a mailing list. The rest of this paper will concern the design of the actual publishing server.

At its heart, the Dagster server is a simple block storage mechanism. This paradigm was chosen as it can be easily applied to everything from a single server to a highly distributed system. Besides being able to store and produce blocks, the requirements made on Dagster servers are actually quite minimal. The decision to relegate most of the complexity to the clients was made so that Dagster would be able to scale easily using almost any underlying distributed system architecture.

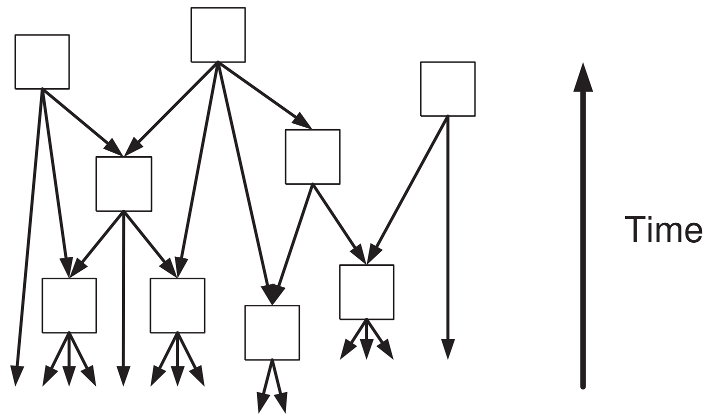

Figure 1: As time goes forward, current blocks in the systems are linked to by new blocks, forming a DAG structure.

3.3 Publishing and Retrieving a Document

Publishing takes place in three steps: Block Selection, Block Committal, and Document Announcement. In the Block Selection stage, the publisher first splits his document into a number of constant size blocks. He then randomly queries the Dagster system for a number of pre-existing blocks. The client then computes a new block, whose value is based on the previously stored blocks, the new message, and a key. In this way, the new data is "linked" to existing data as is shown in figure 1. Notice again how the structure of this linking forms a DAG.

During the Block Committal stage the client sends the new block to the server which stores it. The client then waits until he believes that a legitimate block has chosen to "link" with the block that he has committed. At that point, he would disclose the instructions needed to construct the original block.

To retrieve a document, a client simply asks the system for the required blocks and is able to retrieve the original message.

This process is covered in more depth in section 4.

4 Protocols

This section describes in detail the various protocols used by the Dagster system and how they, together, satisfy each of the goals listed in section 3.1.

4.1 The Basics

Each Dagster server stores a number of fixed length b-bit blocks, each of which can be referenced by a value know as their *block ID* (this is described below). All of these blocks should resemble random data as closely as possible.

4.2 Randomness of Data

Many anonymous publishing systems, most notably Publius [10] and Freenet [5], claim that the server is able to deny that it knows what content it is storing. Publius accomplishes this by storing encrypted content and only a share of the key required to decrypt the content. In Freenet, content is encrypted by the hash of its description.

The problem with both of these systems is that it is the *client* who is assumed to have done the encryption. There is no sanity check in place to catch a malicious client who could post a plaintext message to either of these services. Besides no longer protecting the content, the server operator might also be vulnerable for "knowingly" distributing illegal content.

For this reason, we require both Dagster clients and servers to implement the predicate $\mathsf{Random?}(x)$, which returns true if the block is "random" and false otherwise. There are many possible ways to implement this test. One possibility is through a series of ad hoc statistical tests, such as the ones described in chapter 5 of Menezes, *et al.* [11]. A second possibility is through a non-interactive zero-knowledge proof of encryption. The construction of such a proof is beyond the scope of this paper.

This predicate will allow both clients and servers to decide for themselves whether the other parties in the system are following the rules.

4.3 Block IDs

In Dagster, each block of data stored on a server is referred to by the cryptographically secure hash of its contents. This value is called the *block ID*. It should be noted that so long as the hash function is second-preimage resistant, it is infeasible for an adversary to create two blocks which have the same block id.

This system of identifying blocks avoids many of the pitfalls encountered using the naming schemes of other anonymous publishing systems. For example, Publius [10] is especially vulnerable to targeted denial of service attacks. If an attacker has a copy of some data that he would like to stay out of the Publius servers, he can insert bogus data under the same "name" that the legitimate data would have used[3]. This attack is not possible in Dagster, since the block itself contains all the information needed to compute its ID. Again, this prevents malicious clients from cheating the system.

We decided not to implement a Freenet style block description scheme because, based on our experience, the mode in which Freenet is actually used involves the posting of keys, rather than the blind guessing of descriptions. As an example, a recent document posted to Freenet was posted under at least three different names:

- `SDMI_Attack.html`
- `SDMI_attack-howto.html`
- `SDMI_attack-or-How_I_Learned_To_Piss_Hillary_Rosen_Off.html`

This small example also demonstrates that remembering the proper capitalization and punctuation for even the shortest name is non-trivial.

4.4 Enumeration of Blocks

The most complicated function that a Dagster server must provide is the enumeration of its blocks. This protocol is used at multiple steps during the publication process.

First, the server must sort all of the blocks that it is currently hosting in numerical order based on their block ID. The server then separates the block IDs into k-member groups and computes the hash of each group. These hashes are then separated into k-member groups and the hash of each of these groups is computed. This process continues until there is only 1 hash left, which forms the root of a hash tree whose leaves are the actual blocks. This is illustrated in figure 2.

The server must provide a method for retrieving the value k, the total number of blocks stored in the system, and the root of the hash tree. It must also provide a method whereby the client can ask for a specific portion of the tree.

4.5 Jump-starting the System

As the publication algorithm requires there to be at least one block in the system, we must first jump-start the system. This is done by having the server create approximately 1,000 random blocks, compute their block IDs, and add them to the system.

[3]For some types of data, such as text, whitespace could be added to create a document with the same meaning, but a different "name."

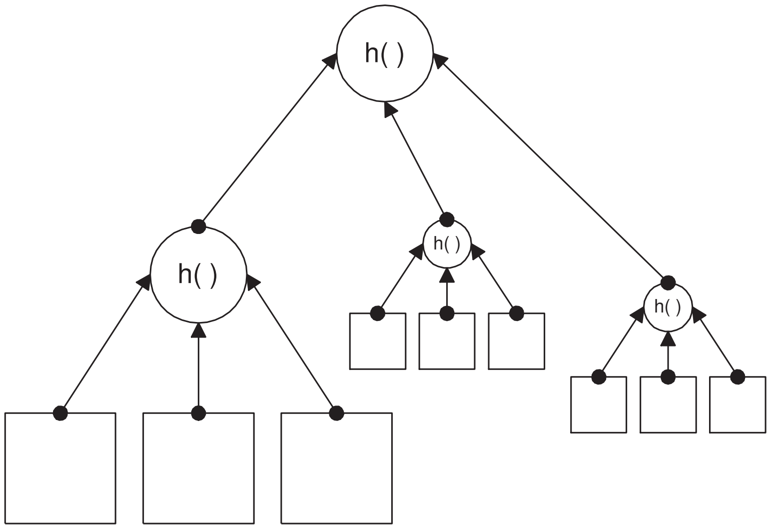

Figure 2: This shows the structure of the hash tree. At the leaves of the tree are the Dagster blocks, and at each intermediate node is a hash of the concatenation of the data at the previous level. The hash tree is organized such that there are a small constant number of leaves off of each node.

4.6 Publish

Publishing takes place in three steps:

1. **Block Selection** A publisher, Alice, wants to publish a document M. She first separates M into b-bit blocks, $M_1, \ldots, M_n$. Alice then queries the hash tree of the server, down random paths from the root down to a number of leaves. By being able to produce the blocks associated with the leaves, the server is able to convince Alice that it is actually storing all of the blocks indicated by the tree.

   Alice then randomly chooses c pre-existing blocks, $B_{1,i}, \ldots, B_{c,i}$ on the server for each M_i and computes the Random? predicate on each. If any of the blocks fail the predicate, Alice will stop because she would suspect the server was not following the Dagster protocol correctly.

2. **Block Committal** Now Alice randomly generates n keys, $k_1, \ldots, k_n$, and computes:
$$X_i = \mathcal{E}_{k_i}(M_i) \oplus \bigoplus_{j=1\ldots c} B_{j,i} \quad \text{for } i = 1, \ldots, n$$
   where $\mathcal{E}$ is a secure encryption function and $\oplus$ indicates exclusive or.

Alice then sends the n blocks, $x_1,\dots,x_n$, to the server over an anonymous channel. The server then computes the Random? predicate on each block and accepts the ones which pass. All blocks generated by the above equation should pass the server's randomness test, since each of the $B_{j,i}$'s passed and the output of the exclusive or operation has randomness equal to the maximal randomness of its inputs. The server then computes the block ID of each block and adds each to the system.

3. **Document Announcement** At periodic intervals after having committed the document to a server, Alice queries the hash tree of the server. She uses recursive search to ensure that her committed blocks are being stored and reported in the tree. Since the communication channel is anonymous, there is no way for the server to know that it is Alice who is querying for the block. Alice also keeps track of how many blocks are added to the tree. When she is satisfied that some later block is probably linking to hers, she will release the block IDs for the M_i's and their corresponding B's as well as the keys. This release is in the form of a Dagster Resource Locator, which is discussed in the next subsection.

4.7 Dagster Resource Locator (DRL)

A Dagster Resource Locator is the means by which Dagster "names" documents. It is simply a tuple of the form:

$$\left[\left(k_1,h(B_{1,1}),\dots,h(B_{c,1}),h(X_1)\right),\dots,\left(k_n,h(B_{1,n}),\dots,h(B_{c,n}),h(X_n)\right)\right]$$

where $h(\cdot)$ is the cryptographic hash function used to compute block IDs and π is a permutation of these hashes.

4.8 Document Retrieval

To retrieve a document after hearing it announced, a client simply asks the server for the blocks corresponding to the block IDs in the DRL. He first checks to make sure that the cryptographic hash of each of the blocks the server returned correspond to the hashes in the DRL. He can then compute:

$$M_i = \mathcal{D}_{k_i}\left(X_i \oplus \bigoplus_{j=1\dots c} B_{j,i}\right) \quad \text{for } i=1,\dots,n$$

where $\mathcal{D}$ represents decryption. The concatenation of the M_i's will result in the original M.

5 Analysis

In this section we will describe how the Dagster system achieves our stated goals.

5.1 Server Independence

The only operations that a server needs to be able to perform are:

1. Seeding the system with random values
2. Storing a block of data based on its block ID
3. Retrieving a block of data based on its block ID
4. The Random? predicate
5. A cryptographic hash function to compute the block IDs
6. The ability to build a hash tree and allow it to be queried by clients

In a single server setting, each of these operations are trivial to implement. We will discuss how they can work in a distributed system in section 6.

5.2 Tamper Evidence

The tamper evidence property is maintained locally to a particular document as a result of block IDs being directly computable from blocks. Therefore, it is simple for a client to verify that the server has indeed returned the correct blocks. The clients can not cheat the server, since the server will only store blocks under their correct block IDs.

Also, because the encryption used is plaintext aware, it is likely that the decryption step in the retrieval protocol will fail if the DRL has been subtly corrupted. This is more of a defense against typos or network errors than an attacker, since an attacker could simply replace an entire DRL with another valid DRL.

Globally, tamper evidence is provided by the anonymous channel and the hash tree. Since the server is required to commit to all of the blocks that it is current hosting without any knowledge of who the requester is, there is no way for a server to only offer certain blocks to certain clients.

5.3 Server Deniability

This property is provided by the Random? predicate. As long as we assume that the Random? predicate is a good one, the server is able to maintain that it was unable to distinguish the stored blocks that is has stored from random data.

5.4 Publisher and Retriever Anonymity

Because all connections between the server and retriever are over the anonymous channel, there is no correlation between their identities and the documents that they are publishing or requesting. Without anonymity, a number of different attacks on the system would be possible, most notably a man-in-the-middle attack.

5.5 Censorship Resistance

This property builds on many of the previous properties. In this section we do not take into account denial-of-service attacks or attacks in which an entire server is taken offline, reserving this discussion for section 6. Firstly, notice that as long as the client is following the protocol, no censorable data is revealed until the Document Announcement step. Even if a censor can guess the blocks that comprise a particular message, they are unable to guess the message, since the key has not been released.

Let's now consider one possibly censorable block. We must show that at the point that the Document Announcement step takes place, there is a chance that there is a valid message which also requires the block in question. Since the server has no way of knowing who is asking for a certain block (by the Anonymity property), the client who inserted a block is able to verify that the block in question is indeed being offered to others. The same client is also able to verify that the system is still accepting new blocks, either by querying the hash tree to verify that new blocks are being added, or by adding more blocks himself. Thus, as long as at least one client is following the protocol, there is a chance that every new block that the valid publisher adds depends on the block in question. It is interesting to note that because of the anonymity property, it is possible for a publisher to fill this role himself. That is, a publisher may publish a document that is known to be legally protected which links to the non-protected block.

If all parties in the system follow the protocol correctly, then the n-th block inserted into the system is expected to be linked by $\ell(n)$ other blocks where:

$$\ell(n) = \sum_{i=n+1}^{m} \left[1 - \prod_{j=1}^{c} \left(\frac{i-j}{i} \right) \right]$$

where m is the total number of blocks currently in the system and c is the number of blocks that each other block is linked to. For example, if $c = 10$, there is a 55 percent chance that the 10^7th block in the system is linked after 10^5 additional blocks are added.

Once any block is injected by a legitimate party, the censor is no longer able to say with complete certainty that a block is completely "bad."

6 Dagster in a Distributed System

Dagster is designed to work in a single server environment without replication; however, it can be scaled to use multiple servers with relative ease. Even though a DRL does not specify a location in which blocks are stored, it is completely unique in the sense that if a block is found which matches the block ID of an element of the DRL, it must be the correct block. For this reason, blocks can be replicated among servers without worrying about collisions.

Replication between servers can use any distributed block storage mechanism. To maintain the global Dagster properties, the only requirement is that the receiving server check for the randomness of the data and compute the data's block ID for itself.

Each server only needs to compute the hash tree with the data that it is currently storing. So long as a publisher is able to reconnect to the server to whom she originally published her document, she is still able to verify that the system is providing her document in its hash tree before disclosing the DRL.

The main advantage of implementing Dagster in a distributed system is increased availability of data. For example, an attacker might be able to use a denial of service attack to take down an entire Dagster server. By simply allowing servers to replicate data among themselves (using the protocol outlined above), Dagster would provide defenses against this class of attacks as well.

7 Future Work

We are planning to implement and deploy a Dagster based system. This will allow us to empirically determine suitable values for variables like the global system block size and the degree of linking to older blocks (higher numbers of links per document increases anti-censorship yet also increases bandwidth requirements). We also plan to explore the option of storing meta-data in blocks, which would allow the system to treat Dagster blocks as disk blocks in a filesystem; some blocks would store data while others would simply store pointers to data. This structure makes sense intuitively, as Dagster itself emulates a write-once hard drive.

Like most other censorship resistant systems, Dagster is vulnerable to a denial of service attack in which an attack simply sends lots of random looking blocks to the server which dutifully stores them. This can use up the server's storage space, preventing legitimate data from being stored. It would be possible to use either an anonymous digital cash (as is suggested in the Eternity service [1]) or "hash-cash" [8] based approach to defend against this attack.

8 Conclusions

We have presented Dagster, a censorship-resistant publishing system that does not rely on replication. It achieves this property through the "intertwining" of legitimate and illegitimate data. We have discussed the protocols through which data is added to and retrieved from the system, and shown how these protocols achieve the goals of the system. We have also sketched how Dagster could be applied to a distributed system.

Acknowledgments

We thank Marc Waldman for many useful comments that improved the presentation of this paper.

References

[1] ANDERSON, R. The Eternity Service. In *Proceedings of the 1st International Conference on the Theory and Applications of Cryptology (PRAGOCRYPT '96)* (Prague, Czech Republic, Oct. 1996), pp. 242–252.

[2] BACK, A. The Eternity Service, 1997. `http://www.cypherspace.org/adam/eternity/phrack.html`.

[3] BENES, T. The Eternity Service, 1998. `http://www.kolej.mff.cuni.cz/eternity/Doc/TondaBenes/Thesis/ps/thesis.ps`.

[4] CHAUM, D. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM 24*, 2 (Feb. 1981), 84–88.

[5] CLARKE, I., SANDBERG, O., WILEY, B., AND HONG, T. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability* (Berkeley, California, 2001).

[6] DEMUTH, T., AND RIEKE, A. On securing the anonymity of content providers in the world wide web. In *Proceedings of SPIE '99* (1999), vol. 3657, pp. 494–502. http://www.thomas-demuth.de/veroeffentlichungen/spie99.pdf.

[7] DINGLEDINE, R., FREEDMAN, M. J., AND MOLNAR, D. The free haven project: Distributed anonymous storage service. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability* (Berkeley, California, July 2000).

[8] DWORK, C., AND NAOR, M. Pricing via processing or combatting junk mail. In *Advances in Cryptology - Crypto '92* (Berlin, 1992), E. F. Brickell, Ed., Springer-Verlag, pp. 139–147. Lecture Notes in Computer Science Volume 740.

[9] GOLDBERG, I., AND WAGNER, D. Taz servers and the Rewebber Network: Enabling anonymous publishing on the World Wide Web. *First Monday 3*, 4 (1998).

[10] MARC WALDMAN, A. D. R., AND CRANOR, L. F. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proc. 9th USENIX Security Symposium* (Denver, Colorado, August 2000), pp. 59–72.

[11] MENEZES, A. J., VAN OORSCHOT, P. C., AND VANSTONE, S. A. *Handbook of Applied Cryptography*. CRC Press, New York, New York, 1997.

[12] MERKLE, R. C. Protocols for public key cryptosystems. In *1980 IEEE Symposium on Security and Privacy* (Oakland, California, 1980).

[13] REITER, M. K., AND RUBIN, A. D. Anonymous web transactions with Crowds. *Communications of the ACM 42*, 2 (Feb. 1999), 32–38.

[14] SHAMIR, A. How to share a secret. *Communications of the ACM 22*, 11 (Nov. 1979), 612–613.

[15] SHOSTACK, A., AND GOLDBERG, I. Freedom 1.0 security issues and analysis. Tech. rep., Zero-Knowledge Systems, Montreal, Canada, 1999. `http://www.freedom.net/info/freedompapers/Freedom-Security.pdf`.

[16] SYVERSON, P., GOLDSCHLAG, D., AND REED, M. Anonymous connections and onion routing. In *1997 IEEE Symposium on Security and Privacy* (Oakland, California, May 1997).

[17] WALDMAN, M., AND MAZIÈRES, D. Tangler - a censorship resistant publishing system based on document entanglements. In *Eighth ACM Conference on Computer and Communications Security* (Nov. 2001).