TAZ Servers and the Rewebber Network

Enabling Anonymous Publishing on the World Wide Web

Ian Goldberg and David Wagner
University of California, Berkeley
{iang,daw}@cs.berkeley.edu
CS 268 Final Project

May 16, 1997

Abstract

The World Wide Web has recently matured enough to provide everyday users with an extremely cheap publishing mechanism. However, the current WWW architecture makes it fundamentally difficult to provide content without identifying yourself. We examine the problem of anonymous publication on the WWW, propose a design suitable for practical deployment, and describe our implementation. Some key features of our design include universal accessibility by pre-existing clients, short persistent names, security against social, legal, and political pressure, protection against abuse, and good performance.

1 Introduction

Recently the Internet has seen tremendous growth, with the ranks of new users swelling at ever-increasing rates. This expansion has catapulted it from the realm of academic research towards new-found mainstream acceptance and increased social relevance for the everyday individual. Yet with this suddenly increased reliance on the Internet comes the threat that freedoms previously enjoyed may be overlooked in the online realm.

The World Wide Web has recently grown in importance to become perhaps the most important Internet application today. The WWW enables the ordinary citizen to reach an audience potentially as large as millions. In short, this technology gives us an extremely cheap publishing mechanism: printing presses for the masses.

Yet we must not forget that anonymous publishers have played an important role throughout the history of publication. Freedom of anonymous speech is an essential component of free speech, and freedom of speech is a critical part of any healthy democracy. For instance, the US Supreme Court has consistently upheld protection for anonymous publication of political speech. As Justice Stevens wrote in the McIntyre v. Ohio Election Commission majority opinion,

> Under our Constitution, anonymous pamphleteering is not a pernicious, fraudulent practice, but an honorable tradition of advocacy and of dissent. Anonymity is a shield from the tyranny of the majority.

We should be careful not to forget the importance of anonymous speech as we move to an increasingly WWW-centric paradigm of communication.

Today the WWW includes little support for anonymous publishing. The current WWW architecture fundamentally includes identity information in the URL that is used to locate published documents, and it is very hard for a WWW publisher to avoid revealing this information. To address these limitations, we would like to build technology to enable anonymous publishing on the World Wide Web.

2 Motivation

We briefly motivate this work with a few potential applications for anonymous WWW publishing. We hope these will help provide some of the general flavor for the requirements and illustrate the goals of such a publication service.

First, we note that anonymous publication is a time-honored tradition. The US Founding Fathers published the Federalist Papers (arguing for the adoption of the Constitution) anonymously under the pseudonym Publius. (Before the revolution, many had to publish secretly because of the very real danger of English prosecution.)

Across the world, from medieval times to the current age, governments have tried to suppress unwanted speech. Political dissidents have often turned to secret printing and distribution of anonymous leaflets, books, and other publications. Moreover, dissidents have frequently taken advantage of technology for expression of controversial ideas: 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"); and 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 [17, 23, 24].

Our vision combines many key aspects of these historically-successful approaches. First, we want to take advantage of the highly-favorable economics and vast potential audience available to WWW publishers. Second, we wish to use anonymity to avoid suppression of controversial ideas: when the targeted party cannot be identified, it is much harder to bring legal and other pressure on the author or printer. Third, we aim to gain resilience through the grass roots nature of samizdat channels. Fourth, because government bureaucracy often limits the speed with which full legal and other pressure can be brought to bear on targets to time periods measured in months or years, we wish to enable temporary stations to pop up and disappear just as rapidly. Through a combination of these techniques, we hope to build a robust infrastructure for anonymous WWW publishing.

3 Background

A few definitions are in order. Anonymity refers to the ability of an individual to prevent others from linking his identity to his actions. We can divide anonymity into two cases: persistent anonymity (or pseudonymity), where the user maintains a persistent online persona ("nym") which is not connected with the user's physical identity ("true name"); and one-time anonymity, where an online persona lasts for just one use. The key concept here is that of linkability: with a nym, one may publish a number of documents that are all linked together but cannot be linked to the author's true name; by using one-time anonymity for each work, none of the documents can be linked to each other or to the user's physical identity.[1] Forward secrecy refers to the inability of an adversary to recover security-critical information (such as the true name of the writer of a controversial article) "after the fact" (e.g. after the writing is published); where possible, providers of anonymity services should take care to provide forward secrecy, which entails (for in-

[1] Users of anonymity services should keep in mind that essays written by the same person tend to share certain characteristics, and that this fact has been used to identify the authors of anonymous works in the past.

stance) keeping no logs.

We can identify a number of goals for a potential anonymous WWW publishing infrastructure. First, we should be clear that we only concern ourselves with the problem of anonymity for the publisher. In contrast, some existing services, such as the Anonymizer proxy [6], aim to provide privacy for the user who is viewing WWW pages; we explicitly do not attempt to solve that problem. We note that the client privacy problem is orthogonal to the anonymous publishing problem, and that the two problems compose well: if full anonymity is needed (to protect the identity of both endpoints), techniques to anonymize the client will work well in tandem with an infrastructure for anonymous WWW publishing.

We should also point out another problem we explicitly avoid solving. Our infrastructure may reveal the fact that a particular server holds anonymous published data, so long as it does not reveal the data itself or allow the document to be linked to the author. In short, documents and authors may each be individually recognized as anonymous, but we aim to prevent the possibility of linking any one document with its author. As long as there are sufficiently many anonymous authors (some of whom are respected citizens) and sufficiently many anonymous documents (some of which are non-controversial and clearly harmless), there will be "safety in numbers": even if Alice is identified as an author who has published *something* anonymously, Alice will retain very strong plausible deniability protections—the infrastructure will give authorities no reason to suspect that she is involved in writing anything other than harmless documents.

We view anonymous WWW publishing as a problem which can benefit from the experience, approaches, and perspectives of computer security researchers. In particular, we attempt to identify the relevant threats, design robust countermeasures, take advantage of defense in depth, and apply other standard security engineering techniques.

One important goal is to avoid the existence of a central single point of failure. Any such single point of failure becomes a very attractive target—a sitting duck for the motivated adversary. Instead, we strongly prefer to distribute trust to obtain resilience in the event that a few network nodes are compromised or colluding. We wish to see a means by which users can protect their privacy, preferably by putting privacy-enhancing technology directly into their own hands inasmuch as is possible. Where the cooperation of others is necessary to ensure personal privacy, the system should not be easily subverted by the mere collusion or compromise of a few participants.

A corollary is that we should attempt to minimize the length of time which a node possesses security-critical information. By limiting the exposure of information that would reveal the identity of publishers, we reduce the severity of the risk of compromise. In other words, supporting forward secrecy is prudent security practice.

We must recognize that in practice there will be tradeoffs between security and other factors such as performance, availability, stability, etc. Different publishers will likely require different levels of security, and the only party who can soundly evaluate the appropriate degree of security is the publisher himself. Therefore, an important design goal is a flexible security strength "knob" under control of the individual publishers: publishers should be able to select the level of security that is appropriate for themselves.

At this point, it is probably appropriate to offer a quick summary of some possible threats to an infrastructure for anonymous WWW publishing. A minimal threat model might concern itself primarily with political, legal, and social attacks, and give less attention to traditional computer security attack techniques. For instance, an adversary might obtain a subpoena from a judge of relevant jurisdiction to compromise a targeted node in the distributed infrastructure. Even an adversary with few resources can recruit the power of the government to his advantage with such a tactic. Social pressure is a concern as well; carefully-fabricated complaints to a site administrator with power over the targeted node may also suffice.

A more cautious threat model will need to take into account the threat of computer intrusion and attacks on the underlying network. Typically, individual hosts are rather hard to secure, and in practice motivated penetration attempts will often succeed in compromising a significant number of nodes. Furthermore, unprotected network traffic is well-known to be easy to monitor, redirect, forge, and modify. In cryptographic circles, the adversary is often presumed to have complete control over the network; while that may seem a rather paranoid assumption, we know that any system that can stand up to that level of attack will provide very strong security. However, in practice such threat models may lead to unrealistic designs. Therefore, we ought to carefully consider the resources available to likely adversaries, and tailor the countermeasures to the anticipated threat level. It is probably realistic to assume that a limited number of nodes will be compromised, dishonest, or colluding; that the opponent may occasionally be able to successfully penetrate targeted nodes of his choosing; that the adversary may be eavesdropping on certain segments of the network; and that the opponent can perform *traffic analysis* (statistical analysis using only unencrypted header data such as source and destination addresses as well as timing coincidences) using the eavesdropped information. We will attempt to design an infrastructure that is flexible enough to admit variations that will work under a wide array of threat models.

Lest we get caught up in the details of obscure security threats, it is important to remember that security systems can only provide security if they are used. Therefore, we must take care to ensure that usability is not sacrificed without careful consideration. In particular, we want to preserve the vast potential audience provided by the World Wide Web, and so we require that WWW pages published anonymously using our infrastructure be accessible from standard WWW browsers without needing any special client software or anonymity tools. The anonymity services ought to be transparent to the browser software and WWW page reader. In short, enabling widespread deployment and use of this infrastructure is a central goal of this work.

4 Existing Technology

In this section, we describe some existing privacy-enhancing technologies for the Internet [13], in order to see what parts of the solutions they offer can be carried over to the problem of protecting the identity of WWW publishers.

In past years email was the most important distributed application, so it should not be surprising that early efforts at bringing privacy to the Internet primarily concentrated on email protection. Today the lessons learned from email privacy provide a foundation of practical experience that is critically relevant to the design of new privacy-enhancing technologies.

The most primitive way to send email anonymously involves sending the message to a trusted friend, who deletes the identifying headers and resends the message body under his identity. Another old technique for anonymous email takes advantage of the lack of authentication for email headers: one connects to a mail server and forges fake headers (with falsified identity information) attached to the message body. (Both approaches could also be used for anonymous posting to newsgroups.) Of course, these techniques don't scale well, and they offer only very minimal assurance of protection.

The technology for email anonymity took a step forward with the introduction of anonymous remailers. An anonymous remailer can be thought of as a mail server which combines the previous two techniques, but using a computer to automate the header-stripping and resending process [3, 11, 14, 18]. There are basically three styles of remailers; we classify remailer technology into "types" which indicate the level of sophistication and security.

The `anon.penet.fi` ("type 0") remailer was perhaps the most famous. It supported anonymous email senders by stripping identifying headers from outbound remailed messages. It also supported recipient anonymity: the user was assigned a random pseudonym at `anon.penet.fi`, the remailer maintained a secret identity table matching up the user's real email address with his `anon.penet.fi` nym, and incoming email to the nym at `anon.penet.fi` was retransmitted to the user's real email address. Due to its simplicity and relatively simple user interface, the `anon.penet.fi` remailer was the most widely used remailer; sadly, it was shut down recently after being harassed by legal pressure [15].

The disadvantage of a `anon.penet.fi` style (type 0) remailer is that it provides rather weak security. Users must trust it not to reveal their identity when they send email through it. Worse still, pseudonymous users must rely on the confidentiality of the secret identity table—their anonymity would be compromised if it were disclosed, subpoenaed, or bought—and they must rely on the security of the `anon.penet.fi` site to resist intruders who would steal the identity table. Furthermore, more powerful attackers who could eavesdrop on Internet traffic traversing the `anon.penet.fi` site could match up incoming and outgoing messages to learn the identity of the nyms. Worst of all, `anon.penet.fi` becomes a central single point of failure, and it is very dangerous to have such an attractive target.

Cypherpunk-style (type I) remailers were designed to address these types of threats. First of all, support for pseudonyms is dropped; no secret identity table is maintained, and remailer operators take great care to avoid keeping mail logs that might identify their users. This diminishes the risk of "after-the-fact" tracing. Second, type I remailers will accept encrypted email, decrypt it, and remail the resulting message. (This prevents the simple eavesdropping attack where the adversary matches up incoming and outgoing messages.) Third, they take advantage of *chaining* to achieve more robust security. Chaining is a technique that involves sending a message through several anonymous remailers; we discuss chaining more completely below in Section 5. This allows us to take advantage of a distributed collection of remailers; diversity gives one a better assurance that at least some of the remailers are trustworthy, and chaining ensures that one honest remailer (even if we don't know which it is) is all we need. Type I remailers can also randomly reorder outgoing messages to prevent correlations of ciphertexts by an eavesdropper. (Note that this feature imposes substantial delays; such increased latency would of course be unacceptable for Web applications.) In short, type I remailers offer greatly improved security over type 0, though they do have some limitations which we will discuss next.

The newest and most sophisticated remailer technology is the Mixmaster, or type II, remailer [5, 7]. They extend the techniques used in a type I remailer to provide enhanced protection against eavesdropping attacks. First, one always uses chaining and encryption at each link of the chain. Second, type II remailers use constant-length messages, to prevent passive correlation attacks where the eavesdropper matches up incoming and outgoing messages by size. Third, type II remailers include defenses against sophisticated replay attacks. Finally, these remailers offer improved message reordering code to stop passive correlation attacks based on timing coincidences (which adds still more latency). Because their security against eavesdropping relies on "safety in numbers" (where the target message cannot be distinguished from any of the other messages in the remailer net), the architecture also calls for continuously-generated random cover traffic to hide the real messages among the random noise.

Another new technology is that of the "newnym"-style nymservers. These nymservers are essentially a melding of the recipient anonymity features of a `anon.penet.fi` style remailer with the chaining, encryption, and other security features of a cypherpunk-style remailer. A user obtains a pseudonym (e.g. `joeblow@nym.alias.net`) from a nymserver, and mail to that pseudonym will be delivered to him. However, unlike `anon.penet.fi`, where the nymserver operator maintained a list matching pseudonyms to real email addresses, newnym-style nymservers only match pseudonyms to "reply blocks": the nymserver operator does not have the real email address of the user, but rather sends the email through a fixed *chain* that eventually ends up at the real user. We will use a similar technique in Section 5 below.

With the increasing sophistication in remailer technology, we find that modern remailers have been burdened with a correspondingly complicated and obscure interface. To deal with this unfriendly mess, client programs have sprung up to provide a nicer interface to the remailers. Raph Levien's `premail` [16] is the archetypical example. Even so, using remailers still requires some knowledge; for even greater user-friendliness, we need this support to be integrated into popular mail handling applications. The need for specialized clients has made it difficult to widely deploy support for anonymous email; from this experience, we conclude that an anonymous WWW publishing infrastructure should not require special tools to browse WWW pages created by anonymous authors.

The "strip identifying headers and resend" approach used by remailers has recently been applied to provide anonymity protection for Web browsing as well. Community ConneXion has sponsored the Anonymizer [6], an HTTP proxy that filters out identifying headers and source addresses from the WWW browser. This allowing users to surf the WWW anonymously without revealing their identity to WWW servers. However, the Anonymizer offers rather weak security—no chaining, encryption, log safeguarding, or forward secrecy—so its security properties are roughly analogous to those of type 0 remailers. Other implementations have since appeared based on the same approach [8, 10].

5 The Problem at Hand

We now turn our attention to adapting the above-mentioned technologies to the problem of anonymous WWW publishing. There are two important distinctions that must be made between the email and WWW realms:

- The WWW is an interactive medium, while email is store-and-forward.
- More notably, email is a "push" technology; the source of the data initiates the transfer, possibly without even the knowledge or consent of the receiver. By contrast, HTTP is a "pull" technology; the intended receiver of the data explicitly requests it from the sender.

Some of the security in remailer technology is obtained by randomly reordering and delaying messages (on time scales ranging from minutes to days). The first point observes that these delays are unacceptable for WWW requests. The second point, however, offers us some extra benefit. As we shall see, the security of a system such as ours or of the remailer network increases as the number of publicly-accessible cooperating nodes increases. In the realm of email, operators of remailers often come under fire when their services are abused by people sending threatening letters or untraceable spam; the undesirability of handling irate end-users causes the number of remailers to stay low, potentially impacting on the security of the overall system. By contrast, an HTTP server cannot initiate a connection with an unwilling client and send it data when no request was made. This "consensual" nature of the WWW should cause fewer potential node operators to become discouraged, and lead to corresponding increases in security.

5.1 Rewebbers

5.1.1 Description

Here, then, is the plan. We situate around the world a number of nodes which we call *rewebbers* (collectively, these nodes are known as the *rewebber network*). Each node is essentially an HTTP proxy which understands "nested" URLs (i.e. URLs of the form `http://proxy.com/http://realsite.com/`); most existing HTTP proxies have this behavior. The basic idea is then to publish a URL such as the one above, which points to `proxy.com` instead of `realsite.com`.

An obvious objection at this point is that this does not hide the real server name. We solve this problem by using public-key cryptography to encrypt the real server name so that only the rewebber can read it. In this case, the URL would look something like `http://proxy.com/!RFkK4J...d_4w/`. The rewebber at `proxy.com`, upon receiving this URL, would first decrypt it (the fact that it is encrypted is indicated by

$$\text{http://A/}\boxed{K_A\text{, http://B/}\boxed{K_B\text{, http://C/}\boxed{K_C\text{, http://real.site/foo.htm}}_C}_B}_A$$

Figure 1: Schematic diagram of a locator: boxes with a subscript indicate encryption with a public key; the K's are DESX keys to use to decrypt the data that is returned. A, B, and C are sites running rewebbers.

the leading `!` in lieu of the expected `http://`), and then proceed to retrieve the nested URL in the normal fashion. We call such encrypted URLs *locators*.

This mechanism hides the real location of the server from the client, but has other flaws. First of all, once the client has retrieved the data from the rewebber, he could use one of the more aggressive WWW search engines to try to find where the document originally came from. We solve this problem by encrypting the document before storing it on the server. Thus, if the document is accessed directly (through a search engine, for example), it will be random data (but see Section 7 for an interesting extension). The encryption used here ought not be public-key; it should be a symmetric-key algorithm. (We used DESX [19] in our implementation, but any strong symmetric-key algorithm would do. Further references to DESX should be taken as referring to whatever symmetric-key algorithm is in use.) The DESX key is given to the rewebber in the encrypted part of the locator; that is, when the rewebber decrypts the locator, it finds not only a URL to retrieve, but a DESX key with which to decrypt the document thus retrieved. It then passes the decrypted document back to the client.

This technique of encrypting the document stored at the server has another benefit as well: the document can be padded in size before being encrypted, and the rewebber will truncate the document to its original size before passing it back to the client. The reason for doing this is similar to that for encrypting the document in the first place; if the retrieved document is 2843 bytes long, for example, a WWW search for encrypted documents near that size would quickly narrow down the possible choices. To thwart this, we always add random padding to the end of encrypted documents so that their total length is one of a handful of fixed sizes. We chose sizes of 10000, 20000, 40000, 80000, etc. bytes, as data in [12] indicate that the median size of files on the WWW is a little below 10000 bytes.

We do not use latency-intensive techniques such as full Chaumian mixes, message reordering, or injecting random delays; interactive WWW browsing is too delay-sensitive to allow those techniques. This is unfortunate, but we feel that the resulting decrease in security is somewhat mitigated by some other features of our system.

If the example above were used as it stood, the rewebber would see both the real URL of the document, as well as the decrypted document itself. Thus, if the rewebber is compromised (if its private key leaks out, it is forced to reveal information, or even had it been being operated by an adversary all along), it could reveal the mapping between the real site and the document. The solution is to implement *chaining*, in the same manner as the type I remailers mentioned earlier.

To implement chaining, we simply replace the URL in the encrypted portion of the locator with another complete locator, one which points to a rewebber at a different site (and preferably, in a different legal jurisdiction). We can repeat this process, thus making the rewebber chain listed in the locator (in encrypted form) as long as we like. A schematic diagram of a locator containing a rewebber chain of length 3 appears in Figure 1. In the next subsection, we identify tradeoffs associated with choosing a length for a rewebber chain.

When using chaining, the document will need to be multiply encrypted on the server. To do this, the publisher randomly selects a DESX key for each rewebber in the chain, encodes them into the locator as depicted in Figure 1, and iteratively encrypts the document. In this way the publisher forms a locator and announces it to the world (by using an anonymous remailer to post it to a newsgroup, for example). Note that all of the security parameters, including the number of rewebbers in the chain, are under the control of the publisher; individual publishers may adjust this "knob" to suit their anonymity needs.

The main benefit of chaining is that only the rewebber closest to the client ever sees the decrypted data, and only the rewebber closest to the server knows where it is really getting data came from. In order to link the two, the cooperation of every rewebber in the chain would be necessary. We note that this avoids the existence of a single point of failure, and allows us to distribute trust throughout the network. The use of chaining and locators here is the rewebber's counterpart of the use of "reply blocks" by nymservers (mentioned above). Remember that there will likely be many opaque WWW requests flowing through the network at any one point; there is "safety in numbers".

Finally, we point out a serendipitous aspect of using HTTP proxies to implement rewebbers. These proxies tend to have caches, in order to improve network performance. Somewhat conveniently, they also turn out to give us an extra benefit. The fact that data is being cached at the intermediate rewebbers in a chain makes it less likely that requests will actually get forwarded all the way back to the publisher. This makes traffic analysis somewhat harder to accomplish. Note that traffic analysis relies on having a large sample of network data to perform statistical analysis; by reducing the amount of traffic on the network, less data is available to a traffic analyst, and so statistical patterns will be harder to discern. Also note that this form of caching gives us the political benefit that rewebber nodes never cache unencrypted data, and so it is less likely they can be held accountable for controversial material they happen to be caching.

5.1.2 Analysis

In this section, we make some computations regarding the security and the reliability of the rewebber network described above. Here, a *node* is a site running a rewebber proxy; a *compromised node* is a node for which logs are available to an adversary (in order that the adversary may learn which decrypted URL (or locator) corresponds to which encrypted locator), or simply one for which the private key is so available (whether because of host compromise, or because the operator of the node was an adversary in the first place). Let n be the number of nodes in the rewebber network, and let c be the number of compromised nodes. Suppose a publisher wished to pick k nodes through which to chain. In the *worst case*, we would have no clue which nodes were compromised (or even which were more likely to be compromised), and he would choose the k nodes uniformly at random with replacement (as we assume k is small as compared to n, sampling with or without replacement makes little difference, but sampling with replacement gives a worst-case bound).

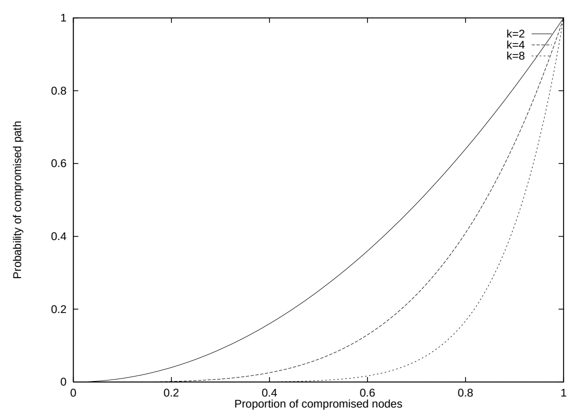

Figure 2: Probability of compromised path

We now calculate the probability of a *compromised path*. Here our threat model is that all of the nodes in the chain collude to decrypt the publisher's published locator, in order to determine the publisher's WWW site (assumedly from that information learning some information about his identity). In this model, the worst-case probability of a compromised path is $\left(\frac{c}{n}\right)^k$. A plot of this value for various values of k appears in Figure 2. Looking at this result, it would seem that choosing a larger value of k would always be preferable.

However, we now consider *reliability*. We call a node *broken* if for some reason it is unable (or unwilling) to decrypt locators and forward the resulting URLs (or locators). If b is the number of broken nodes in the network, then a chain of k rewebbers will suffer *path failure* if any of the nodes on it is broken. Assuming that the broken nodes are randomly distributed (at least with respect to the nodes in the chain), this occurs with probability $1 - \left(1 - \frac{b}{n}\right)^k$. This value is plotted for various values of k in Figure 3.

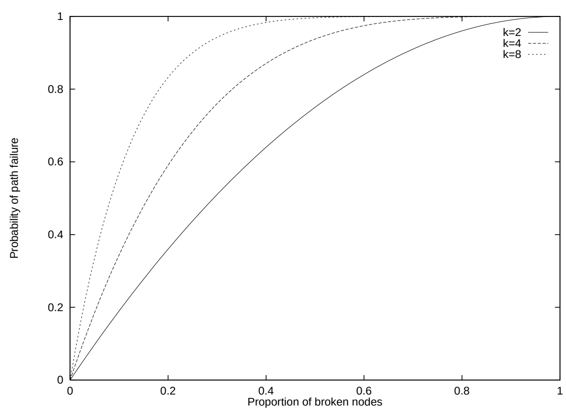

Figure 3: Probability of path failure

When choosing k, then, a tradeoff needs to be made between the probability of path compromise and the probability of path failure. The smart choice is usually to give higher priority to preventing path compromise; path compromise is catastrophic, but path failure is "fail-safe": requests will not succeed until a new locator is created and published, but the identity of the publisher will not be revealed.

Some additional heuristics can be used to further decrease the probability of path compromise. First of all, a slight decrease can be gained by choosing the rewebbers to form a chain without replacement (so that no rewebber appears twice in a chain). More interestingly, we could use knowledge about the independence (or lack thereof) of rewebbers. It seems logical to assume that the closer two rewebbers are in terms of administrative control, the more likely it would be that their compromises would be correlated. For example, two rewebbers run by the same person are likely to have high compromise correlation; rewebbers run in the same administrative domain would have somewhat high correlation; rewebbers in the same legal jurisdiction would have higher correlation than rewebbers in different countries, etc. Therefore, a clever choice of rewebber chain would tend to include rewebbers being run by different people, preferably in different countries.

5.1.3 Implementation

We implemented a rewebber on top of the scalable TACC server architecture described in [12]. This architecture provided us with a high-performance scalable HTTP proxy and caching system, on top of which it was straightforward to implement the extra functionality necessary for a rewebber. We observe excellent performance with our prototype implementation; CPU costs are minimized with the scalable TACC server, so that network costs dominate, and network costs are themselves reduced with the caching provided by the TACC architecture.

We also implemented the handful of tools necessary for publishers to pick random DESX keys, create locators, and encrypt the files they wish to place on their server.

All of the crypto portions of the code will likely be recoded from scratch the next time the first author goes to a country with somewhat less ridiculous laws about publishing cryptography, so that they can be released.[2]

[2]The irony of being unable to publish a system whose function is to enable anonymous publication is noted.

5.2 TAZ Servers

Once a rewebber network is deployed, the stage will be set for the ability to publish anonymously on the WWW. One major drawback, however, is that a locator that contains just a simple chain of rewebbers looks something like this:

```
http://rewebber.com/!RjVi0rawjGRT50ECKo_UBa
7Qv3FJlRbyej_Wh10g_9vpPAyeHmrYE1QL1H2ifNh2M
a4UYt3IaqeQRXXd7oxEvwR8wJ3cnrNbPF6rc1Uzr6mx
IWUtlgW0uRlL0bGkAv3fX8WEcBd1JPWGT8VoY0F1jxg
PL70vuV0xtbMPsRbQg0iY=RKLBaUYedsCn0N-UQ0m5J
WTE1nuoh_J5J_yg1CfkaN9jSGkdf51-gdj3RN4XHf_Y
Vyxfupgc8VPsSyFdEeR0dj9kMHuPvLivE_awqAwU_3A
f8mc44QBN0fMVJjpeyHSa79KdTQ5EGlPzLK7upFXtUF
cNLSD7YLSc1gKI3X8nk15s=RXbQaqm0Ax4VhKPwkLVK
_MMJaz9wchn_pI48xhTzgndt5Hk09VToLyz7EF4wGH3
XKPD7YbKVyiDZytva_sUBcdqpmPXTzApYLBnl4nD0y1
o1Pu1Rky8CxRfnC9BvQqof853n99vkuGKP9K4p3H7pl
6i8D0at-Nr0Indpz5xgwZKc=/
```

It is evident that there is a naming problem and that it needs to be solved.

We therefore create a virtual namespace called the `.taz` namespace, and we create new servers called *TAZ servers* to resolve this namespace. TAZ stands for Temporary Autonomous Zone [4]; the emphasis here is that `.taz` sites can relocate quickly, utilizing the level of indirection offered by the TAZ server.

The function of a TAZ server is to offer publishers an easy way to point potential readers at their material, as well as offering readers an easy way to access it. A TAZ server consists essentially of a *public* database mapping virtual hostnames ending in `.taz` to rewebber locators. The emphasis on "public" is to stress that nothing in this database is a secret; unlike `anon.penet.fi`-style (type 0) remailers (which associate a virtual email address with a real one), TAZ servers merely associate `.taz` addresses with locators. (In practice, the TAZ server database would also usually contain a hash of a password, so as to effect access control for updates to the database, but this need not be secret information.) Most importantly, the TAZ server operator *cannot decrypt the locators* that are in its database.

The public nature of the TAZ database brings several benefits. First of all, it provides protection for TAZ server operators: there is no incentive to threaten them with legal, social, or political pressure, because any information the operator can access is also publicly available. It would be prudent to widely publicize this fact, in order to reduce the inevitable flood of requests for identity information.[3] Second, it separates the document location problem from the rewebber network, so we get extensibility: TAZ servers can be readily upgraded without needing to change existing rewebber code. Finally, the purely public nature of TAZ databases gives very strong security benefits, because it ensures that the TAZ servers cannot reduce the security of the system. TAZ servers may very well become extremely complicated in the future, if some type of coherent dynamic distributed database were implemented. In such a future, we would be very glad that the security of our system does not rely on the security of the TAZ servers.

We are not overly concerned with the authenticity of the TAZ database, because the best solution is to rely on end-to-end authentication [22]. Anonymous publishers concerned about forgeries should establish a public key for their pseudonym, sign all anonymous documents with this key, and distribute their nym's public key widely along with their `.taz` hostname.

One very important feature to note about TAZ servers is that the mechanism it provides (resolving virtual names to locators) is completely orthogonal to the rewebber mechanism; there is no need for the operator of a TAZ server to also run a rewebber, nor vice versa. A TAZ server is merely a convenient central repository for a public database.

Currently, we have two interfaces to a working TAZ server available; each interface offers a slightly different way of sending a rewebber locator to any standard WWW client. At

[3] Remailer operators get asked all the time to divulge the identity of the originator of various pieces of email, even though the headers on the email in question clearly state that the operator does not have the ability to do so.

`http://www.isaac.cs.berkeley.edu/taz/` is a list of all currently registered `.taz` hosts. From this page, one can click on a host's name, and be taken to that page (the `HREF` in the link points to the locator for the host). Also, to access a specific host (`cellspec.taz` for example), one can just use the URL `http://www.isaac.cs.berkeley.edu/taz.cgi/cellspec.taz/` (the CGI on this page issues an HTTP Redirect which points to the locator).

These centralized approaches will work fine, until the namespace gets too big or the central `.taz` server becomes a bottleneck; then we will probably prefer to move to a decentralized distributed solution for better scalability and availability. This issue is addressed further in Section 7.

6 Related Work

Ross Anderson proposes the Eternity Service [2], which is intended to provide very long-term digital storage and publication. It is intended to offer strong availability guarantees even in presence of adversarial attacks. While the Eternity Service shares several similar features with anonymous WWW publication, the Eternity Service is especially geared towards highly-available publication; anonymity is not a direct goal. The contrast is perhaps best appreciated with a direct quote from the paper:

> The net effect will be that your file, once posted on the eternity service, cannot be deleted. As you cannot delete it yourself, you cannot be forced to delete it, either by abuse of process or by a gun at your wife's head.

Note that our design would attempt to stop threats like "a gun at your wife's head" by protecting your anonymity, so that the thugs will not know whose wife to point a gun at.

Anderson discusses a number of engineering tradeoffs involved in building such a service. One interesting observation is that anonymous communication models can stand as a countermeasure against denial of service attacks; thus, in the context of the Eternity Service anonymity is viewed as a useful technique to increase availability, rather than a goal in and of itself. No implementation or full design of Anderson's functional specification is available, probably because of the very aggressive design goals, but we can learn some interesting lessons about the engineering tradeoffs in highly available publication from this paper.

There has also been some recent work on building infrastructures for general-purpose interactive anonymous communication on the Internet. We can learn a great deal about the challenges of anonymity for low-latency communications. One important lesson to take away from these efforts is that there are a great deal many clever and powerful attacks against low-latency anonymized traffic when adversaries have control over many segments of the network.

Wei Dai's PipeNet architecture [9] is based on a distributed network of packet forwarders, connected by long-term encrypted communications links that are padded to constant bandwidth when not in use to prevent traffic analysis; Chaumian mixes are used at each router node. The result is much like a low-latency IP-level cousin of the type II remailer network, with the addition of a lot of cover traffic to make up for the increased potential for timing attacks that arises from the low-latency requirements. The requirement for constant-bandwidth long-lived encrypted links incurs serious performance costs to provide security against this very cautious threat model of pervasive eavesdropping on the network. No complete PipeNet design, much less implementation, is available, and PipeNet does not appear to be a serious candidate for practical deployment in the short term; but PipeNet was an influential early proposal for anonymous communication that established the possibility a high level of security and illuminated a number of subtle possible attacks on such a system.

A more recent and mature research effort to build a practical infrastructure for interactive anonymous Internet traffic is the Onion Routing project [20]. The Onion Routing proposal embodies slightly more practically-oriented engineering decisions than vanilla

PipeNet. For instance, links are not padded to constant bandwidth; instead, Chaumian mixes and "safety in numbers" is used to protect against traffic analysis. The authors do not attempt to protect against attackers who can control (or monitor) many diverse portions of the network simultaneously. By taking on slightly less ambitious goals, the Onion Routing researchers have managed to design a full system and preliminary implementations are available.

Our rewebber system bears a certain resemblance to PipeNet and Onion Routing; after all, it is essentially attempting to anonymize low-latency IP traffic. However, for our sanity, we have simplified the problem in a number of ways. First, our rewebbers operate at the application layer, rather than the network layer, which enables us to make several optimizations and simplifications: we take advantage of the request-reply nature of HTTP, and take advantage of the weak semantics of HTTP as well as the relatively static nature of WWW pages to cache data throughout the system. Second, to achieve very rapid implementation and deployment, we have weakened the threat model somewhat; protection against network attacks is deemed a lower priority (and much harder problem) than immediate deployment with protection against political, legal, and social attacks. We expect to build upon the rewebber system to improve its security against network attacks as experience with Onion Routing and the general low-latency communications anonymity problem improves. Third, we note that infrastructures for routing general IP packets anonymously is subject to a potentially unprecedented levels of abuse: malicious hackers could use this service to break into vast numbers of Internet-connected hosts without fear of discovery. In contrast, by restricting ourselves to the problem of anonymous web publication, we can avoid worrying about abuse. Finally, we note that once the Onion Routing infrastructure becomes widely available, we could easily add support for anonymous publishers connected via Onion Routing links, if we expected that publishers were sophisticated enough to take advantage of the Onion Routing infrastructure; that allows us to take advantage of the best of both worlds.

URNs (Universal Resource Names) [21] attempt to address nearly the same naming problem as TAZ servers do. Both attempt to provide short persistent names for resources via a simple resolving mechanisms, using the standard computer science trick—a level of indirection. The major difference is that, when a URN ultimately points to a resource described by a URL, it is a design goal that one be able to learn the pointed-to URL from the URN; in contrast, the whole point of the anonymous WWW publishing infrastructure is that, given a TAZ name, one cannot obtain the URL for the author's real site. These differences notwithstanding, future experience with URN resolving mechanisms is likely to lead to improved name resolution service from the TAZ servers.

7 Future Work

There are a number of possible improvements to the rewebber system. Currently we implement some but not all of the features of type I and type II remailers; we do not provide strong security against wide-area network attacks at present. It would be interesting to explore techniques for better security. For instance, we could apply "fan-out"—where one request to a first-hop rewebber spawns several parallel requests to unrelated sites (whose responses are ignored)—to attempt to generate some cover traffic and disguise the location of the publisher's endpoint. A variation on "fan-out" can be used to attempt to increase availability in the presence of network faults: include several independent paths to the content in the locator, and then if one fails because of a network error, the content can still be retrieved. More generally, these "fan-out" schemes can be generalized using k-out-of-n secret sharing schemes, where a single client request fans out into n requests inside the rewebber network, and any k responses suffice to recover the content. All of these schemes need to be analyzed to examine just how much benefit they provide, and what cost they impose upon the system.

It would be valuable to carefully analyze the security benefit that application-level caching affords us. We have not worked this out in detail yet, because at this point we consider eavesdropping and other network attacks a second priority. However, it appears that very large benefits may potentially be available, especially if caching is combined with a special high-security mechanism using considerable re-ordering and random delays for non-interactive prefetching to warm the caches.

Another improvement to the rewebber system would take advantage of steganography [1] to hide the fact that a particular author has published some (unspecified) works anonymously. Currently an anonymous author must store encrypted data on his WWW server, but ciphertext usually stands out; by disguising the encrypted data file within normal-looking data (e.g. within the low bits of a large image or sound file), we can avoid this problem. That would allow an anonymous publisher to camouflage himself and blend in among all the other noise on the World Wide Web.

There is the potential for a great deal of future work in the TAZ servers. Centralized solutions such as the ones we have implemented may work well for a while, but in the future decentralized solutions may be preferable, for both scalability and availability reasons. Of course, a distributed naming database would incur all the coherency, performance, and other problems found in a dynamic distributed database; we expect that a great deal can be learned from DNS. Two possible decentralized methods worthy of further study are multicast capable servers, and Usenet-based updates (a form of slow multicast). If the `.taz` namespace starts becoming too large, we will have all the problems of a static `/etc/hosts` file with our current implementation, and a transition to a more scalable solution will be of paramount importance.

In any case, more implementation and real-life deployment experience is needed to understand the engineering tradeoffs better. Fortunately, by separating the TAZ database from the rewebber system, and by ensuring that the TAZ database contains only public data (and so is not security-critical), we have reserved great flexibility for upgrading this portion of the system.

8 Conclusion

We have designed a practical infrastructure that enables anonymous publishing on the World Wide Web. Some key features include universal accessibility by standard clients, short persistent names, security against social, legal, and political pressure, protection against abuse, and good performance.

We have completed a prototype implementation, and we expect to start deploying more mature code for practical use this summer. Implementation was eased by using the scalable TACC server architecture [12]. Extensibility is provided by separating the naming and document location problem from the core document anonymization services.

One novel contribution was the observation that caching can be used to improve not just performance but also improve security against network eavesdroppers as well. More generally, by providing anonymity services at the application layer, we were able to take advantage of the semantics of the World Wide Web to simplify and optimize our system.

In short, we have demonstrated that anonymous publishing for the World Wide Web is well within practical reach today.

9 Acknowledgements

We gratefully acknowledge Raph Levien for suggesting many of the general directions this work has taken.

References

[1] Ross Anderson, "Information Hiding—An annotated bibliography," 1995.
`http://www.cl.cam.ac.uk/users/fapp2/steganography/bibliography/index.html`

[2] Ross Anderson, "The Eternity Service," *PRAGOCRYPT 96*.
`ftp://ftp.cl.cam.ac.uk/users/rja14/eternity.ps.Z`

[3] Andre Bacard, "Anonymous Remailer FAQ," 1996. `http://www.well.com/user/abacard/remail.html`

[4] Hakim Bey, *T.A.Z. : the temporary autonomous zone, ontological anarchy, poetic terrorism*, Autonomedia, New York, 1991.

[5] David Chaum, "Untraceable Electronic Mail, Return addresses, and Digital Pseudonyms," *Communications of the ACM*, February 1981, vol. 24 no. 2. `http://www.eskimo.com/~weidai/mix-net.txt`

[6] Community ConneXion, "Anonymous Surfing," 1996. `http://www.anonymizer.com/`

[7] Lance Cotrell, "Mixmaster & Remailer Attacks," 1995. `http://www.obscura.com/~loki/remailer/remailer-essay.html`

[8] Ray Cromwell, "Welcome to the Decense Project," 1996. `http://www.clark.net/pub/rjc/decense.html`

[9] Wei Dai, "PipeNet," Feb 1995, post to the `cypherpunks` mailing list.

[10] Laurent Demailly, "Announce: Anonymous Http Proxy (preliminary release)," Usenet post. `http://www.lyot.obspm.fr/~dl/anonproxy.txt`

[11] Arnoud Engelfriet, "Anonymity and Privacy on the Internet," 19 Dec 1996. `http://www.stack.nl/~galactus/remailers/index.html`

[12] Armando Fox et al., "Scalable Network Services," Submitted under blind review, 1997.

[13] Ian Goldberg, David Wagner, Eric Brewer, "Privacy-enhancing Technologies for the Internet," *Proceedings of IEEE COMPCON '97*, 1997.

[14] C. Gulcu and G. Tsudik, "Mixing E-mail with Babel," *Proc. Symp. Network and Distributed System Security*, 1996, pp. 2–16.

[15] Johan Helsingius, press release, 30 August 1996. `http://www.cyberpass.net/security/penet.press-release.html`

[16] Raph Levien, "premail". `http://www.c2.net/~raph/premail.html`

[17] Veran Matic, "Another 'Jamming Device' Activated Against Radio B92," Press release, 1 Dec 1996. `http://www.dds.nl/~pressnow/news/9611300.htm`

[18] Andreas Pfitzmann and Michael Waidner, "Networks without user observability—design options," *EUROCRYPT 85*, LNCS 219, Springer-Verlag, pp. 245–253. `http://www.informatik.uni-hildesheim.de/~sirene/publ/PfWa86anonyNetze.html`

[19] Bruce Schneier, *Applied Cryptography*, second edition, John Wiley & Sons, 1996.

[20] Paul Syverson, David Goldschlag, Michael Reed, "Anonymous Connections and Onion Routing," draft manuscript. `http://www.itd.nrl.navy.mil/ITD/5540/projects/onion-routing/overview.html`

[21] K. Sollins and L. Masinter, "Functional Requirements for Uniform Resource Names," RFC1737.

[22] J. Saltzer, D. Reed, and D. Clark, "End-to-end Arguments in System Design," *ACM Transactions on Computer Systems (TOCS)*, Vol. 2, No. 4, pp. 195–206, 1984.

[23] Rebecca Vesely, "Banned on Radio, Belgrade Dissidents Take to Net," Wired News, 3 Dec 1996. `http://www.wired.com/news/politics/story/777.html`

[24] Rebecca Vesely, "Net Presence Widens as Serbia Shuts Down Media," Wired News, 4 Dec 1996. `http://www.wired.com/news/politics/story/806.html`