# Toward an Information Theoretic Secure Anonymous Communication Service

## The Pynchon Gate Pseudonymous Mail System

Len Sassaman[1], Nick Mathewson[2], Brian Warner[3], Bram Cohen[4], and Bart Preneel[1]

[1] Katholieke Universiteit Leuven
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{len.sassaman,bart.preneel}@esat.kuleuven.be
[2] The Free Haven Project
nickm@freehaven.net
[3] Allmydata, Inc.
555 De Haro Street, Suite 400, San Francisco CA, 94107 USA
warner-pynchon@lothar.com
[4] BitTorrent, Inc.
201 Mission Street, Suite 900, San Francisco, CA 94105
bram@bittorrent.com

**Abstract.** We describe the Pynchon Gate, a practical pseudonymous message retrieval system. Our design uses a simple distributed-trust private information retrieval protocol to prevent adversaries from linking recipients to their pseudonyms, even when some of the infrastructure has been compromised. This approach resists global traffic analysis significantly better than existing deployed pseudonymous email solutions, at the cost of additional bandwidth. This design improves upon previous versions by being resistant to *Byzantine servers*. We provide a review of related work, and evaluate it in comparison to our approach. We examine security concerns raised by our model,and propose solutions. Special emphasis is placed on ease of implementation and deployment of the system, as well as end-user usability issues.

## 1 Introduction

Over the last several decades, there have been numerous systems proposed which aim to preserve the anonymity of the recipient of some data. Some have involved trusted third-parties or trusted hardware; others have been constructed on top of link-layer anonymity systems or mix networks.

Pseudonymous messaging services allow users to send messages that originate at a pseudonymous address (or "nym") unlinked to the user, and to receive messages sent to that address, without allowing an attacker to deduce which users are associated with which pseudonyms. These systems can be used for parties to communicate without revealing their identities, or can be used as a building-block for other systems that need a bi-directional anonymous communication

channel, such as Free Haven [27]. But, as we will argue below, most existing deployed solutions are either vulnerable to traffic analysis or require unacceptably large amounts of bandwidth and storage as the number of users and volume of traffic increase.

We review the architecture and design of our proposed solution: the Pynchon Gate [58], a design that uses private information retrieval (PIR) [13] primitives to build a secure, fault-tolerant pseudonymous mail retrieval system.

In our system, pseudonymous users (or "nym holders") use an existing anonymous email network (such as Mixmaster [52] or Mixminion [22]) to send authenticated requests to a *nym server*, which delivers outgoing messages to the email network and handles administrative commands. The nym server also receives incoming messages and passes them to a *collator* component, which encrypts the messages and periodically packages them into regular batches. These batches are then replicated at a number of *distributor* servers, which use a private information retrieval protocol to allow nym owners to receive mail while maintaining unlinkability between a message and its recipient.

**Goals.** First, our design must be *secure*: we want the Pynchon Gate to resist active and passive attacks at least as well as the state of the art for forward message anonymity. Thus, we should try to protect users' identities from a global eavesdropper for as long as possible; to hinder active attackers who can delay, delete, or introduce traffic; and to resist an attacker who has compromised some (but not all) of the servers on the network.

In order to provide security, however, we must ensure that the system is *deployable* and *usable*: since anonymity and pseudonymity systems hide users among each other, fewer users means less protection [1]. Thus, we should handle node failure without loss of mail; we must not require more bandwidth than volunteer servers can provide or users are willing to use; and we should not require a complicated interface.

**In this paper.** We begin in Section 2 with a discussion of related work, and an overview of known attacks against existing pseudonymity systems. (To motivate our work, section 5.2 presents new analysis on the effectiveness of passive traffic analysis against current reply-block based nym servers.) Section 3 presents the Pynchon Gate in more detail, describing its organization, design rationales, and network formats. We describe our simple PIR protocol in Section 4, and discuss methods of extending it to resist Byzantine server actions. In Section 5 we analyze security, and in Section 6 we discuss optimizations and performance qualities of our solution. We close with an evaluation of our design in Section 7.

## 2 Background

Here we present a brief outline of existing pseudonymity solutions and discuss their limitations and attacks against them.

### 2.1 Related Work

First, we discuss existing designs for pseudonymous message delivery. Many assume the existence of a "forward" anonymous channel that a sender can use to send a message to a known recipient while preventing the recipient, the infrastructure, and any attackers from knowing who is communicating with whom. Currently deployed designs are based on Chaum's mix [12] architecture, and include the Mixmaster [52] and Mixminion [22] anonymous remailer networks. It is trivial to use these systems to *send* pseudonymous messages: the sender can make an anonymous message pseudonymous by signing it with a public key associated with her pseudonym. Thus, these designs focus on how to *receive* messages sent to a pseudonymous address.

Other descriptions of the use of PIR in preserving recipient anonymity have been independently proposed but not deployed. Earlier work by Jim McCoy describes a similar architecture to the Pynchon Gate, but does not use an information-theoretic primitive for preserving privacy [50]. Independent work by Cooper and Birman [15] describes a PIR-based message service for mobile computing systems, and Berthold, et al. have presented work [5] which shows that simple optimizations to the PIR protocol are possible.

**Reply blocks and return addresses.** In 1981, Chaum [12] described a method of using *return addresses* in mix-nets: recipients encode a reply path, and allow senders to affix messages to the encoded path. As the message moves through the network, the path is decoded and the message encoded at each hop, until an encoded message reaches its eventual recipient. This system relies upon all selected component nodes of the chosen path remaining operational in order for mail to be delivered, which can make the system too unreliable for practical use if significant time elapses between path generation and message origination.[5]

In addition to reliability issues, some implementations of these "reply blocks" suffer from a pseudonym management perspective. Cypherpunk nym servers based on the first generation implementation of Chaum's mix-nets (Type I remailers [31]), such as `alpha.c2.net` [3] and `nym.alias.net` [49], implement a central reply-block repository that allowed the pseudonym holders to receive messages delivered to a email address. Unfortunately, Type I remailers allow multiple uses of their reply blocks, which are vulnerable to replay and flooding attacks as discussed in [16, 48]. Type II (Mixmaster) and Type III (Mixminion [22]) systems do not permit multiple-use reply blocks, and prevent replay attacks [17].

**Single-use reply blocks.** While the Type II system does not support anonymous reply blocks, the Type III (Mixminion) system introduces single-use reply

[5] Forward-only messages through a mix-net, however, are sufficiently reliable. The client software can evaluate network health information [54, 42] before sending a message, and thus can construct robust remailer chains based on the current health of the remailer network.

blocks (SURBs) [41] to avoid replay attacks. The Type III protocol requires the recipient to create a large number of reply blocks for senders to use. In practice, this is likely to be automated by a nym server [44] that stores a number of SURBs and uses them to deliver pseudonymous mail to to the recipient—one such design is Underhill [46]. Type III also has the property that the forward and reply messages share the same anonymity set, and recent work has been done by Danezis and Laurie on attack-resistant anonymous packet formats suitable for reply messages [23]. However, since reply blocks are still used, reliability issues remain: if any given node in the pre-selected SURB's path is defunct during the interval in which the mail is to be delivered, the mail is lost. Reply block systems are also susceptible to intersection attacks [7]: a global observer can collect data on who is sending and receiving mail, and given enough time and data, can reliably determine who is talking to whom [19].

**Network-level client anonymity.** The ZKS Freedom Network [8] provided anonymous access to a POP3 server [51], enabling its users to maintain pseudonyms using standard email protocols. Freedom was discontinued due to high operating expenses, especially in bandwidth. Other network-level anonymity systems, such as Pipenet [18], Onion Routing [35], the Java Anon Proxy [6], or Tor [29], could be used in much the same fashion; unfortunately, they face the same barriers to widespread deployment [34]. Attempts to address the practical barriers to deployment of low-latency anonymity systems have resulted in designs which are at greater risk to traffic analysis methods such as end-to-end timing attacks [21, 20, 53, 25] It is possible that such a low-latency system may be developed which is both secure against end-to-end analysis and cost-effective to operate, but no such system has yet been proven feasible.

**Network-level server anonymity.** The second generation implementation of Onion Routing, Tor [29], implements rendezvous points [33] that allow users to offer location-hidden services. A user wishing to anonymously receive messages can use this to receive mail at a hidden location: messages are delivered to the server over the Onion Routing network, and successful delivery does not require the sender to know the IP address of the destination server.

Rendezvous points offer an alternative method of leveraging network-level anonymity systems for anonymous mail receipt; however, they do not address the previously mentioned concerns with these anonymity systems.

**Re-encryption mixes.** Re-encryption mixes [37] aim to improve the reliability of anonymous message systems. Recent work has shown that re-encryption mixes can be used to facilitate anonymous message replies [36]. While reusable anonymous return channels in re-encryption mixes do improve on the robustness of simple reply blocks in a Chaumian mix-net, reliability problems are still possible. Re-encryption mixes require that the security vs. reliability tradeoffs be made by the sender at the time that the message is sent. A more desirable

property would be to allow the recipient to make security determinations at the time the message is retrieved.

**Broadcast messages and dead-drops.** Chaum discusses a traffic-analysis prevention method in which all reply mail in the anonymous mail system is sent to all possible recipients. A less invasive optimization has already been implemented in the form of Usenet mail drops [10]: an anonymous remailer can deliver mail to a newsgroup, rather than to an email recipient. Such mail can be encrypted to a recipient's private key, and left for her to collect from the newsgroup. If recipients use the same newsgroup and behave identically (for instance, by downloading the entire set of newsgroup messages daily), the possible statistical attacks on direct mail delivery of reply messages to individual email addresses are avoided. This solution also removes the necessity for reply-blocks, as the drop location can be established upon out-of-band.

Of course, this "send everything everywhere" approach suffers massive scalability problems. As the number of users in the system increases, each user's bandwidth requirements become prohibitive. Users are thus encouraged to "cheat" and only download sections of the newsgroup that they are sure contain their messages, or not download on days that they do not expect messages. This allows an attacker to gather information about messages in which an individual has interest, and provides a way to attack the security of the system [2].

## 3 The Pynchon Gate Design

We present a design framework for the Pynchon Gate. A detailed implementation specification can be found in [45].

### 3.1 Overview and Rationale

The Pynchon Gate is a group of servers that provide anonymous message retrieval capabilities (see Figure 1). A nym server receives messages for different pseudonym accounts via email.[6] Once every "cycle" (e.g., 24 hours), the nym server passes these messages to a collator, which batches them into an indexed "bucket pool," and and passes these pools to each independently operated distributor node in the network. Each pseudonym holder then makes a series of requests to $k$ chosen distributor nodes, enabling her to receive her messages without the individual distributors determining the pseudonym being requested. The protocol used is resistant to collusion: even if the adversary controls $(k-1)$ of the chosen distributors, the adversary cannot link the user to her pseudonyms.

[6] The servers could also receive messages through any suitable medium for message transfer, such as "instant message" systems [24]. We require a forward anonymity protocol to allow the nym holder to communicate with the nym server, so at a minimum the nym server must be able to receive email in addition to any optional support for other protocols.

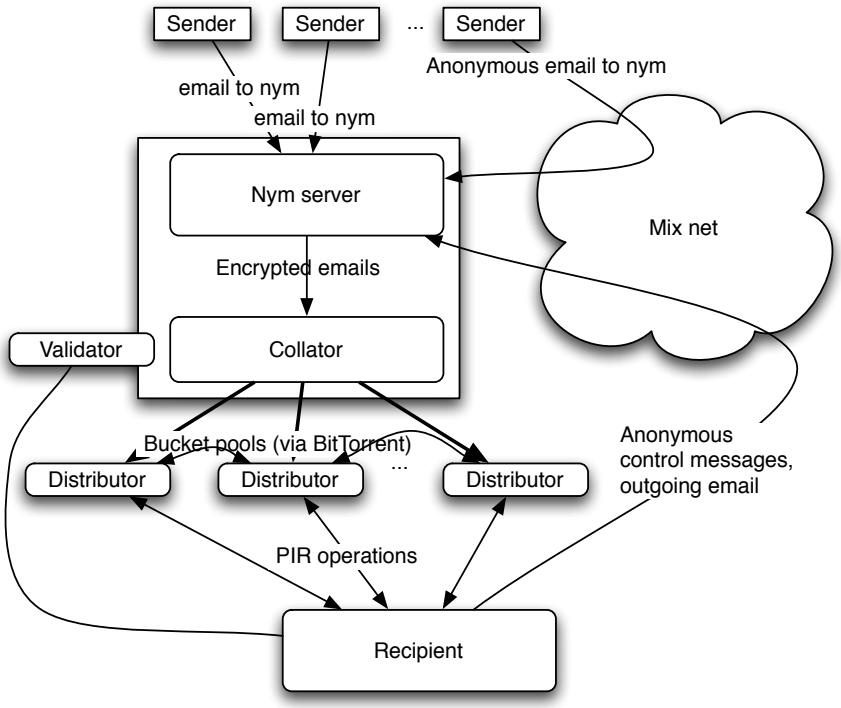


**Fig. 1.** The Pynchon Gate Architecture

This distributed-trust PIR-based message retrieval system lets us keep the reliability, and security of the "send everything everywhere" method, while bringing the system into the realm of feasibility for users with limited resources.

We discuss the components of the Pynchon Gate architecture below.

### 3.2 The Nym Server

The public-facing side of the Pynchon Gate consists of a nym server that sends and receives pseudonymous email. The nym server itself provides no sender anonymity; rather, it relies on existing mix networks [22, 52]. The nym server is visible to external email correspondents, and receives messages for the nym owners at their specified email addresses.

Nym servers manage email accounts for pseudonyms. For each pseudonym, the nym server stores a *long-term public key* used by the nym holder to encrypt and authenticate outgoing email and administrative messages. Similarly, nym server stores a *short-term shared secret* for each account, used to encrypt messages to the nym holder. This secret can be reset by the nym holder after account creation.

The shared secret is updated every cycle, such that, if $S[i]$ is the shared secret in a given cycle $i$, then $S[i+1] = H(S[i]|\texttt{"NEXT CYCLE"})$, where $H(\cdot)$ is a cryptographic hash and | denotes concatenation. From each $S[i]$, the nymserver derives a set of sub-secrets for individual messages received that cycle. The $j$'th sub-secret on day i is $\text{Subkey}(j+1, i) = H(\text{Subkey}(j, i)|\texttt{"NEXT SECRET"})$, with $\text{Subkey}(0, i) = H(S[i]|\texttt{"NEXT SECRET"})$.

Once it no longer needs a shared secret or a given subkey, the nym server drops it immediately, to limit the impact of key compromise (at the server or client) and improve forward security. We use a separate chain of keys for each cycle so that it is easier for a user to resynchronize after missing a few cycles.

These subkeys are used to encrypt and identify the messages received on day $i$. When the $j$'th message for the nym is received, the nym server compresses it, encrypts it with the symmetric key $H(\text{Subkey}(j, i)|\texttt{"KEY"})$, and prefixes it with the opaque identifier $H(\text{Subkey}(j, i)|\texttt{"ID"})$. By deriving the message keys and identifiers in this way, we allow users to store keys and identifiers for pending messages without risking exposure of messages encrypted in the same cycle.

Finally, the nymserver also generates an different independent identifier for each user every cycle: $\text{UserID}[i] = H(S[i]|\texttt{"USER ID"})$.

### 3.3 The Collator

At the end of each cycle, the nym server passes messages to the *collator*, which typically resides on the same physical server. The collator organizes all previously unretrieved messages into a three-level structure, consisting of a *meta-index*, a set of fixed-size *index buckets*, and a set of fixed-size *message buckets*.

Each user's messages are stored in a different set of message buckets, ordered by UserID. The index buckets contain, for each UserID (in order), the first message bucket containing that user's messages, and a digest of that bucket. Finally, the meta-index lists, for each index bucket, the first and last UserID in that bucket, and a digest of that bucket (see Figure 2). The index buckets and the message buckets together comprise the cycle's "bucket pool." To ensure integrity, each bucket contains a hash of the next.

The metaindex is signed with the collator's private key, along with the index of the cycle to which it applies.

To prevent an attacker from flooding a nym and observing which user receives a large volume of traffic, each nym has a maximum number of message buckets that may be filled in a given cycle. If there are more pending messages than will fit in the nym's buckets, the collator defers some for a later cycle.[7] Because the encrypted messages are prefixed with $H(\text{Subkey}(j, i)|\texttt{"ID"})$, the user can tell which key to use for messages that are delivered out of order.

[7] As an extension to save bandwidth and prevent denial of service attacks, the nym server can build a special "summary message" containing the headers of pending emails, and their opaque identifiers, and include this message in the user's message bucket. The user can then send a signed control message to the nym server requesting that unwanted emails be deleted and desired ones given priority.

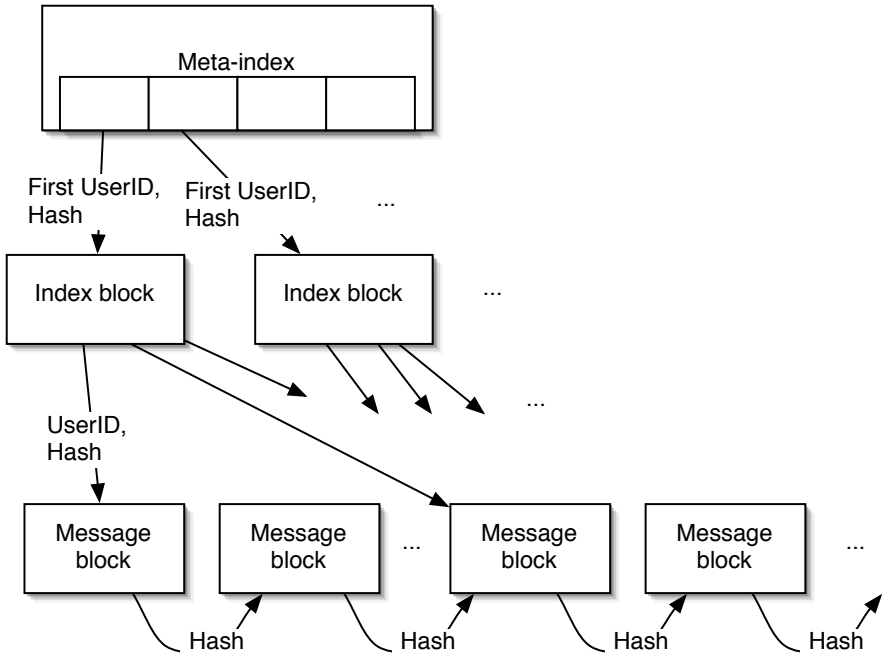


**Fig. 2.** The meta-index and bucket pool

The collator should use S-expressions [56] for representing the structure of the database. More details on the exact data format used can be found in the byte-level spec [45].

### 3.4 Distributors and clients

Once the collator is done, it relays the signed meta-index, and the entire bucket pool, to a set of independently operated *distributor nodes*. (This data should be transmitted using a bandwidth-sparing protocol such as BitTorrent [14], so that the collator does not need to send the entire pool to each distributor.)

At this point, clients can download their messages for the cycle. First, a given client downloads the meta-index from a randomly chosen distributor, and verifies its signature. The client then computes its UserID for the day, and uses the meta-index to tell which index bucket will contain an entry for that UserID.

The client then uses the PIR protocol described in Section 4 to retrieve the correct index bucket, checks that the bucket's digest is as expected, and uses the index bucket to learn which message buckets will contain the client's messages. The client downloads these buckets with using PynGP, and checks their digests. If the client has received fewer buckets this cycle than her maximum, she performs extra PIR operations up to that maximum, to prevent an observer from learning how many messages she has received.

Depending on the length of the cycle, clients may not be able to download messages every cycle. Therefore, distributors must retain meta-indexes and bucket pools for a reasonable window of time, to be sure that all clients have time to download their messages.

The message integrity and tagging attack protection mechanism described in Section 5.1 also ensures that malicious distributors will be discovered if they attempt to execute denial of service attacks by dropping or garbling messages.

Since it is not necessary for every distributor to be operational or honest at the given point that a client wishes to retrieve mail, the system handles distributor node failure in a graceful manner.

## 4 The PIR Protocol

We present the PIR protocol used in this system in two parts. In Section 4.1 we describe a simplified version of the protocol that does not consider denial of service attacks as part of its threat model. In Section 4.2 we present the extensions to the first protocol, first proposed in [59], which permit probabilistic detection of misbehaving servers.

### 4.1 PynGP 1.0: The Byzantine-naive PIR protocol

Using PynGP 1.0, the user submits a PIR query to $\ell$ distributors, and his message is returned with none of the distributors able to deduce any information about the user's query unless all $\ell$ distributors collude. This form of PIR is referred to as an **information-theoretic $(\ell - 1)$-private $\ell$-server PIR** protocol. The protocol runs as follows: after choosing distributors, the client establishes an encrypted connection to each (e.g., using TLS [26]). These connections must be unidirectionally authenticated to prevent man-in-the-middle attacks, and can be made sequentially or in parallel.

The client sends a different "random-looking" bit vector $\vec{\nu}_{s\beta}$ to each distributor $s$ for each message block $\beta$ to be retrieved. Each bit vector has a length equal to the number of message blocks in the database. Each distributor $s$ then computes $R(\vec{\nu}_{s\beta})$ as the exclusive-OR of all message blocks whose positions are set to 1 in $\vec{\nu}_{s\beta}$. The resulting value is then returned to the client.

Thus, in order to retrieve the $\beta$'th message block, the client need only choose the values of $\vec{\nu}_{s\beta}$ such that when all $\vec{\nu}_{s\beta}$ are XORed together, all values are 0 at every position except $\beta$. (For security, $\ell - 1$ of the vectors should be generated randomly, and the bit vectors should be sent in a random order so that the $\ell$'th, specially-crafted vector cannot be distinguished.) When the client receives the corresponding $R(\vec{\nu}_{s\beta})$ values, she can XOR them to compute the message block's contents.

As an optimization, a client may send $k - 1$ of the distributors a key for a stream cipher instead of a bit vector. The distributors can use the stream in place of the vector [4, 5]; only one still needs to receive a full vector.[8]

[8] Obviously, this introduces a reliance on cryptographic security with regard to the stream cipher, and the PIR protocol can no longer be considered fully "information-

### 4.2 PynGP 2.0: Byzantine server detection extensions for PynGP

We have modified PynGP 1.0 only as much as necessary to address its known security flaws, originally reported in [60]. The revised version of the protocol retains the same security properties set forth in the original design paper. We address the issue of Byzantine nodes [43] by introducing a *cut-and-choose* methodology [55, 9]. To support this addition, we modify the query algorithm and add a response validation algorithm (to be run if the reconstruction algorithm fails) at the cost of trivial computation expense and a doubling in the bandwidth needed to perform queries of the database.[9] Finally, we introduce a new component in the Pynchon Gate architecture, known as the *validator*.

Thus, PynGP 2.0 is identical to PynGP 1.0 as described in Section 4.1, with the following modifications to the protocol:

Each time the protocol runs, the client prepares two sets of bit vectors to send to the chosen distributors. The first set, $\{\vec{\alpha}\}$, is used to obtain the private mailbox data via the PIR protocol; the second set, $\{\vec{\eta}\}$, is used to challenge the honesty of each distributor.

At the step in the PynGP 1.0 protocol where the client would transmit the "random-looking" bit vector to each distributor, the client submits two "random-looking" bit vectors instead, one from $\{\vec{\alpha}\}$ and one from $\{\vec{\eta}\}$, transmitted in a random order.

Upon receiving these bit vectors, the distributor performs the operations as described in Section 4.1, and then returns two responses, in the same order which the requests were received (or otherwise in such a manner that the responses are linkable to the requests which generated them). The client caches the response for the $\{\vec{\eta}\}$ request, then performs the PIR operation as previously described using the $\{\vec{\alpha}\}$ results from all distributors.

### 4.3 An additional architecture component: The Validator

We now introduce a new component in the Pynchon Gate architecture, known as the *validator*. This component is essentially a distributor, except that it only exists to confirm that the other distributors are not Byzantine. This specialized distributor validates the "cut-and-chosen" responses as being correct, deterring the operation of Byzantine nodes and probabilistically uncovering them should they exist.

To ensure that additional trust is not required of this new component to the system, the validator **must** be operated by the same entity who operates the collator. The operator of the collator is already empowered to perform a denial

theoretically secure." However, the benefits of this optimization can be argued to justify the potential reduction in security.

[9] While this is indeed a significant increase in system overhead, it is still feasible, especially given the Pynchon Gate's resource tuning properties which permit a linear trade-off between bandwidth and storage, adjusted simply by changing the size of the message blocks stored in the database. Doubling the size of the message blocks halves the bandwidth necessary to perform a query.

of service attack by simply unplugging the power cord of the server running the collator. Ergo, the balance of power in this distributed trust system is maintained by placing the validator (whose operator could also force a denial of service attack, though not as easily) under the control of the same entity.[10] Note that communication with the validator occurs over an encrypted link, just as with normal distributors.

**Auditing the distributors.** The requests comprising set $\{\vec{\eta}\}$ are crafted such that they return a specific *validation block* when the PIR algorithm is performed. Under normal circumstances, the contents of this block are of no interest to the user. The individual responses are cached by the client, along with the corresponding request that was sent to the distributor to generate them as well as an identifier for the distributor which returned the responses. To verify that a distributor is not attempting to behave in a Byzantine manner, the same bit vectors in $\{\vec{\eta}\}$ that had been submitted to each distributor can subsequently be submitted to the validator, and the validator should return a response identical to that which the original distributor returned for each request. (The entire $\{\vec{\eta}\}$ needs to be submitted, as there may be multiple Byzantine servers acting simultaneously.) Should the validator return a response that differs from the one received by the client from a given distributor, that distributor should be suspected of being Byzantine.[11]

**Auditing the Validator.** The addition of the validator component does raise the concern that a corrupt nym-server/collator/validator coalition may attempt to mount an attack on a user's anonymity by systematically framing honest distributors as Byzantine nodes so that the user selects only nodes operated by the coalition. As one of the main premises behind the security of the Pynchon Gate design is that the nym-server operator not be trusted to preserve the user's anonymity, a way to confirm that the validator is honest is needed. This confirmation procedure is simple:

If the $\{\vec{\eta}\}$ responses from the distributors differ from the $\{\vec{\eta}\}$ responses from the validator, the client should first attempt to verify the correctness of the $\{\vec{\eta}\}$ response by performing the PIR protocol and comparing the result to the known validation block. If the correct block is returned, there is nothing to be learned by querying the validator. If the responses to the $\{\vec{\eta}\}$ requests yield an incorrect

[10] The validator is never provided the contents of $\{\vec{\alpha}\}$ queries, since the validator, collator, and nym server are *not* considered trusted with regard to a user's privacy, and knowledge of the contents of $\{\vec{\alpha}\}$ queries (or responses) could provide information about the user's identity.

[11] Strictly speaking, the $\{\vec{\eta}\}$ vectors should always be sent to the validator, regardless of the outcome of the PIR operation using the $\{\vec{\alpha}\}$ results, to guard against the scenario where an additional adversary not in collusion with the Byzantine server might learn additional information about the state of the network. However, this level of paranoia may not be affordable until bandwidth and computation become less expensive.

validation block, the presence of at least one Byzantine distributor is verified. The client then proceeds as described above, submitting each $\{\vec{\eta}\}$ query to the distributor *one query at a time* and recording the results. When a differing result is received for a given query, it should be swapped in for the original result, and the PIR protocol performed. The substitution of the validator's response for the original response should yield the correct validation block if the validator is honest. In cases where the validator's responses differ for more than one query, this challenge should be performed for each differing response both individually, and as a whole.

## 5 Attacks against Pseudonymity Systems

We present the common types of attacks against pseudonymity systems, and present novel analysis of the effectiveness of one kind of traffic analysis against the most popular currently deployed design. We also highlight security concerns specific to the Pynchon Gate design.

### 5.1 Attack categories

Most known attacks on pseudonymity systems fall into one of the following categories.

**Legal and hacking attacks.** Attackers may attempt to coerce the operators of pseudonymity systems through lawsuits or other means [49, 62, 40, 38, 32], or may attempt to surreptitiously obtain information about nym holders.

We limit these effects of these attacks by greedily encrypting incoming messages and deleting encryption keys. All sensitive data is deleted as the bucket pool is generated, ensuring that the collator has as little information useful to an attacker as possible.

Without dynamic key rotation it would be trivial for an attacker to archive all data sent to distributors, and then at some later time obtain the nym's collator address and key from the nym server though an active attack on those components. The attacker could then read all messages that nym has ever received. In the interest of retaining little information for an attacker, implementations should discard old secrets *as soon as they are no longer needed.* Thus, at the start of each cycle $i$, a nymserver should derive $S[i+1]$, UserID$[i]$, and Subkey$(0, i)$, and immediately discard $S[i]$. After using each Subkey$(j, i)$, the nymserver should calculate Subkey$(j+1, i)$ and discard Subkey$(j, i)$. After each cycle, the nymserver should discard the last Subkey$(j, i)$, and UserID$[i]$.

**Mix attacks.** Since we rely on mix networks, we must be concerned with attacks against them. Furthermore, reply-block-based nym server systems require additional security properties that normal mix-net systems may not have [23].

The Pynchon Gate uses mix-nets for forward message delivery only. Attacks that do not work against a mix-net in normal forward-delivery mode will not impact the Pynchon Gate.

**Man-in-the-middle attacks.** An attacker able to pose as a user's chosen distributors could trivially see all $k$ PIR requests. We use TLS authentication to prevent this attack.

**Replay attacks.** An attacker capable of monitoring the communications network may attempt to obtain information about nym holders by comparing network and user behavior when a given message or packet is transmitted multiple times.

The Pynchon Gate uses TLS when communicating between components and the client, so that data is encrypted with a short-lived session key. The topology of the Pynchon Gate infrastructure further eliminates areas of potential replay attack risk.

**Tagging and known-cleartext attacks.** An attacker may alter a message, or observe the cleartext of a message, so that he may be able to later link an input message with a given output retrieved by the nym holder.

The Pynchon Gate's message and link encryption prevents an attacker from observing the cleartext of a message. Tagging attacks are ineffective, as TLS protects data integrity on the wire. The metaindex provides the client with the hash of the index bucket. Each message bucket provides the hash of the next message bucket, and with this information, the client can verify the integrity of information downloaded from distributors and respond to garbled data without leaking information about which bucket it was requesting.[12]

**"*Who am I?*" attack.** An attacker may send messages intended for nym Alice to nym Bob instead, so that if "Alice" responds, the attacker will know they are the same nym holder [20].

This attack relies primarily upon the ability to link one nym, $Alice_1$, with another nym, $Alice_2$, by sending a message encrypted to $Alice_1$'s to $Alice_2$'s address. The Pynchon Gate avoids this, though a similar social engineering attack may be performed if the nym holder is using a separate message encryption protocol such as PGP [11]. More research needs to be done to improve the area of privacy-preserving human-computer interaction [57, 63].

**Usage pattern and intersection attacks.** An attacker may analyze network usage and anonymity set members over time to sub-divide anonymity sets such that a given user is identified. In addition to passive observation of the network, there are a number of active attacks. For example, an attacker could flood a nym to observe a corresponding increase in traffic by the recipient.

Users of the Pynchon Gate select distributors from which to receive data at random, each time the nym holder retrieves her messages. Unlike systems where

[12] If a client *does* notice a corrupt bucket, it should not re-attempt the PIR operation until it has received all buckets, to avoid leaking which bucket it was reading through the timing of its response.

the pseudonym infrastructure initiates the delivery of messages, the Pynchon Gate Client initiates the retrieval of messages, and thus message retrieval cannot be correlated to a given nym by a malicious sender.

Message buckets are of a fixed size, to protect against active flooding attacks [61] as well as simple usage pattern analysis. If the volume of messages is too great to fit in a users' buckets, delivery continues by *trickling* the pending messages to the distributors over the next several tree distributions. To prevent denial of service attacks, users can selectively retrieve or delete excess messages.[13]

A hash tree of variable depth would risk leaking usage information about nym-holders. The Pynchon Gate uses a tree of a fixed depth.

Since the time between sending a message and receiving a reply may leak information about the nym holder, traffic from the client to the distributors is regulated by the client, which queries the distributors only at given intervals. To thwart active attacks against the distributor targeting a specific client, clients should choose these intervals randomly.

### 5.2 Statistical disclosure against reply-block-based nym servers

Nym servers based on reply blocks (discussed in Section 2.1 above) are currently the most popular option for receiving messages pseudonymously. Nevertheless, they are especially vulnerable to end-to-end traffic analysis.

Suppose an adversary is eavesdropping on the nym server, and on all recipients. The attacker wants to know which user (call her Alice) is associated with a given pseudonym (say, `nym33`). The adversary can mount an *intersection attack*, by noticing that Alice receives more messages, on average, after the nym server has received a message for `nym33` than when it has not.[14] Over time, the adversary will notice that this correlation holds for Alice but not for other users, and deduce that Alice is likely associated with `nym33`.

Recent work [19, 47] has studied an implementation of these intersection attacks called *statistical disclosure*, where an attacker compares network behavior when Alice has sent to network when she is absent in order to link an *anonymous* sender Alice to her regular recipients $\text{Bob}_1 ... \text{Bob}_N$. Against *pseudonymous* recipients, however, these attacks are far easier: in the anonymity case, many senders may send to any given recipient $\text{Bob}_i$, but with pseudonymous delivery, only one user sends or receives messages for a given pseudonym.

To examine this effect, we ran a version of the attack simulations described in [47], assuming a single target pseudonym and $N$ non-target pseudonyms providing cover. In order to make the attack as difficult as possible (and thus establish an upper bound on security), we assume that users behave identically: they

[13] If a client will be retrieving large amounts of data on a regular basis, this method will not work. Instead, the client should at account creation time request a sufficient number of buckets to receive all data destined to it. Pending data queued on the collator should be expired after a reasonable amount of time.

[14] This task is especially easy if the adversary can distinguish reply messages from non-reply messages, as is possible with Type I remailers.

receive messages with equal probability according to independent geometric distributions in each unit of time (receiving no messages with probability $1 - P_M$); they use identical reply blocks with path length $\ell$ through mixes in a steady state that delay each message each round with probability $P_D$.

We ran the simulated attack with different values for $P_M$, $P_D$, and $\ell$, against a nym server with $N = 2^{16}$ active pseudonymous users. (This is probably an overestimate of the number of users on typical nymserver today [49].) We performed 100 trials for each set of parameters. In the worst case (for the nym holder), when $P_M = 0.5, \ell = 1, P_D = 0.1$, the lack of mix-net delay variance allowed the simulated attacker to guess the user's identity correctly after the user received an average of only 37 messages. In the best simulated case ($P_M = 0.5, P_D = 0.9, \ell = 4$), the user received an average of only 1775 messages before the attacker guessed correctly. For an active user, this is well within a month's expected traffic.

Although there are ways to use dummy traffic to resist statistical disclosure attacks, these difficult to implement perfectly in practice (due to outages) and even slight imperfections render users vulnerable to attack [47].

### 5.3 System availability attacks

Attacks against the security of an anonymous communication system are not the only concern designers of such systems must address. It may simply be enough for an attacker to render such systems unusable by reducing the reliability or uptime of the system to such a degree that users seek other, possibly weaker or compromised, means of communication. We do not address all possible denial of service attacks in this paper, as most are commonly known in the information security community, and are either attacks against lower-layers on the network protocols upon which the Pynchon Gate system operates, or commonly-generalizable techniques for denying access to a given network service. More information on these general attacks can be found in RFC 4732 [39]. Instead, we limit our discussion of this topic to attacks specific to PynGP or those which must be performed by malicious servers operating as part of a Pynchon Gate deployment.

**Byzantine server protection.** In a distributed-trust anonymity system such as the Pynchon Gate, there exists the possibility that some servers may be *Byzantine*, i.e., they may behave incorrectly, either due to intentional malice or simple error.[15] In the case of the Pynchon Gate, the Byzantine behavior we are concerned with is an incorrect response to a PIR query of a distributor's database.

All $n$ distributors in the system have the exact same copy of the database, and the system is designed such that any attempt by a Byzantine server to modify its response to the PIR query will be detected by the user when he verifies the root of the hash tree. This is crucial to preserving the anonymity properties of the system, for if an attacker can alter a message or observe the cleartext of a

[15] This concern is present in many other anonymity systems, including Chaumian mixnets [12, 52, 22] and systems built on top of them [49, 46].

message, he may potentially be able to later link an input message with a given output retrieved by the nym holder.

The Pynchon Gate's message and link encryption prevents an attacker from observing the cleartext of a message. Active attacks that are dependent upon the attacker's ability to alter some of the data being transmitted to the user such that the attacker may later link the user to his pseudonym based either on a variance in the user's response to altered versus unaltered data, or by simply recognizing the product of the altered data as it is processed by the system (collectively known as *tagging attacks* [30]) are ineffective, as TLS protects data integrity on the wire. Thus, any tagging attacks an attacker wished to attempt against a user would have to occur through the use of a corrupt distributor. To protect against the case where a distributor provides (intentionally or otherwise) an incorrect response to the PIR query, the client verifies that the hash of the message block it has received can be authenticated through the hash tree with the verified hash root.

**Byzantine server detection.** It is not enough, however, for the Pynchon client to simply detect when a Byzantine action has occurred; ideally the user would have a way of learning the identity of the Byzantine server, to avoid relying upon it for future requests.

The hash tree verification system does not prevent a corrupt distributor from creating, either through malice or error, a denial of service attack on the system by responding with incorrect data to a client's query. While the client will detect that the message block is invalid after performing the final step of the PynGP PIR protocol, and thus can conclude that *some* server was Byzantine, the client cannot determine *which* server or servers returned the incorrect response. The client cannot safely pass the message block contents (assuming they consist of anything other than garbage) to the user, lest the user's anonymity be potentially compromised.

Furthermore, if attacks on portions of the pseudonymity infrastructure affect some users differently than others, an attacker may exploit such attacks on components of the system to facilitate an intersection attack against a user of the system as a whole [28]. In the Pynchon Gate, if a Byzantine distributor selectively performed denial of service attacks against certain users by returning garbage results to their queries, but correctly responded to other users' queries, the attacker would increase his chances of learning the identity of certain users, based on which users responded to messages that were successfully delivered.[16] In other cases, a passive adversary could observe the actions of Byzantine servers not under his control (and perhaps not even behaving maliciously, but simply

[16] This type of attack is present (in a slightly different form) in non-PIR-based nym server systems as well. For instance, in a reply-block system, an attacker could disable certain mixes and observe which nyms ceased receiving traffic. If the nym holder has a fixed-route reply-block, this would enable the attacker to identify the mixes used in the nym holder's reply-block path, and increase his chances of successfully linking the nym with the nym holder's true name [61].

incorrectly) to help facilitate intersection attacks [64]. Additionally, if a user cannot know with confidence which server is behaving in a Byzantine fashion, she is more likely to change the nodes she uses on a regular basis, both increasing her exposure to long-term intersection attacks and increasing the probability of selecting a server-set that consists of nodes operated entirely by a single adversary.

We address this potential attack through the cut-and-choose protocol described above, in Section 4.2.

**Probability of Byzantine detection.** As proposed, PynGP 2.0 gives a Byzantine server a 50% chance of being discovered each time it attempts to behave in a Byzantine manner. That threshold can be increased at the expense of greater bandwidth overhead; however, we feel that a 50% detection rate is sufficient to deter this sort of attack, due to the inherent reputation system involved with the distributor network.[17] This probability of detection is based on the assumption that the Byzantine server considers it acceptable that its Byzantine action may be *ineffective*. If the server wishes to *guarantee* a successful Byzantine operation, it can do so by providing Byzantine answers to all the client's requests, but the probability of detection in that case is 100%.

One Byzantine action by a distributor *verifiable as such* to the collator, validator, or nym server operator should be sufficient to blacklist a Byzantine distributor. Care must be taken to ensure that an attacker does not frame an honest server as Byzantine to have it blacklisted. Multiple reports identifying a given server as Byzantine might simply indicate a Sybil attack being performed to achieve the blacklisting of an innocent server.

### 5.4 Other security concerns

The information used for authentication of the system components (such as the certificates and the hash tree root and metaindex) must be published widely to prevent either the collator or any of the distributors from attacking clients by tricking them into thinking that the hash root of the tree is something other than what all of the other clients know it to be. Distributors should do basic sanity checks, such as verifying tree integrity. The distributors should also send audit messages of their own to verify that the messages correctly appear in the system. Finally, clients should make sure that each of the distributors they use agree about the value of the hash root.

## 6 System Performance, Scalability and Optimizations

In this protocol, the size of requests is linearly proportional to the total number of messages and the size of responses is the bucket size. If one or the other of these

[17] A Byzantine server's chances of successfully providing a Byzantine response of unidentified origin decreases in an manner inversely proportional to its probabilty of detection.

values is large enough to cause scaling problems, then the collator can trade off bucket size for bit field size by doubling the bucket size, which halves the bit field size. With this approach, if the number of buckets becomes very large, then the message size rises proportionately to the square root of the number of buckets. This scales well, although it may necessitate multiple collators if the number of buckets gets extremely large. (Note that while different collators may share the same distribution nodes for architecture or resource reasons, their anonymity sets would remain separate.)

The PIR algorithm in this paper does not have optimal asymptotic performance, especially in bandwidth. We present it nonetheless because it is easy to explain, implement, and analyze, and offers sufficient scalability to be useful.

Another potential bottleneck lies in the fact that distributors have to perform a linear scan of the entire bucket pool in order to fulfill a request. However, they can use a single linear pass to compute the results of many requests in parallel. Thus, a distributor can fulfill a large number of requests at once, though the latency to answer those requests is limited by the total size of the bucket pool, and the throughput to the distributor's hard drive (unless the bucket pool fits in RAM). Also, by performing continuous linear scans of the entire database, the distributor can begin computing the result of a request at any point during the scan, finishing when the next scan returns to that same point. Thus, the latency is exactly the time of one full scan.

Latency in the PIR protocol can be reduced by allowing the client to retrieve all its buckets at once with a single execution of the PIR protocol on a column vector of all its messages. This approach makes the whole database be the same size for each publication period (given the same user set) which could waste bandwidth and storage space, though unused sections could be optimized out, at the cost of requiring some compression to distribute everything to the distributors efficiently. Other similar tradeoffs between latency, bandwidth, storage, and computation also exist.

### 6.1 Performance comparison between PynGP 1.0 and PynGP 2.0

As previously stated, PynGP 2.0 has a higher performance cost than its predecessor. We have attempted to strike a balance between security and excessive resource consumption when the security issues are unlikely to be problematic, or the resource requirements too great to qualify the protocol as "deployable". Areas where security could be increased, if performance cost were no object, include the addition of more than a single challenge-tor set to decrease the probability of a Byzantine server successfully avoiding detection.[18] Also, as previously noted, it would be ideal, were it affordable, for the challenge-vectors to be validated every time a PynGP protocol run was performed. This is likely cost-ineffective, though, given that knowledge of the user's failure to validate the hash root is unlikely to give an adversary any significant advantage, let alone lead to a user-level privacy

[18] For $n$ challenge-vector sets, the probability of avoiding detection is $\frac{1}{n+1}$, and consequently, the probability of failing to cause a Byzantine failure is $\frac{n}{n+1}$.

vulnerability. It is far more important that all clients in the system behave according to the same policy in this regard. Also, if the number of challenge-vector sets is increased, the cost of validation increases proportionally.

The resource requirements for the validator must be more fully investigated, and will not truly be known until the system is tested in a live environment with actual Byzantine nodes. It is conceivable that the validator might best be deployed as multiple load-balanced servers, should the level of resource consumption warrant it. It is also conceivable that there may be few to no challenge validations necessary under normal conditions. When challenge validations are required, however, the bandwidth cost per challenge is non-trivial. (The validator receives $(r \cdot \ell)$ bits from a given client, and returns $(n \cdot \ell)$ bytes, where $r$ is the number of message blocks (and thus the length of any given bit vector) and $n$ is the size of a message block.) Though a client must validate its challenge-set in its entirety, the client should avoid sending the entire set at once, however. It is advisable to submit each element of the challenge-set to the validator successively, so that the validator can impose rate-limiting on the incoming validation requests to cope with instances of sudden high load.

The capability to gracefully address the issue of Byzantine nodes is itself an anti-Byzantine measure; one possible approach when a Pynchon Gate system is first deployed might be to use PynGP 1.0 until evidence of the existence of Byzantine servers in the system is observed. It is possible that merely having PynGP 2.0 implemented in the software and able to be enabled instantly could serve as a deterrent to a casual attacker whose goal is to deny service to the system, as the only effect such an attacker would have by deploying a Byzantine node would be to increase the network communication and verifier computation costs to a level that we have already deemed acceptable.

## 7 Conclusions

We have presented a system for anonymous message retrieval that provides stronger anonymity assurance and more robustness than other theorized or deployed high-latency pseudonymous message retrieval systems. Our system resists traffic analysis better than current deployed systems, and offers convenient scalability options. Additionally, it discourages Byzantine servers by offering a technique for detecting misbehaving servers without compromising or weakening the system's security.

We have proposed a client protocol that does not rely upon an obtrusive user interface. Much work remains in the field of effective user interface design for privacy and anonymity systems, particularly when the privacy component is viewed by the user as optional.

## 8 Acknowledgments

Numerous individuals have contributed to the Pynchon Gate project in one form or another over the years. We owe a debt of gratitude to all of them, especially

Elena Andreeva, Sonia Araña, Adam Back, Nikita Borisov, David Chaum, Roger Dingledine, Orr Dunkelman, Lucky Green, Markulf Kohlweiss, Ben Laurie, Jim McCoy, David Molnar, Russell O'Connor, Peter Palfrader, Meredith L. Patterson, Adam Shostack, Bryce Wilcox-O'Hearn, Shar van Egmond, and Julia Wolf, and many others.

Additionally, we thank the Free Haven Project for graciously providing mailing list and code repository resources.

The work of Bart Preneel and Len Sassaman was supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government and by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy). Len Sassaman's work was supported in part by the EU within the PRIME Project under contract IST-2002-507591.



## References

1. Alessandro Acquisti, Roger Dingledine, and Paul Syverson. On the economics of anonymity. In Rebecca N. Wright, editor, *Financial Cryptography*. Springer-Verlag, LNCS 2742, 2003.
2. Anonymous. alt.anonymous.messages considered harmful. Mailing list post, November 1995. `http://cypherpunks.venona.com/date/1995/11/msg00089.html`.
3. Andre Bacard. FAQ for the ALPHA.C2.ORG Remailer. Usenet post, October 1995. `http://groups.google.com/groups?selm=4q4tsr%248ui%40crl14.crl.com&output=gplain`.
4. Adam Back. Personal communication, April 2003.
5. Oliver Berthold, Sebastian Clauß, Stefan Köpsell, and Andreas Pfitzmann. Efficiency improvements of the private message service. In Ira S. Moskowitz, editor, *Proceedings of Information Hiding Workshop (IH 2001)*, pages 112–125. Springer-Verlag, LNCS 2137, April 2001.
6. Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 115–129. Springer-Verlag, LNCS 2009, July 2000.
7. Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. The disadvantages of free MIX routes and how to overcome them. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 30–45. Springer-Verlag, LNCS 2009, July 2000.
8. Philippe Boucher, Adam Shostack, and Ian Goldberg. Freedom systems 2.0 architecture. White paper, Zero Knowledge Systems, Inc., December 2000.

9. Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988.
10. Rick Busdiecker. Message pool: alt.anonymous.messages. Mailing list post, August 1994. `http://cypherpunks.venona.com/date/1994/08/msg00185.html`.
11. J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. OpenPGP Message Format. Request for Comments: 2440, November 1998.
12. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1981.
13. Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *IEEE Symposium on Foundations of Computer Science*, pages 41–50, 1995.
14. Bram Cohen. Incentives Build Robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, May 2003. `http://www.sims.berkeley.edu/research/conferences/p2pecon/papers/s4-cohen.pdf`.
15. David A. Cooper and Kenneth P. Birman. Preserving privacy in a network of mobile computers. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, May 1995.
16. Lance Cottrell. Mixmaster and remailer attacks. `http://www.obscura.com/~loki/remailer/remailer-essay.html`.
17. Lance Cottrell. Re: Strengthening remailer protocols. Mailing list post, September 1996. `http://cypherpunks.venona.com/date/1996/09/msg00730.html`.
18. Wei Dai. Pipenet 1.1. Usenet post, August 1996. `http://www.eskimo.com/~weidai/pipenet.txt`.
19. George Danezis. Statistical disclosure attacks: Traffic confirmation in open environments. In Gritzalis, Vimercati, Samarati, and Katsikas, editors, *Proceedings of Security and Privacy in the Age of Uncertainty, (SEC2003)*, pages 421–426, Athens, May 2003. IFIP TC11, Kluwer.
20. George Danezis. *Better Anonymous Communications*. PhD thesis, University of Cambridge, 2004.
21. George Danezis. The traffic analysis of continuous-time mixes. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*, LNCS, May 2004.
22. George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.
23. George Danezis and Ben Laurie. Minx: A simple and efficient anonymous packet format. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2004)*, Washington, DC, USA, October 2004.
24. M. Day, S. Aggarwal, G. Mohr, and J. Vincent. Instant Messaging / Presence Protocol Requirements. Request for Comments: 2779, February 2000.
25. Claudia Díaz, Len Sassaman, and Evelyne Dewitte. Comparison between two practical mix designs. In *Proceedings of 9th European Symposium on Research in Computer Security (ESORICS)*, LNCS, France, September 2004.
26. T. Dierks and C. Allen. The TLS Protocol. Request for Comments: 2246, January 1999.
27. Roger Dingledine, Michael J. Freedman, and David Molnar. The Free Haven Project: Distributed anonymous storage service. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, July 2000.
28. Roger Dingledine and Nick Mathewson. Anonymity loves company: Usability and the network effect. In *Proceedings of the Fifth Workshop on the Economics of Information Security (WEIS 2006)*, Cambridge, UK, June 2006.

29. Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.
30. Roger Dingledine and Len Sassaman. Attacks on Anonymity Systems: Theory and Practice. In *Black Hat USA 2003 Briefings*, Las Vegas, NV, USA, July 2003.
31. Hal Finney. New remailer... Mailing list post, October 1992. `http://cypherpunks.venona.com/date/1992/10/msg00082.html`.
32. Independent Centre for Privacy Protection. AN.ON still guarantees anonymity. `http://www.datenschutzzentrum.de/material/themen/presse/anonip_e.htm`, 2003.
33. Ian Goldberg. *A Pseudonymous Communications Infrastructure for the Internet*. PhD thesis, UC Berkeley, December 2000.
34. Ian Goldberg. Privacy-enhancing technologies for the Internet, II: Five years later. In Roger Dingledine and Paul Syverson, editors, *Proceedings of Privacy Enhancing Technologies workshop (PET 2002)*. Springer-Verlag, LNCS 2482, April 2002.
35. David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In *Information Hiding*, pages 137–150, 1996.
36. Philippe Golle and Markus Jakobsson. Reusable anonymous return channels. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2003)*, Washington, DC, USA, October 2003.
37. Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. Universal re-encryption for mixnets. In *Proceedings of the 2004 RSA Conference, Cryptographer's track*, San Francisco, CA, USA, February 2004.
38. Thomas C. Greene. Net anonymity service back-doored. *The Register*, 2003. `http://www.theregister.co.uk/2003/08/21/net_anonymity_service_backdoored/`.
39. M. Handley, E. Rescorla, and Internet Architecture Board. Internet Denial-of-Service Considerations. Request for Comments: 4732, November 2006.
40. Johan Helsingius. press release announcing closure of anon.penet.fi. `http://www.penet.fi/press-english.html`.
41. Mike Ingle. Interoperability, one-use remailer tickets. Mailing list post, December 1994. `http://cypherpunks.venona.com/date/1994/12/msg00245.html`.
42. Klaus Kursawe, Peter Palfrader, and Len Sassaman. Echolot and leuchtfeuer: Measuring the reliability of unreliable mixes. Technical report esat-cosic 2007-005, Katholieke Universiteit Leuven, 2007.
43. Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.
44. Andrew Loewenstern. Re: Strengthening remailer protocols. Mailing list post, September 1996. `http://cypherpunks.venona.com/date/1996/09/msg00898.html`.
45. Nick Mathewson. Pynchon Gate Protocol draft specification, September 2004. `http://www.abditum.com/pynchon/`.
46. Nick Mathewson. Underhill: A proposed type 3 nymserver protocol specification, August 2004. `http://www.mixminion.net/nym-spec.txt`.
47. Nick Mathewson and Roger Dingledine. Practical traffic analysis: Extending and resisting statistical disclosure. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*, LNCS, May 2004.
48. Tim May. Re: Strengthening remailer protocols. Mailing list post, September 1996. `http://cypherpunks.venona.com/date/1996/09/msg00167.html`.
49. David Mazières and M. Frans Kaashoek. The Design, Implementation and Operation of an Email Pseudonym Server. In *Proceedings of the 5th ACM Conference on Computer and Communications Security (CCS'98)*. ACM Press, November 1998.

50. Jim McCoy. Anonymous mailbox servers. Presentation, HIP'97, August 1997.
51. Roger McFarlane, Adam Back, Graydon Hoare, Serge Chevarie-Pelletier, Bill Heelan, Christian Paquin, and Deniz Sarikaya. Freedom 2.0 mail system. White paper, Zero Knowledge Systems, Inc., December 2000.
52. Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster Protocol — Version 2, July 2003. `http://www.abditum.com/mixmaster-spec.txt`.
53. Steven J. Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*. IEEE CS, May 2005.
54. Peter Palfrader. Echolot: a pinger for anonymous remailers. `http://www.palfrader.org/echolot/`.
55. M. O. Rabin. Digitalized signatures. In R. Lipton and R. De Millo, editors, *Foundations of Secure Computation*, pages 155–166, New York, 1978. Academic Press.
56. Ron Rivest. SEXP—(S-expressions), May 1997. `http://people.csail.mit.edu/rivest/sexp.html`.
57. Len Sassaman. The promise of privacy. Invited talk, LISA XVI, November 2002.
58. Len Sassaman, Bram Cohen, and Nick Mathewson. The Pynchon Gate: A Secure Method of Pseudonymous Mail Retrieval. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2005)*, Arlington, VA, USA, November 2005.
59. Len Sassaman and Bart Preneel. Solving the Byzantine Postman Problem. Technical Report ESAT-COSIC 2007-004, Katholieke Universiteit Leuven, June 2007.
60. Len Sassaman and Bart Preneel. The Byzantine Postman Problem: A Trivial Attack Against PIR-based Nym Servers. Technical Report ESAT-COSIC 2007-001, Katholieke Universiteit Leuven, February 2007.
61. Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a trickle to a flood: Active attacks on several mix types. In Fabien Petitcolas, editor, *Proceedings of Information Hiding Workshop (IH 2002)*. Springer-Verlag, LNCS 2578, October 2002.
62. Robyn Wagner. Don't Shoot the Messenger: Limiting the Liability of Anonymous Remailer Operators. *New Mexico Law Review*, 32(Winter):99–142, 2002.
63. Alma Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *8th USENIX Security Symposium*, 1999.
64. Matthew Wright, Micah Adler, Brian Neil Levine, and Clay Shields. Defending anonymous communication against passive logging attacks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.