A Reputation System to Increase MIX-net Reliability

Roger Dingledine[1], Michael Freedman[2], David Hopwood[3], and David Molnar[4]

[1] Reputation Technologies, Inc.
arma@reputation.com
[2] MIT
mfreed@mit.edu
[3] Independent consultant
hopwood@zetnet.co.uk
[4] Harvard University
dmolnar@hcs.harvard.edu

Abstract. We describe a design for a reputation system that increases the reliability and efficiency of remailer services. Our reputation system uses a MIX-net in which MIXes publish intermediate messages to a distributed public ledger. These messages allow senders to verify the correctness of each MIX and prove misbehavior. We suggest a simple model and metric for evaluating the reliability of a MIX-net, and show that our reputation system improves over randomly picking MIX paths.

1 Introduction

Anonymous remailers are the most common method of anonymous e-mail communication. But despite wide use of the current global remailer network, this network is generally considered unreliable. Messages are often dropped, and the newsgroup `alt.privacy.anon-server` contains many examples of a message being sent two or three times in the hope that one message reaches the destination. This unreliability directly affects the number of people using the remailer network and their traffic patterns. This in turn negatively affects the anonymity these networks can provide.

Several approaches to the problem of unreliability are possible. One approach is to write more reliable software[31]. Our approach is to build a reputation system to track MIX reliability. Because users can choose paths based on the published scores for each MIX, this reputation system provides two improvements to the remailer network:

- Reliability: fewer messages get routed through dead MIXes.
- Efficiency: the system dynamically rebalances the load based on available reliable resources.

Currently deployed remailer reputation systems (better known as 'remailer statistics') collect data independently, and are treated as trusted third parties by client software. Since shutting down the server means that its statistics are no longer available, remailer statistics servers that prove reliable are good targets for adversaries. Furthermore, these statistics often measure secondary properties of the MIX-net servers, such as up-time, which do not necessary translate to reliability. In order to develop a reputation system that is less affected by loss of a few servers, we present a variant MIX-net design which allows the sender to verify that each MIX has behaved honestly at each hop of the path for that message. This verification is possible because MIXes publish intermediate messages to a distributed public ledger.

Similar schemes using a public ledger have been suggested before for MIX-net-based electronic voting protocols[26, 16]. However, the main difference is that these schemes only wish to verify that all votes (messages) have been *correctly* handled, i.e., shuffled from one MIX to the next to ensure anonymity and then counted in the final election tally. Our protocol, on the other hand, provides a system to identify points of failure, integrated with an overall reputation system architecture. Further, while the voting papers abstract the physical construct of a central message board, we describe real design and implementation issues of a distributed append-only ledger.

In our MIX-net scheme, senders can prove to any third party the failure of a specific MIX to pass on their message. We introduce a new set of agents called *scorers*, who tally failure proofs and serve them to client software. Because these negative ratings are published on the ledger, loss of any scorer does not imply loss of these negative ratings. Further, these scorers also perform basic up-time checks in order to distinguish highly reliable MIXes (few delivery failures because they actually deliver messages well) from new MIXes (few delivery failures because nobody has sent any messages through them yet). Client software can be configured to only choose MIXes with a minimum number of successful checks; after that, the number of verified delivery failures is used to rate MIXes.

We go on to describe a metric for evaluating the reliability of a MIX-net or remailer system, and then show that our reputation system improves over randomly picking MIX paths. Our metric is the expected probability of a message successfully being sent through the MIX-net. We specify a probability model for a MIX-net and then give an example of how to evaluate our metric on a simple example of the model where no reputation system is specified. Then we do the same analysis with a reputation system in place and show that a message's chance of surviving improves.

Specifically, our paper is organized as follows. Section 2 contains a description of related and historical MIX-net works and the statistics systems that are currently used. Section 3 presents our MIX-net design, describes specifically how to send a message in this MIX-net, and then argues security of the overall network. Section 4 provides a list of requirements and a design outline for the message ledger. In Section 5, we describe the design and motivation for the reputation scoring system, and the implications of deploying this system. Section 6 describes a number of potential attacks introduced by our new design, and argues that its benefits outweigh the new vulnerabilities. We continue in Sections 7 through 9 to present a model for describing reliability of a MIX-net with and without a reputation system. Finally, we close in Section 10 with a discussion of possible directions for future research.

2 Related Work

2.1 MIX-nets

Chaum introduced the concept of a MIX-net for anonymous communications[3]. A MIX-net consists of a series of servers, called MIXes, each of which is associated with a public key. Each MIX receives encrypted messages, which it then decrypts and forwards after stripping the sender's name and identifying information.

After Chaum, MIX-nets found wide theoretical application, particularly in the design of voting protocols. We mention particularly Kilian and Sako[16], Park, Itoh, and Kurosawa[26], and Jakobsson[14] which make use of a "public message pool" similar to the ledger we propose in Section 4. This idea of a message pool was also used by Riordan and Schneier to provide certified e-mail[29] and by Han in an investigation of simultaneous exchange[12]. While the notion of a public message pool is not new, this work is the first we are aware of that combines a pool with a reputation system in order to provide accountability for individual MIXes.

Work on MIX-nets is ongoing. Current research directions include "stop-and-go" MIX-nets[15], distributed "flash MIXes"[14] and their weaknesses [5, 24] and hybrid MIXes[25]. Much of this work on MIX-nets focuses primarily on the anonymity provided by various MIX protocols.

The reliability and efficiency of MIXes have been a secondary concern. Increasing reliability and efficiency would have a direct positive impact on the anonymity provided by a MIX-net. In order to properly conceal the identity of their users, MIXes require large amounts of traffic. An unreliable MIX-net that requires users to re-send messages or an inefficient MIX-net that often chooses slow MIXes in

MIX paths will discourage most users. Addressing reliability issues will draw more users, increase traffic in the MIX-net, and achieve better anonymity overall.

2.2 Cypherpunk, Mixmaster, and Zero Knowledge Systems Remailers

The first widespread public implementations of MIXes were produced by the cypherpunks mailing list. These *anonymous remailers* were inspired both by theoretical work on MIXes and by the problems surrounding the anon.penet.fi service[13]. Hughes wrote the first cypherpunks anonymous remailer[20]; Finney followed closely with collection of scripts which used Phil Zimmerman's PGP to encrypt and decrypt remailed messages. While these "Type I" remailers did not include crucial MIX features, such as reordering or batching of messages, they provided a better approximation of MIXes than had been generally available. Later Cottrell implemented the Mixmaster system, or "Type II" remailers, which added message padding, message pools, and other MIX features[9]. At about the same time, Gulcu and Tsudik introduced the Babel system, which also gave a practical remailer design (although one that never saw widespread use)[11].

Producing properly formatted remailer messages by hand is a difficult and error-prone task. The program Private Idaho, developed by McNamara, was one of the earliest and best known means of automatically formatting remailer messages[23]. The program is no longer maintained, but many different splinter versions exist. Currently, Potato Software's Jack B. Nymble and Reliable remailer software offer a range of features, including the ability to configure user preferences for picking remailer nodes[30].

Recently, Zero Knowledge Systems began providing commercial anonymity services. Version 1.0 of their "freedom mail" service made use of a Mixmaster-style anonymous remailer. The recently announced Version 2.0 seems to move away from this model and towards a model where mail is held via a POP server accessed by the ZKS network[32].

2.3 Remailer Statistics

After the deployment of the original cypherpunk remailers, the next major step was Levien's development of remailer *statistics pages.* Statistics consist of two parts. The first part lists remailer capabilities, such as what kinds of encryption the remailer supports. The second part lists remailer *up-times* observed by either pinging the machines in question or sending test messages through each machine or group of machines.

Levien statistics are an example of what we will call a *reputation system.* Section 5 will discuss this in more detail. Reputation systems offer a means of improving the reliability of MIX-nets by allowing users to avoid choosing unreliable MIXes. The Jack B Nymble 2 client allows importing statistics files and can then pick remailers according to that data. Users can specify minimum reliability scores, decide that a remailer should always or never be used, and specify maximum latency.

2.4 Reputations and MIX-nets

The cypherpunks list saw much discussion of reputations as a means of improving MIX-net reliability. One common theme, due to May and others, is the use of commercial remailers and for-profit reputation services. Remailer operators would be paid for their trouble using some kind of digital cash and audited by a reputation service that performs up-time tests, site visits to audit hardware, and periodic reports of remailer reliability[19]. This reputation-based approach is contrasted with a "neo-Chaumian" approach which tries to ensure reliability at the protocol level by, for example, publishing encrypted logs or receipts; May argues that reputation-based approaches are more realistic, scale better, and are less prone to abuse[22]. Levien statistics are a first step in this direction; commercial remailers and commercial reputation services would be the next step.

While Levien statistics almost certainly have improved remailer reliability, commercial remailers have yet to see widespread use. Our work explores a combination of the two approaches. We specify a reputation system and then modify the MIX-net protocol to support easy tallying of MIX reputations.

3 A MIX-net With Verifiable Failures

We present a MIX-net design which allows us to build a robust reputation system. As we saw in the previous section, the current reputation systems for MIX-nets are not very robust or well-analyzed. Statistics pages measure remailer "up-time", but make no claim as to the probability of nodes properly remailing messages. In this section, we present the overall design of a MIX-net that supports verification of failures.

One common way to prevent raters from cheating is to design some means of verifying transactions. In many systems, transaction verification is done by having both parties exchange signed digital receipts, or by performing the transaction publicly. We extend the latter notion to develop a design in which all messages in the MIX-net are publicly known. That is, MIXes publish their input and output messages. Since the source of each message is not published, this information is still less than what would be seen anyway by a passive adversary with complete network access. We retain unlinkability between message sender and recipient due to the batching and reordering that each MIX performs internally.

This idea was originally due to Chaum in [3]. We obtain anonymity because messages coming into the MIX are encrypted. The MIX decrypts them and sends out the decryptions, which are themselves encrypted messages (except for the last hop). If we have a semantically secure cryptosystem, then a passive adversary has only negligible probability of determining which incoming message corresponds to which outgoing message. However, we should keep in mind that traffic analysis research is still in its infancy, and it may be that hiding even the information that a passive adversary would see is crucial to real anonymity for remailer networks; see Section 6 for more consideration of this point.

In general, a system to verify transactions should provide verification of both successful and failed transactions. However, due to the pseudonymous nature of a MIX-net, successful transactions can easily be spoofed (via pseudospoofing, as described in Section 5.5) and thus do not provide an accurate means of evaluating MIX reliability.

We present a system that allows senders to identify the point-of-failure MIX in the case of an unsuccessful transaction, and prove this knowledge to any other party. This technique was inspired by Riordan and Schneier's work on certified e-mail delivery [29].

3.1 Overall MIX-net design

Alice wants to send Bob a message anonymously. She uses a MIX-net system consisting of n MIXes, $N_1, \ldots N_n$. We require some public-key encryption scheme with randomized encryption. This will be modelled as a key pair generation algorithm G, a randomized encryption algorithm E, and a deterministic decryption algorithm D. In our notation, we will explicitly include the random value used in the encryption, so $E_i^r(M)$ means the encryption of message M and random value r under the public key of N_i.[1] We assume the usual correctness property that if N_i's key pair is valid (i.e. was generated by G), $D_i(E_i^r(M)) = M$ for any plaintext M and random value r. We also assume that authentic public keys for each node are known to all parties.

Our design includes a public, distributed, append-only ledger. We describe this ledger in much greater detail in Section 4, but we provide a brief summary here. All participants write to this ledger; each entry includes the date and time the entry was written, and a destination node address. Each MIX is responsible

[1] Note that while $E_i(M)$ is a random variable ranging over all the possible encryptions of M, $E_i^r(M)$ is a particular string.

for monitoring the ledger at least every T minutes, checking to see if any new messages are addressed to it. Nodes that do not process new messages within $2T$ minutes are said to have *failed* – they did not pass on the message.

To send a message, Alice chooses a path through the network, repetitively "onion" encrypts her message, and then writes the first onion to the ledger, addressing it to the first MIX in her path. That MIX reads the onion from the ledger, processes it, and finally writes the unwrapped-by-one-layer onion back to the ledger.

If the message does not reach Bob, the transaction has failed. Our system should be able to identify the MIX in the path that caused the transaction to fail. Further, the system should not be vulnerable to attacks to confuse or mislead. More precisely, we have two goals.

- Goal 1, **Identify failure**: If N_j fails to pass on a well-formed message sent by Alice to Bob, then Alice can prove to any third party that N_j was the failing MIX.
- Goal 2, **Reject false claims**: No participant (including Alice) can claim that N_j failed to pass on a well-formed message sent by Alice to N_{j+1} (and possibly from there to Bob), unless it really did fail to send such a message.

We proceed to show the specifics of constructing and sending a message, and then show how we can fulfil these two goals.

3.2 Sending a Message

This section describes the protocol for sending a message from Alice to Bob, using our MIX-net design with verifiable failures.

1. Alice chooses a recipient Bob.
2. Alice chooses k MIXes $N_1, \ldots N_k$ to form a "MIX path".
3. Alice picks $k+1$ random padding values $r_1, r_2, \ldots r_k, r_{Bob}$.
4. Alice creates an initial packet P, defined as
$$P = E_1^{r_1}(N_2, E_2^{r_2}(N_3, \ldots E_k^{r_k}(Bob, E_{Bob}^{r_{Bob}}(M)) \ldots))$$
5. Alice writes P and N_1 (the identity of its initial recipient) to the public ledger.
6. N_1 reads P from the ledger and processes the packet.
7. N_1 publishes to the ledger: "To N_2: $D_1(P)$", where
$$D_1(P) = E_2^{r_2}(N_3, E_3^{r_3}(N_4, \ldots E_k^{r_k}(Bob, E_{Bob}^{r_{Bob}}(M)) \ldots))$$
8. This process is repeated by N_2, which publishes "To N_3: $D_2(D_1(P))$" to the ledger, and so on for $N_3, N_4, \ldots N_{k-1}$.
9. Eventually, $E_k^{r_k}(Bob, E_{Bob}^{r_{Bob}}(M))$ is published to the ledger, and then N_k publishes $E_{Bob}^{r_{Bob}}(M)$. Bob queries the ledger; he obtains and decrypts the message. Thus, the message has successfully been transmitted from Alice to Bob.

While the senders of each message are not published to the ledger, we assume that an adversary can watch the ledger and determine which messages are downloaded by which parties. Together with the information on the ledger, this is exactly the same as the *view* of a passive adversary in traditional MIX-net protocols. Therefore the security proofs that apply to traditional MIX-net protocols apply to our protocol for sending messages as well. As we will see, however, the same does not hold for our protocol for proving MIX-net failures.

It may be possible to deny an adversary information about the senders of messages. Each MIX or potential recipient could download the entire database at once. Alternatively, we could investigate techniques of private information retrieval to reduce communication complexity[18]. We leave this for future work.

3.3 Identifying and Proving Failed Transactions

This section describes the method by which Alice proves the failure of a specific MIX to deliver a message, as well as the protection our protocol offers against a malicious Alice who either sends ill-formed messages or provides false claims.

Alice publishes her claims of failure to the message ledger. In this context of proof systems, Alice is the prover, and another party who reads the message log is the verifier. We will describe these verifiers, and the reputation systems that we develop with them, in Section 5.

Proving that a delivery failure results in a proper claim

Sketch Suppose that a MIX N_i fails to handle a message that N_{i-1} sends it via the ledger. Alice wants to prove that N_i failed to process this message.

1. Because N_{i-1} behaved according to protocol, "To N_i: $D_{i-1}(\ldots D_1(P)\ldots)$" is written on the ledger, where
$$D_{i-1}(\ldots D_1(P)\ldots) = E_i^{r_i}(N_{i+1}, E_{i+1}^{r_{i+1}}(\ldots E_{Bob}^{r_{Bob}}(M)\ldots))$$
2. As Alice created the initial packet P herself, and knows all the random padding values, she can compute the message that N_i is supposed to write to the ledger
$$E_{i+1}(\ldots E_{Bob}^{r_{Bob}}(M)\ldots)$$
which should equal
$$D_i(D_{i-1}(\ldots D_1(P)\ldots))$$
She notices that this message has not been written to the ledger within the allowable time period.
3. Alice claims that N_i failed by writing to the ledger "I blame N_i, claim: $r_i, N_{i+1}, E_{i+1}(\ldots)$."
4. At this point, anybody can verify if Alice's claim is true by encrypting Alice's claim under N_i's public key and checking if it equals $D_{i-1}(\ldots D_1(P)\ldots)$.

Therefore, we satisfy Goal 1, being able to identify failures in the MIX-net. We note that knowledge of N_{i+1} is *not* part of the view of a passive adversary in a normal MIX-net protocol. This is extra information given to a possible adversary; we do not currently know how to both verify MIX failure and avoid giving away this information.

False Claims Are Rejected

Sketch We wish to show that no participant can claim that N_j failed to pass on a well-formed message sent by Alice to N_{j+1}, unless it really did fail to send such a message. We will consider only the case when Alice is an adversary.

- Suppose Alice posted a real message P addressed to N_j as normal and then tried to claim that N_j failed. A well-behaving N_j will produce an intermediate message and post it to the ledger. After Alice claims N_j has failed, any verifier will see a correct intermediate message from N_j on the ledger that refutes Alice's claim. We assumed that the ledger was append-only, so Alice cannot delete such an intermediate message from the ledger.
- Suppose Alice does not post a real message P addressed to N_j on the ledger. Then in order to make a credible claim, Alice needs
  - an entry "To N_j: foo" present on the ledger,

- knowledge of some string bar, seed r_j, and destination N_{j+1}, such that $E_j^{r_j}(N_{j+1}, bar) = foo$,
- for N_j to time out and not post an intermediate message to N_{j+1}.

If $E_j^{r_j}(N_{j+1}, bar) = foo$, then $D_j(foo) = (N_{j+1}, bar)$, assuming N_j's key pair is valid.[2] Therefore if "To N_j: foo" was present on the ledger (who wrote it there is irrelevant), and N_j correctly followed the protocol, then N_j would have obtained (N_{j+1}, bar) from foo, and posted "To N_{j+1}: bar" to the ledger. As a result, the only time Alice can credibly claim failure in this case is if N_j really does fail and not post the message to N_{j+1}.

Therefore, we satisfy Goal 2. Note that the above argument covers the case where Alice attempts to send ill-formed messages to N_j, since (N_{j+1}, bar) is not ill-formed, and we have proven that when Alice's claim is accepted, N_j must have received a message with that as the plaintext.

Note also that we're neglecting replay attacks in this discussion. Replays complicate the situation: a MIX can choose not to respond to a message if it believes it to be a replay, yet such MIXes may lose reputation from this behavior. A simple solution might be to examine the entire ledger to see if the MIX had ever processed it; but there are still timing attacks, e.g., sending replays just as the ledger expires an entry. More complicated and more effective attacks against verifiability and reputations may also emerge as we introduce anonymity-protecting techniques such as replay detection to our design.

4 Design of the ledger

The success of our proposed MIX-net is based in large part on the success of the public ledger that it uses. If the ledger can be destroyed or made unreachable, or even if individual entries on the ledger can be deleted or forced to expire, then the correctness of the entire MIX-net is in jeopardy. Thus we sketch the following requirements:

- Append-only: Entries can be added but not modified.
- Timestamped: Each entry is timestamped upon arrival – by the ledger, not by the submitter.
- Expiration: Entries must be expired to avoid filling the ledger. The lifetime of an entry should be a globally agreed-upon value.
- Robust: The system should resist attacks by powerful adversaries to modify or remove an entry, or otherwise deny service.
- Available: It must be feasible to write to and query from the system. Additionally, the system must support queries for all of the messages to a specified address within a specified timespan.

In general, this can be solved by a distributed "k of n" database where writers split their entries into shares and hand them out to all the participating nodes. Nodes would timestamp each share on arrival. Readers would query the system with requests like "tell me all the messages in this timespan" and "tell me the messages in this timespan that are addressed to Charlie". Since each share is stamped with a time, each node can delete it at the appropriate time (modulo some buffer so we can neglect time synchronization issues – we treat nodes with sufficiently invalid times as adversaries).

Garay et al[10] build on Rabin's Information Dispersal Algorithm[27] to present a system that shares data among n nodes and can store and retrieve data even in the face of up to $t < \frac{n}{2}$ inactive or malicious nodes. They provide a system whereby clients can proxy queries through so-called *gateway* nodes. This approach saves the clients from performing complex calculations and opening many parallel connections; they avoid potential security bottlenecks by allowing any node to serve as a gateway. They also touch on

[2] We can assume that N_j's public key is valid because it is chosen by N_j, who would have no incentive to generate an invalid key. We omit consideration of an N_j controlled by Alice that purposely chooses an invalid key, because then Alice only damages the reputation of nodes she controls.

a technique known as *proactive* recovery, wherein the system maintains its integrity even if an adversary corrupts all of the servers during the lifetime of the system, as long as he corrupts only a fraction during some window of time known as the 'vulnerability window'.

Castro and Liskov[2] consider the notion of proactive recovery from a systems perspective, by building a highly-replicated Byzantine-fault-tolerant filesystem that can resist a very strong adversary by proactively identifying and rebuilding failed nodes. They note that an attacker who compromises a node can learn the keys used to authenticate messages from that node, and address this problem by defining a notion of *authentication freshness* so that other nodes can reject messages that are not sufficiently fresh. They provide a library that is flexible enough to perform a wide range of state-machine based calculations – certainly enough to build our ledger, including expiration of old entries. Their Byzantine-fault-tolerant NFS implementation achieves comparable efficiency to a standard NFS implementation, and their tests show that the window of vulnerability can be made as small as five minutes without any significant impact on service latency.

However, there are some design details that complicate the issue when putting these solutions into the dynamic peer-to-peer context of MIX-nets: for instance, who keeps the list of ledger nodes, and how is it updated? One approach is to instead use a system like Freenet[4] or Free Haven[6] that already addresses the problems of dynamic participants. However, the added complexity of the anonymity requirements for these systems makes it difficult for them to prove with certainty that a given entry is not present. Publius[33] might be a better approach, but it does not yet deal with dynamic joining and leaving of nodes. If we want to go with the Byzantine-fault-tolerant replicated approach described above, we can address this issue by making each MIX node into a ledger node; this allows the same solution that is used to manage the MIX list to also solve the ledger node management problem.

4.1 How often should MIXes query the ledger?

We choose a period T. If a message to N_i has been posted on the log at timestamp t, then at time $t + 2T$ if N_i has not posted a response, N_i has failed to follow through on the transaction.

Thus each MIX should query the log every T time units. Below, we describe some intuition to show that this is sufficient.

```
       0      A       T      B      2T     C      3T      D     4T
-------+--------------+--------------+--------------+--------------+--

     *   &
```

Let's say our MIX queries at times iT for $i \in 0, 1, \ldots$

If Alice posts at point *, then the MIX will read it at time 0, and publish a reply during timespan A. If Alice posts at point &, then the MIX will read at time T, and publish a reply during timespan B. Thus a well-behaved MIX will always respond within time $2T$ to a post.

Based on how long it takes to do batching and reordering, we might conservatively set T to 12 hours. Then we want to expire old messages on the log after at least $2nT$ time, where n is the maximum 'reasonable' number of hops in a path. If Alice publishes her claim on the log too, then it should be $2(n + 1)T$ to account for that extra timestep between when Alice publishes and the scorers read her claim.

If a given MIX publishes late, then other MIXes in the path cannot determine that it was late, so the message will get delivered as normal after the delay. However, this MIX has still failed to follow the protocol, and is just as at fault as a MIX that ignored the message entirely.

4.2 Flood protection for the ledger

Like other peer-to-peer systems, MIX-net systems are vulnerable to flooding by malicious adversaries or even by simply overloading the available resources with too many requests. Specifically, the ledger is vulnerable both to connection flooding (too many reads) and to storage flooding (too many writes). While a broad discussion of these topics is outside the scope of this paper (and we cover them in[7]), we can draw some lessons to use micropayments to protect against some of these problems.

We can require a certain amount of hash cash[1] payment or other client puzzles before a read or write is performed. Indeed, we can even make this a fungible payment in order to allow ledger nodes to materially benefit from providing their services. But this neglects the fact that each MIX needs to be able to read and respond to messages 'for free', else an adversary can deny service to the MIX network by making it too expensive for a MIX to process its messages.

We can allow connections from known MIXes to perform reads and writes without paying. Other parties – specifically, the Alices who want to send messages, and the Alices who post claims – would have to do a time-wasting or other payment operation to get their writes accepted. This reduces the flooding potential for the ledger and MIXes (and also makes it more noticeable when one of these MIXes is misbehaving), while introducing a new authentication issue. But even if the authentication fails (e.g. from an adversary who spoofs his source), then this reduces to the earlier problem – some more parties will be able to read or write for free. Therefore, this is a good start to a solution.

5 Reputation Systems

Reputations have been suggested as a means of improving MIX-net reliability[19, 21]. To date, discussions of reputation systems in the context of MIX-nets have been vague. We now specify what we mean by a reputation system in the context of a MIX-net, and investigate a specific design to help solve the reliability problem.

As stated earlier, the major parties in the overall MIX-net system are Alice (the sender), Bob (the recipient), and the MIXes $N_1, \ldots N_n$. In layering on a reputation system, we add two more agents to the system. *Raters* are entities who make observations about the performance or honesty of MIXes. *Scorers* are entities who tally observations from raters and make these tallies (or *scores*) available. Any of these agents might overlap – indeed, since we're dealing with an entirely pseudonymous system, we have to assume that they can.

5.1 Requirements for Our Reputation System

Our goals for developing a scoring system in a MIX-net context include the following:

- We want to allow senders to configure their software to automatically take into account the scores for each MIX. Indeed, current client software such as [30] already includes this support.
- We want the system to maintain the level of anonymity provided by the MIX-net.
- Scores should be accurate – clients should be able to draw conclusions from scores that lead to 'good' predictions.
- Robust against attacks: The system should resist attempts of any entity or entities to influence scores in a way that doesn't 'reflect reality'.
- No single point of failure: To maintain efficiency, security, and reliability, no single server or small subset of the servers should be a bottleneck anywhere in the protocol.
- Weighted towards current behavior: The system recognizes and reflects recent trends in entity performance. For instance, an entity that has behaved well for a long time but suddenly goes downhill should be quickly recognized and no longer trusted.

– Feasible to implement: We cannot require any features that make an efficient implementation (in respect of both time and bandwidth) impossible.
– Anybody can rate: Any user of the MIX-net can contribute observations about MIX behavior.
– Verifiable: Scorers need some way to determine the credibility of ratings; other users need some way to verify that scorers are tallying ratings correctly.

We would like to achieve these goals without requiring excessive overhead or operations that compromise the anonymity offered by the MIX-net.

5.2 Reputation System Overview

Section 3 outlines a technique by which Alice can confirm the failure of a specific MIX to forward her message. When she notices this failure, she publishes a 'claim' on the public ledger; this claim provides sufficient information for any observer to verify that the MIX did in fact fail to perform its duty.

We introduce a set of scorers, each named Sally. Each of these scorers keeps her own database of performance scores for MIXes in the system. Because Alice publishes her claims to the ledger, each Sally can independently query the ledger for new claims. She then performs the encryption as detailed in Section 3, and confirms that the previous onion is present on the ledger.

Sally makes available her database of MIX reputations, such that Alice's client software can download it and use it to automatically choose paths.

There are three interesting things that Sally can remember: negative ratings, positive ratings, and the timestamp of each rating. Since each rating is verifiable entirely by the material publicly available on the ledger, Sally gains no real information by trying to remember the rater's identity. This is convenient, because the claimant may well wish to remain anonymous anyway.

We would like to develop as simple a scoring system as possible – simple systems are easier to analyze for security, and they allow the users to more easily understand the implications of a given score. Ideally we would like to simply count the number of messages that each MIX drops and present that as the score. But that causes brand-new MIXes to have the highest possible reputation; an effective attack against the MIX-net would be to constantly add new unreliable (or even unreachable) MIXes. To get around this, we instead compute the score so it reflects both the count of negative ratings, and also a 'minimum number of confirmed positive ratings' requirement, which is configurable on the client side. We detail both of these approaches below.

5.3 Tallying negative ratings

As we showed in Section 3.3, each negative rating really does represent a failed delivery. That is, if there's a valid onion addressed to N_i published on the ledger, then it really is N_i's fault if an onion to N_{i+1} doesn't show up within the allowed time window. Anybody who is able to demonstrate that it was a valid onion is either that MIX or the onion's author; the MIX has no incentive to publicize that it failed at a transaction.

Thus, there is no way to spoof negative ratings. Note that there does seem to be a way for an adversary to force negative ratings on a MIX: if he floods the ledger with valid onions all addressed to that MIX, then at some point the MIX will no longer be able to sustain the load. However, this point is exactly the point where the MIX is demonstrating the fact that it is unreliable – it's unable to handle the current load of messages that it has, so it's unlikely to be able to handle new ones either. Thus, causing MIXes to lose reputation in the face of *successful* flooding is consistent with our scoring system goals; after all, the scoring system measures reliability and capabilities, not intent.

5.4 Tallying positive ratings

Positive ratings differ from negative ratings in that they can be easily faked. Specifically, an adversary can make up a set of messages, each using a MIX path entirely owned by him, and then simply post the alleged transcripts of the message on the ledger all from the same machine that built the onions in the first place.

Thus, algorithms that consider the number of positive ratings (either by adding positive and negative ratings, as eBay does, or by taking the ratio of positive ratings to negative ratings) are generally going to be vulnerable to this sort of shilling attack.

Yet using only the number of negative ratings to calculate reputation score gives a perfect score to completely untested MIXes. Specifically, reducing the reliability of the overall system is as easy as getting a new evil MIX added to Sally's list. We need some way to confirm that positive ratings actually reflect a MIX's ability to successfully deliver Alice's message in the future.

5.5 Possible approaches to increase confidence in positive ratings

One approach to making positive ratings more reliable (and thus more meaningful) is to build a graph based on rater credibility, such as that employed by Advogato[17]. Similar to the PGP web of trust, this technique builds a directed graph where edge weights represent amount of trust that one vertex places in the other; in our case, each vertex would be a MIX. Calculating maximum flow from some node A in the graph to another node B shows how much trust A places in B, even if A never directly rated B.

One of the goals of the Advogato trust metric is to reduce the amount of damage that an adversary can cause by *pseudospoofing* – creating a multitude of identities each controlled by that adversary. Even if one entity joins under many different assumed names, none of these names will gain very much more trust than if they had joined alone. Pseudospoofing is resisted because each of the new names, at least at first, is connected only to itself in the graph. No one else has any reason whatsoever to trust it. Therefore, there is no trust flow from the source to any of the pseudospoofing nodes and none of them are trusted at all. Even after the pseudospoofing nodes begin to form connections with the rest of the graph, there will still be "trust bottlenecks" that limit the amount of trust arriving at any of the pseudospoofing nodes.

A more standard approach might be to develop a Bayesian network to treat reputations as probabilities. A reputation is effectively an estimate of how likely a future transaction with that MIX is to be honest. In this case, scores are simply computed as the sum of ratings, perhaps weighted by the credibility of raters. More complex systems can be built from neural networks or data clustering techniques, to develop ways of applying non-linear fitting and optimization systems to the field of reputation.

It is difficult for these systems to be easily verifiable by the other users in the system. Indeed, to our knowledge most of the more statistical approaches to aggregating reputation values have never been studied in the context of adversaries who understand the algorithm and have a wide range of possible approaches to exploit the system. Indeed, in many statistical or academic approaches the goal is simply to combine the ratings into as accurate a score as possible. In the statistical analysis, no regard is given for whether participants in the system can conspire to provide ratings that break the particular algorithm used. When developing scoring systems, we need to keep in mind that simply applying evaluation techniques that are intended to be used in a "clean" environment may introduce serious vulnerabilities. This note is particularly crucial when deploying reputation systems in an environment where pseudospoofing is a real threat.

In order to address the 'new node' problem, we could introduce a new party in the system called a *reviewer*. The job of the reviewer would be to send test messages through the system and verify the correct behavior of MIXes. Each reviewer would publish *signed* claims to the ledger. Reviewers would develop credibility based on accuracy of these ratings. But this introduces a recursive problem: we would need to develop a system where reviewers can develop a reputation for 'rating well'. There are a number of questions that complicate this idea:

– Would Sally keep track of reviewer reputations, to shield her users from dealing with that? If so, are the reviewers really much more than just an extension of a trusted Sally?
– If Alice kept track of reviewer reputations herself, then either Sally would have to hand her a whole lot more information, or Alice would have to give Sally all her preferences and Sally would crunch the numbers for her. Neither of these seems particularly appealing.
– What precisely does a reviewer giving a positive rating for a MIX mean? Is it an endorsement, or just a statement of fact? If a reviewer gets unlucky and his message is delivered successfully by a flaky MIX, then by rating that MIX well his reputation is bound to decrease. Reviewers already have little incentive to publish ratings; this risk gives reviewers an incentive to remain quiet.

These issues indicate that a full-blown reviewer system may be very complicated to design and deploy effectively. Fortunately, we can solve the MIX-net reputation problem with a much simplified version.

5.6 The current solution

It suffices in this case to have Sally be her own reviewer. That is, Sally sends the test messages herself – after all, she knows that they are unbiased ratings. She periodically sends messages through the MIX-net, co-ordinated so as to produce an even distribution coverage of each MIX. MIXes that reliably and correctly process her message will earn positive ratings that Sally knows to be accurate.

Sally should expire her memories of transactions after a certain amount of time so that failures in the distant past don't haunt a MIX forever. Similarly, MIXes need to have a threshold of recent successes in order to stay 'in the running'.

Alice configures her client software to choose only among MIXes that have a specified minimum number of positive ratings. Out of this pool, she weights the MIXes that she chooses for her path based on the number of verified delivery failures that have been observed from this MIX.

Because the client software simply requires a minimum number of successful deliveries, Sally has a lot of flexibility. Her only real goal is to weed out the systems that are not able to deliver any messages – that is, systems that are not 'real' MIXes. Beyond that, the count of message failures will be the differentiator between a reliable MIX and an unreliable one.

Indeed, this system has the property that it reacts quickly to a decrease in reliability of a MIX. If a MIX has a high reputation, then it is likely that there are a number of users routing messages through it. If it suddenly stops delivering messages, then there will be a lot of users that notice these failures, and a series of negative ratings will quickly arrive. This negative feedback process serves to stabilize the system and help the scores reflect reality.

5.7 Redundancy and verifiability of scorers

Alice can use whichever Sally she happens to like. Some Sally's might add value simply by combining the reputation scores from other Sally's – that is, they would not observe the ledger directly or verify ratings themselves.

If Alice keeps a tally herself of her transactions (and she can, by examining the ledger and confirming which mails got through and which didn't), then she can build her own score table in which she is confident of both the positive ratings and the negative ratings. Then every so often she might compare her score tables with those of the available Sally's, and start using the most closely matching Sally as her primary score provider. This process of allowing Alice to 'test the waters' herself means that each Sally does not have to be trusted as much – Sally is more accountable for her scores because Alice can also compute scores and compare.

Indeed, since the claims for dropped messages are published and available on the ledger, anybody at all can verify them. Thus Alice might keep track of negative ratings for a few weeks, and then compare with Sally to determine if Sally's scores are actually reflecting all of the negative ratings that are claimed.

If a given Sally suddenly disappears, all of the negative ratings about each MIX are still present and available on the public ledger. Others can step up and take her place.

6 Implications for traffic analysis

Just about any extension or variant of a MIX-net protocol has implications for traffic analysis. Because there is no real accepted framework for considering traffic analysis attacks and defenses, however, it is difficult to analyze the results of any changes in the protocols. In this section we present a couple of questions and considerations and some thoughts on resolving them.

A very important flaw that we create by publishing intermediate messages is that all adversaries are now at least as powerful as a global passive adversary. Indeed, we could argue that the ledger provides even greater information to observers – some MIX-net designs use secure channels between each MIX; secure channels plus padding means that adversaries need to compromise some nodes in order to gain any information at all.

An active adversary who modifies (or *marks*) a message as it comes through his MIX could trace the MIX path much more easily if all intermediate messages are published. We may be able to protect against this by publishing a fraction of the random padding, to resist the creation of an alternative 'valid' message; but we have not considered this further.

An alternative MIX-net design sends message receipts only to Alice (and maybe even only on demand, if she uses some proof of knowledge to demonstrate that she is allowed to get the receipt). Then Alice can do some rudimentary message tracing without having any of the intermediate messages published on a ledger. Indeed, our proposed reputation system is flexible enough that it would work well in this new design, with only a few changes. However, this design provides no mechanism for Alice to prove to the rest of the network that a specific node dropped her message. Our goal is to build a system in which it is absolutely clear when a given MIX is at fault; without this verifiability, a complex and possibly delicate trust network is necessary to provide accurate reputation estimates in the face of pseudospoofing and other pseudonymity-related attacks. Another goal is to create a system where loss of scorers does not imply loss of scores; without this robustness, ordinary users may find it much more difficult to obtain credible reputation scores. We intentionally choose a simplified (and possibly more vulnerable) MIX-net design in order to provide a more easily analyzed reputation system and reliability metric.

Indeed, the publishing of the intermediate messages themselves may not be the biggest flaw in our system – the claims regarding a bad node may give out path information as well. When Alice publishes a claim about a specific MIX's failure to forward a message, is she helping Charlie learn that that MIX was part of her path? Does that information allow Charlie to more easily link previous messages that Alice sent over that same path? Indeed, can an active message-marking adversary trick Alice into proving to the world that she was the message's sender?

Note that traffic analysis discussions from Crowds[28] indicate that Alice should not choose a different path for each message; rather, she should choose a single path and then stick to it until it fails. Of course, since she only reveals more traffic analysis hints on a failure, she'll want to be switching paths at that point anyway – after all, that path no longer works.

This problem is further magnified by the realization that unless we use some mechanism to obscure the fact that Bob is not a MIX, it is very difficult for Alice to make a claim against the last MIX in her path. That is, if she publishes "N_i dropped my message, and it was supposed to go to Bob" then she has just published the fact that she was sending a message to Bob. This may be a very strong argument for having Alice only make claims regarding messages that are not meaningful to her – that is, messages that she sent with the purpose of identifying a failure and publishing a claim.

The reputation system introduces new attacks as well. One conceivable attack on the system is for Charlie to gain a high reputation in order to get more traffic routed through his MIX. Charlie might want this increased traffic in order to make traffic analysis easier. In the current system without reputations, the way to purchase more traffic to analyze is not so clear – but once we introduce a good reputation system, it's simply a matter of maintaining a reliable MIX in order to draw traffic. The other side of this is that now there is incentive for Charlie to degrade or sabotage the performance of other nodes in order to make his relative reputation higher. This kind of attack was described by RProcess as "selective denial of service": the bad guys want traffic to go through their nodes, so they ensure that all other nodes are less reliable[31].

On the other hand, we should remember also that we may well be making the system *more* anonymous by making the system more reliable. Currently, users may have to send a message many different times, through many different paths, before it successfully arrives at its destination. These re-sends of the same message also offer more information to traffic analyzers. Reducing the number of repeated messages may be an effective measure at increasing the anonymity offered by the overall remailer system.

7 Quantifying MIX Reliability

We now turn our attention to quantifying MIX reliability to measure the improvement in reliability provided by reputation systems. It seems intuitively clear that using Levein style statistics to pick remailers gives a message a better chance of reaching its destination safely. After all, isn't it obvious that the statistics give a remailer client "more information" than it would have had otherwise? Why would we need to introduce a formal way of quantifying this information and its result? There are at least three answers to this:

1. To confirm our intuition. Cryptography, especially anonymous systems, is a surprising field.
2. To offer a means to compare two different proposals for MIX-nets based on a quantitative measure of the reliability improvement they provide.
3. To better understand why and how reputation systems work. In particular, to gain insight into what could happen if a reputation system provides incorrect reputations (either through stupidity or malice).

We now present a model and metric which will fulfil the first of these goals. The second and third are left for future work.

7.1 Reliability Metrics

A *reliability metric* is a quantitative measure of the reliability of a MIX-net. We want to be able to specify a model for MIX-nets, then evaluate some function R that gives us a value or set of values representing the reliability of the MIX-net. In addition, we want the following from any such metric:

1. A higher value should correspond to a more "reliable" MIX-net.
2. An adversary cannot cause the metric to go to a value that does not reflect the actual reliability.
3. The metric should be easy to evaluate, at least in a model where we specify an adversary's range of possible behavior.

7.2 A Proposed Reliability Metric

We now propose a reliability metric which is the expected value of the probability that a message sent by a sender reaches its intended receiver.

MIX Model Before we can specify a metric, we first must specify a model of a MIX-net. This model will concern *only* an adversary interested in minimizing the reliability of the MIX-net; we leave a model that unifies this concern with anonymity for future work. Further, we specify a constant upper bound on the number of senders and receivers of the MIX-net at any point in order to simplify computation.

First the players. Let $MIXes$ be the set of m mixes making up the network. Let the set of senders be S, and the set of receivers be R. Let $Paths \subset MIXes^*$ be the set of valid *MIX paths*, that is, the set of sequences of MIXes allowed as paths by the MIX-net protocol.

Senders first choose a receiver and then choose an ordered sequence of MIXes; we will, however, neglect the ordering of MIXes in what follows. Both choices are random variables. For each sender $s \in S$ we define the random variables R_s corresponding to s's choice of receiver given its knowledge about the MIX-net, and $Path_{s,r}$ corresponding to s's choice of MIX paths when sending to receiver r. Each random variable may have any distribution, although at this point we can only analyse instances of this model with particular distributions as shown below.

For simplicity, we restrict ourselves to a single receiver at a time with MIX paths that are exactly k nodes long; this means that R_s takes on single values $r \in R$ while $Path_{s,r}$ takes on values that are sequences of length k in $Paths$.

We consider an adversary that corrupts MIXes and causes them to fail. Every $mix \in MIXes$ is either *good* or *bad*. Good MIXes work normally (but may fail due to error), and bad MIXes are controlled by an adversary. Good MIXes have probability of failure p_{good} and bad MIXes have probability of failure p_{bad}. Here "failure" means failure to deliver a message in a sense we will soon make more precise. An *adversary* is then the subset $BAD \subset MIXes$.

The fundamental operation in our MIX-net model is *sending a message*. To send a message, a sender $s \in S$ picks a recipient $r \in R$ according to the distribution R_s, and then picks a MIX path, $path \subset Paths$. The selection of nodes may or may not be with replacement. The sender then sends the message along $path$ to r.

We say a message *survives* the MIX-net if it passes through *path* without any of the MIXes failing. We say that the message is *dropped* by the MIX-net if any of the MIXes along the path fail. The goal of the sender – and our goal as protocol designers – is to minimize the number of dropped messages.

The Expected Success Value Metric The metric we will use is that the reliability of the MIX-net is the expected value of V defined as the the event "a message survives the MIX-net." We further define V_s for "a message from sender s survives the MIX-net" and $V_{s,r}$ for "a message from sender s to recipient r survives the MIX-net."

We can evaluate the expected value of V by using the law of total expectation:

$$\begin{aligned} E(V) &= E(E(V_s)|\texttt{sender} = s) \\ &= E(E(E(V_{s,r})|R_s = r)|\texttt{sender} = s) \end{aligned}$$

That is, we can compute $E(V)$ by first computing the probability of survival for a single message sent from a sender to a receiver, then averaging and summing over all senders and all receivers.

8 MIX Reliability With No Reputation System

We now compute the reliability metric on a sample MIX-net with no reputation system whatsoever. We assume that there is no information available on which MIXes are good and which are bad, and so each client picks randomly; that is, for all s and r, $Path_{s,r}$ is a discrete uniform random variable. We make the simplifying assumption that senders choose from all receivers uniformly at random, so for all s, R_s is a discrete uniform random variable. Finally, we set $p_{good} = 0$ and $p_{bad} = 1$; good nodes are always reliable, while bad nodes always fail.

Suppose the adversary controls a randomly chosen subset BAD of the m MIXes, with $|BAD| = q$. Then when picking a MIX path to send to a receiver, a sender has chance $\frac{q}{m}$ each time it picks a MIX of picking a bad MIX. Therefore, the chance of picking no bad MIXes is $(1 - \frac{q}{m})^k$, since the sender picks k MIXes. Because a message survives the MIX-net if and only if no bad MIXes are picked in the path, this gives us $E(V_{s,r}) = (1 - \frac{q}{m})^k$.

We can now compute V_s, making use of the fact that every receiver has the same $E(V_{s,r})$ and that receivers are picked uniformly.

$$V_s = \Sigma_{i=1}^{|R|} \frac{1}{|R|} E(V_{s,r})$$

$$V_s = \Sigma_{i=1}^{|R|} \frac{1}{|R|} (1 - \frac{q}{m})^k$$

$$V_s = (1 - \frac{q}{m})^k$$

We want $E(V)$, which we can compute by taking $E(E(V_s))$ to find $E(V) = |S| \cdot (1 - \frac{q}{m})^k)$. Notice that if the adversary controls half the MIXes so $q = \frac{m}{2}$, for example, then we have $(1 - \frac{q}{m})^k = (\frac{1}{2})^k$ – the reliability of the MIX-net drops off exponentially as the number of hops in the MIX path increases.

9 MIX-net Reliability with Reputation Systems

We now show that a reputation system improves the reliability of a MIX-net in our model. The only quantities that change between the previous section and this section are the distributions $Path_s$. A reputation system will give each sender s some information about which MIXes are good and which are bad. If all goes well, s can then adjust its choice of MIX paths to avoid the bad MIXes.

In order to study this adjustment, we make several simplifications. First, time is divided into two phases called the "observe phase" and the "advice phase." In the observe phase, Q queries are made by raters into the MIX-net. The information from these queries is collected and made available by a central scorer. Then in the advice phase, we assume that all senders adjust their choice of MIXes based solely on the advice of the scorer.

In the observe phase, we assume that raters make queries independently of each other by sending test messages to individual MIXes. If the test message survives, the rater should *report* that the MIX is good. If the test message is dropped, the rater should report that the MIX is bad. We call a report *honest* if it issues from a query and the report and query agree. We further call a report *correct* if the report identifies an adversary-controlled MIX as a bad node or a good MIX as good; a report may for example be honest but incorrect if a good MIX drops a query.

After the queries are complete, the scorer makes the results of the queries available to all senders. Each sender then picks MIX paths only from those MIXes that have the least number of bad reports. In our example situation where $p_{good} = 0$ and $p_{bad} = 1$, all honest reports will be correct, so no good nodes will have bad reports.

Therefore a single query to an adversary-controlled MIX is enough to establish a bad report and disqualify it from MIX path selection. The probability that a rater queries a bad MIX is $\frac{q}{m}$, so in Q queries we remove an expected $\frac{Q \cdot q}{m}$ bad MIXes from consideration by the senders.

Recall that we found the chance that a sender picks a bad MIX path to be $(1 - \frac{q}{m})^k$. Now in the "observe phase" we have established that instead of q bad MIXes, there are an expected $q - \frac{Q \cdot q}{m}$ bad MIXes. Following the analysis in the previous section, we see that $E(V)$ becomes $|S| \cdot (1 - \frac{q \cdot (1 - \frac{Q}{m})}{m})^k$, which is an improvement over random guessing.

Therefore we have shown in this simplified model that honest reports yield an improvement in MIX-net reliability. Note that the first part of this paper is precisely about how to ensure reports are honest, so that we can assume honest reports at this point. There are several directions we could go from here to make this a more realistic analysis:

- Investigate other reliability metrics that may be more natural or easier to work with.
- Bring the adversary model closer to reality.
- Treat anonymity and reliability simultaneously, with an eye to "how many people use the service to provide cover." One way to model this might be to set the number of users as a function of the reliability of the MIX-net.
- Investigate the cases when $p_{good} \neq 0$ and $p_{bad} \neq 1$.
- Investigate non-uniform distributions for each of the random variables involved.

10 Conclusion and Future Directions

We have described a more robust reputation system for remailer networks, based on a MIX-net design that employs a public ledger. This reputation system improves the reliability of the overall network over a random MIX path, and is less susceptible to service denial than other remailer reputation solutions such as statistics pages. There are a number of directions for future research:

- A more detailed construction, as well as deployment, of a distributed robust ledger – including addressing the flooding protection problems – would be of great use to many different projects.
- Along with the construction of the ledger comes the deployment of our proposed reputation system, to see how well it really works.
- More analysis of our model and our reliability metric, as described in the previous section, could lead to better reputation systems. Indeed, a parallel model characterizing *efficiency* of a MIX network might be very enlightening.
- Commercial remailers. Although there are substantial digital cash issues involved, making remailer nodes profitable would really enhance reliability, leading to larger anonymity groups. Indeed, the NymIP group[8] has a long list of similar improvements that need to be further analyzed and developed.
- Currently Bob needs to query the ledger in order to receive his messages. A scheme that enforces accountability for the last hop in the path but that also delivers the messages directly to Bob would be far superior.
- Can we make a reputation system that is both efficient and universally verifiable? Currently, only Alice can claim and prove that a message did not reach its destination. Can we extend this so that anyone can detect a failed MIX node?

This paper provides a foundation for further analysis of MIX-net reliability and reputations. Our model and reputation system are designed to be simple and easily extensible. Much work remains in a wide variety of directions before a reliable, secure, and ubiquitous remailer network can be put in place.

Acknowledgements

We'd like to thank Nick Mathewson and Blake Meike for their help with the reputation system, Anna Lysyanskaya and Chris Laas for various intelligent statements, and our anonymous reviewers for their many useful comments.

References

1. Adam Back. Hash cash.
<http://www.cypherspace.org/~adam/hashcash/>.
2. Miguel Castro and Barbara Liskov. Proactive recovery in a byzantine-fault-tolerant system.
<http://www.pmg.lcs.mit.edu/~castro/application/recovery.pdf>.
3. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1982.
4. Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system.
<http://freenet.sourceforge.net/>.
5. Yvo Desmedt and Kaoru Kurosawa. How to break a practical MIX and design a new one. In *Eurocrypt 2000*. Springer-Verlag, 2000.
6. Roger Dingledine, Michael J. Freedman, and David Molnar. Free Haven: A distributed anonymous storage service.
<http://freehaven.net/>.
7. Roger Dingledine, Michael J. Freedman, and David Molnar. Accountability. In *Peer-to-peer*. O'Reilly and Associates, 2001.
8. John Bashinksi et al. The NymIP effort.
<http://nymip.sourceforge.net/>.
9. Electronic Frontiers Georgia (EFGA). Anonymous remailer information.
<http://anon.efga.org/Remailers/>.
10. J. Garay, R. Gennaro, C. Jutla, and T. Rabin. Secure distributed storage and retrieval, 1997.
11. C. Gulcu and G. Tsudik. Mixing E-mail With Babel. In *NDSS 96*. Springer-Verlag, 1996.
12. Yongfei Han. Investigation of non-repudiation protocols. In *ACISP 96*. Springer-Verlag, 1996.
13. J. Helsingius. anon.penet.fi press release.
<http://www.penet.fi/press-english.html>.
14. Markus Jakobsson. Flash Mixing. In *PODC 99*. ACM, 1999.
15. D. Kesdogan, M. Egner, and T. Bschkes. Stop-and-go MIXes providing probabilistic anonymity in an open system. In *Information Hiding Workshop 1998*. Springer Verlag, 1996.
16. Joe Kilian and Kazue Sako. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In *EUROCRYPT '95*. Springer-Verlag, 1995.
17. Raph Levien. Advogato's trust metric.
<http://www.advogato.org/trust-metric.html>.
18. Tal Malkin. *Private Information Retrieval*. PhD thesis, MIT, 2001? see
<http://www.toc.lcs.mit.edu/~tal/>.
19. Tim May. Cyphernomicon.
<http://www2.pro-ns.net/~crypto/cyphernomicon.html>.
20. Tim May. Description of early remailer history. E-mail archived at
<http://www.inet-one.com/cypherpunks/dir.1996.08.29-1996.09.04/msg00431.html>.
21. Tim May. Description of Levien's pinging service.
<http://www2.pro-ns.net/~crypto/chapter8.html>.
22. Tim May. Payment mixes for anonymity. E-mail archived at
<http://www.inet-one.com/cypherpunks/dir.2000.02.28-2000.03.05/msg00334.html>.
23. Joel McNamara. Private Idaho.
<http://www.eskimo.com/~joelm/pi.html>.
24. M. Mitomo and K. Kurosawa. Attack for Flash MIX. In *ASIACRYPT 2000*. Springer-Verlag, 2000.
25. M. Ohkubo and M. Abe. A length-invariant hybrid mix. In *ASIACRYPT 2000*. Springer-Verlag, 2000.
26. C. Park, K. Itoh, and K. Kurosawa. Efficient anonymous channel and all/nothing election scheme.
27. Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance, April 1989.
28. Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *DIMACS Technical Report*, 97(15), April 1997.
29. James Riordan and Bruce Schneier. A certified e-mail protocol with no trusted third party. *13th Annual Computer Security Applications Conference*, December 1998.

30. RProcess. Potato Software.
<http://www.skuz.net/potatoware/>.
31. RProcess. Selective denial of service attacks.
<http://www.eff.org/pub/Privacy/Anonymity/1999_09_DoS_remail_vuln.html>.
32. Zero Knowledge Systems. Freedom version 2 white papers.
<http://www.freedom.net/info/whitepapers/index.html>.
33. Marc Waldman, Aviel Rubin, and Lorrie Cranor. Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system.