Echolot and Leuchtfeuer
Measuring the Reliability of Unreliable Mixes

Klaus Kursawe[1] and Peter Palfrader[2] and Len Sassaman[1]

[1] Katholieke Universiteit Leuven
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{len.sassaman,klaus.kursawe}@esat.kuleuven.be
[2] Paris Lodron Universität Salzburg
Salzburg, Austria
ppalfrad@cosy.sbg.ac.at

Abstract. In a mix-net, information regarding the network health and operational behavior of the individual nodes must be made available to the client applications so they may select reliable nodes to use in each message's path through the mix-net. We introduce the concept of a *pinger*, an agent which tests the reliability of individual mixes in the mix-net, and publishes results for the mix clients to evaluate.
We discuss the security concerns regarding pingers, including the issues regarding anonymity set preservation, information disclosure, and node cheating. We present our software *Echolot*, the most comprehensive and widely adopted pinger for the Mixmaster anonymous remailer network.
To address a serious anonymity weakness potentially introduced by the careless deployment of pingers, we present *Leuchtfeuer*, a new protocol enhancement for mix-nets. Leuchtfeuer eliminates the active and passive intersection attacks that are possible when different users obtain conflicting reliability statistics about the mix-net.

1 Introduction

Chaum [8] introduced the concept of mixes as a method of providing secure anonymous network communication. The publicly accessible mix networks, such as the "Type I" Cypherpunk remailers [13], the "Type II" Mixmaster [19] network, and the "Type III" Mixminion network [10], as well as the low-latency network anonymity service Tor [12], are operated on a volunteer basis and are prone to intermittent failure of individual nodes.[3] It is therefore necessary for mix client software to have an accurate view of the health of the nodes in the mix network. This information is gathered by sending test messages through each node and observing the success or failure of the mix to successfully transmit the message. In a similar fashion, links between mixes are examined by sending messages through every combination of two consecutive mixes.

[3] Furthermore, the security of these mix-net systems is based on the principle of *distributed trust*: no individual mix is considered to be trusted, but rather the set of mixes chosen by the mix-net client software is presumed to contain enough honest nodes to ensure the security of the communication.

Since the overhead and operational complexity involved in monitoring an entire network of mixes is too great for the average user, reliability testing servers, or *pingers*, perform this function and publish their results in a machine-parseable format. The results are downloaded and interpreted by the mix clients. Pingers track additional information as well, such as the average latency provided by each mix, changes in the key information and capabilities of the mixes, and so forth.

In this paper we give an overview of the different pinger systems that have been developed for the Mixmaster network, and describe the problems they attempt to address, as well as their relative success at doing so. We present Echolot, our pinger implementation which more adequately addresses the problem of reliability monitoring than the other pingers. Finally, we explain the problem of pinger inconsistency, an issue which poses significant security implications and is shared by all existing pingers and mix clients. To solve this, we present the pinger agreement protocol Leuchtfeuer.

2 Related Work

A variety of pinger software has been available for many years prior to the creation of our pinger implementation, *Echolot.* These pingers, as well as Echolot, fundamentally behave in the same way: they attempt to relay messages through known mixes, record the success or failure of these attempts as well as the latency of the mix, and publish their results for mix-net clients to use. In this section, we briefly describe the other pinger implementations which preceded the creation of Echolot.

2.1 rlist

Raph Levien introduced the concept of a monitoring service for anonymous remailers. His software rlist [16] was the first remailer reliability service to monitor the status of the system by sending test messages through each known remailer. At first designed only to work with the Cypherpunk remailer network, rlist later incorporated support for the Mixmaster network as it became widely deployed.

Once started, rlist would run indefinitely, regularly sending simple test messages through remailers and building statistics files, which were obtainable via a `finger` interface.

2.2 pingstats

Pingstats [9], developed between 2000 and 2003 by a programmer using the pseudonym of cmeclax, is a pinger for both Cypherpunk and Mixmaster remailers. It consists of a C program which computes the statistics and a collection of shell scripts that manage the creation, sending, and receiving aspect of pinging, as well as help with collecting keys of known remailers and making a list of their capabilities. These programs are called from Cron, a Unix daemon that executes programs at previously specified times.

Pingstats employs random tokens in message payloads as a counter-measure to cheating mixes (see Section 3 below).

Pingstats presents a more timely view of the state of the network by using weighted pings for its reliability calculation. Thus, older ping results are given less weight than more recent data.

2.3 remlist

Christian Mock wrote remlist [18] at about the same time as cmeclax developed pingstats. Mock's pinger also generated non-deterministic ping payloads but did not do any weighting of data. Remlist's main distinguishing feature was the use of MySQL [1], a relational database, as its storage backend.

2.4 Mixminion Directory Server

Mixminion[10] generalized the concept of pingers, defining a directory server component of the Mixminion system which is responsible for the distribution of all information about remailer availability, performance, and key material. The designers of the Mixminion system considered the attacks on the independent pinger model, and specified that directory servers be synchronized as well as redundant.

Mixminion publishes signed *capability blocks* in the directory server, consisting of the supported mix protocol versions, mix's address, long-term (signing) public key, short-term (message decryption) public key, remixing capability, and batching strategy.

3 Veracity Attacks

A mix which is otherwise honest (in that it correctly performs mixing duties without breaking the anonymity of the messages transmitted through it) may attempt to convince a pinger to provide false information regarding the performance of the mix by identifying the source address of pings and treating the pinger messages differently than normal messages. While this manipulation will not change basic results such as the operational status of a defunct mix, it could allow a mix to alter the latency statistics reported for its operation.

We experimented in Echolot 1.x with a technique intended to discourage such cheating by creating ping messages which originate and terminate at a local mix that also mixes normal messages, so that the target mix cannot distinguish between user messages and pinger messages. Unfortunately, systems such as Mixmaster have a minimum distance between hops which is considered when creating a mix chain, and thus messages which consist of the mix chain A,B,A will still be distinguishable as pinger messages, since no properly functioning mix-net client would generate this chain. If a pinger were to create a chain of

A,B,C,A, neither mixes B or C would be able to tell that the message contained pinger information, but the results would only indicate the combined latency of the mixes B and C, as well as the health of both B and C and the link between those mixes. It would not provide any useful information about B or C alone.

The pinger message data (or *pings*) themselves should not be deterministic, lest a mix attempt to "back-fill" the results for pings sent during a period when the mix was offline. This can be prevented by the inclusion of random tokens in the message payload of the ping, so that pings returned to a pinger without bearing the token of an outstanding ping are discarded without being collated.

4 Echolot

The first version of Echolot [22] was written in 2001. It sent pings through a local Mixmaster node in order to prevent the nodes being tested from learning that a message contained a ping. Other than that it was fairly similar to other pinger software in existence at the time.

Echolot 2.x, a complete rewrite, followed in 2002. Instead of a being a group of programs with order-of-execution dependencies and potential race conditions as its predecessor (and all other existing pingers) were, Echolot 2.x was built as a single daemon. In addition to the normal single hop pings, this version also featured automated node discovery. The following year Echolot 2.1 was released, adding the capability for chain pinging.

Echolot is the most widely used pinger for the Types 1 and 2 remailer networks. As of this writing, there are over a dozen Echolot pingers operating publicly [20].

4.1 Reliability Measurement Aspects

Echolot tests multiple areas of failure in the remailer networks and collates this data in a format the Mixmaster software can process, allowing the mix-net clients to make as much use of the available network resources as possible without preventable packet loss.

The most basic test of reliability is the "single ping" test, wherein the known nodes of the mix-net are each periodically sent individual messages encrypted using the network-specific packet format, and the response times and success rates are tallied. These results allow the mix-net client to make general assumptions about the overall behavior of the node being tested.

When performing a "chain ping" test, Echolot creates a message of path length equal to two for all combinations of any two remailers in the network, and tests each of these pairings. If both remailers consistently return single pings, but fail to return chained pings, one can deduce that the failure is occurring at the link between the two nodes. Thus, either remailer can be reliably used, as long as they are not selected to be adjacent nodes in the message path.

Similarly, the latency value observed from the transmission of a chain ping until its return at the pinger measures the combined latency of the two nodes.

If this is significantly different than the sum of the single-ping latencies for each node, the mix-net client could make determinations about the suitability of that pairing of nodes in a chain. As no mix-net clients exist which make use of the latencies of chain pings, Echolot does not currently report them.

4.2 Node Discovery

Distributed mix-nets consisting of independent operators often do not allow for a guaranteed means for nodes to communicate join and exit events to the other nodes in the network. In the case of Mixmaster, there is no central control structure for tracking the existence of functional nodes, and the components in the system must devise a way of obtaining this information. Often, the human operator of a node joining or exiting the network will announce the status change to other node operators and users via the "remops" mailing list or by posting to USENET. Just as often, nodes will come into existence or cease operation without any warning or notification at all.

Echolot regularly queries each remailer for copies of its current keys, a report of its capabilities, and a list of the other remailers known to it. If a quorum of remailers know about a new node in the remailer network that was previously unknown to this Echolot pinger, the pinger will automatically add the new address to the list of remailers, request information about keys and capabilities, and start pinging the address.[4]

Remailers that have not responded to any requests for an extended duration of time are also removed automatically.

Echolot's automation of most routine maintenance tasks produces a more accurate report about the remailer network than would be possible if it relied on intervention by the pinger operator. Additionally, since the remailer network infrastructure is operated by volunteers, it is especially important to minimize the effort necessary to operate components of the network to attract and retain volunteer operators. While the bandwidth, disk storage, and computation resources required to operate a pinger are negligible for most potential volunteers, even as little as one hour a month of human administration time would likely be too costly for many.

4.3 Echolot Algorithm

Echolot's approach for determining remailer reliability is as follows: Distributed over the course of a day, Echolot sends several pings through each remailer, recording the time when they were sent. For pings already received, the time interval between the sending of the ping and its return to the pinger is recorded.

[4] Bootstrapping Echolot requires that an initial list of active mixes be provided to the software by the administrator, who may learn this information by participating in the aforementioned forums. Additionally, Mixmaster releases are packaged with a list of active mixes at the time of the release; if any of these mixes are still active when the Echolot administrator configures his system, Echolot can learn the identity of other, newer mixes by automatically querying the currently-known mixes.

The "reliability" of a remailer is basically the quotient of pings received and pings sent, with some skewing in place to put more emphasis on more recent data while still being fair to remailers with higher latencies: $rel := \frac{\sum w_i \cdot rcvd_i}{\sum w_i}$, where $rcvd_i$ is 1 if a ping was received and 0 otherwise.

The weight w_i of a ping is made up of two factors: $w1_i \cdot w2_i$. The first of them, $w1_i$ is strictly a function of the ping's age: Pings younger than 24 hours have a weight of 0.5; after 24 hours $w1$ is 1.0 for a while until the weight decreases to zero for pings older than 12 days. See Table 1 for the exact numbers used in the Echolot software.

age [days]:	1	2	3	4	5	6	7	8	9	10	11	12
weight $w1$:	0.5	1.0	1.0	1.0	1.0	0.9	0.8	0.5	0.3	0.2	0.2	0.1

Table 1. Weight of pings based on their age

The second part of a ping's weight also considers this node's latency behavior over the last 12 days. If a ping already has returned, its $w2$ is 1.0. However, should it still be outstanding, Echolot computes the ping's age, again with some constant skewing factors: $age_{skewed} := (now - send - 15min) \cdot 0.8$. The weight $w2$ is now the percentage of pings returned with a latency lower than age_{skewed} within the past 12 days.

To illustrate this, assume a ping was sent two hours ago, which makes age_{skewed} 84 minutes. If all pings returned from this node were faster than 84 minutes, then $w2$ is 1.0. If only a third of pings were received within that time frame, then $w2$ is 0.33. If no ping was ever faster than 84 minutes, then $w2$ is zero.

This weighting based on past behavior was introduced to accurately report the reliability of remailers that have vastly different latencies. There exist Mixmaster nodes which return pings within minutes of sending, while others often take many hours to forward a message [21].

In addition to reliability, Echolot also reports a node's latency. The latency reported is simply the median of latencies of all pings received within the last 12 days.

Chain-Pinging: In addition to single-hop pings, Echolot also performs chain pinging to uncover cases where two remailers A and B perform well when tested individually but for obscure reasons,[5] messages sent through A to B fail to arrive at their destinations.

Since pinging every two-hop chain on a frequent basis would put an unnecessary load on the remailer network, Echolot contents itself with only testing each

[5] Broken chains may be unidirectional or bidirectional, depending on their root cause. We have observed broken chains which have resulted from both hardware faults and configuration errors impacting the network path between the two nodes, as well as mistakes in the configuration of the remailer's host server. Others have been due to misconfigurations in anti-spam measures in place on one or both of the remailers' underlying infrastructure. Most broken chains, though, are unexplained.

chain once a week. Chains that warrant closer attention (so called “interesting chains”) are pinged more often—daily.

Echolot reports a chain to be broken if

– at least 3 pings were sent to test the chain, and
– the resulting chain reliability is far smaller than could be expected from the nodes’ individual performance: $\frac{\text{received pings}}{\text{sent pings}} <= rel_A \cdot rel_B \cdot 0.3$.

Chains are considered interesting by Echolot when

– fewer than 3 pings have been sent without any returning, or
– the chain is currently reported broken.

Because Echolot pings chains that it considers working only once a week, it may take a while before it realizes that a previously working chain is now broken. Fortunately, experience with the currently deployed Mixmaster network shows that broken chains do not change very often.

5 Anonymity Set Attacks Based on Pinger Data

A mix network in which users obtain their view of the network’s heath and status from multiple independent sources opens the system to partitioning attacks [25] against the users, based on differences in the mix selection based on the results of the different pingers. In a system of entirely honest pingers, this attack is feasible for an adversary operating in the passive observer model.

Suppose Alice retrieved her network health information from pinger 1, and Bob retrieved his information from pinger 2. An observer watching the network local to Alice or Bob would know which pinger the user under observation chose. If the information provided by both pingers differed in some manner, the delta would contain mixes that, if chosen by Alice or Bob, would betray which pinger had been used to obtain this information. A message processed by a mix contained only in the results provided by pinger 1 could not have been sent by Bob, who obtains his results from a pinger lacking that information.

Additionally, differences in pinger results for mix latency could lead to more subtle variations in mix path selection, which may aid an attacker.

If a pinger is operated by an attacker, it becomes possible to specifically target individual users by providing them with unique information about the network in order to partition them into an anonymity set of size 1. Users can attempt to prevent against this attack by obtaining their pinger results from a widely-published location, such as Usenet, though this does not completely solve the partitioning attack problem, and introduces additional reliability constraints on the quality of the pinger information.

Additionally, many users retrieve from the pinger updated keys for the remailers at the same as time they update their stats. The pinger[6] could manipulate the user into using keys other than those the user intended to. An attacker

[6] or an attacker performing a man-in-the-middle attack on the data retrieval session.

who controlled both the pinger used by his target and a number of mixes in the network could observe the target's messages moving through his mixes by performing a key-swapping attack with the pinger. By providing the target with a public key other than the one generally available for the mixes he controls, the target's messages would be easily distinguishable when processed by his mixes.

6 Creating a Consensus Directory

The solution to the partitioning attacks mentioned in the previous section involves providing all clients with the same view of the network. Each pinger should calculate its results for the status of the network and compare these results with those of the other pingers. The pingers then must agree on a compromise result which reflects an accurate view of mix availability and provides every client with a consistent information with which to calculate message paths.

Since the infamous FLP impossibility proof [14], it is known that a simple problem such as reaching consensus in the presence of faulty parties – even if the worst they can do is to crash – is rather hard, and a significant amount of research has been put into securing distributed systems against faulty parties [24, 11]. In the mix-net scenario, the main parties for the agreement protocols are the *pingers*, i.e., the parties that maintain a database on the active mixes, their authentication keys and some of their properties. They need to agree on a consistent status of the *mixes* and then deliver it on request to the *clients*. As the clients should not be forced to contact more pingers than needed, each honest mix should be able to prove that the result it forwards to the client is in fact the genuine outcome of the agreement.

Depending on the attack model, the solutions can be rather complex, and in many implementations weaker attack models are chosen to allow for simpler protocols. In the Mixminion protocol, for example, the agreement protocol can easily be circumvented by one (or several) parties stopping the communication after a while – a condition that may be constructed even if none of the involved pingers is actually corrupt. The honest parties may recognize they are in a bad condition and alert the administrator, but it is left to human interference to actually recreate consensus; as the disagreement may be created without any party behaving obviously dishonestly, this may place quite some burden on the administration.

The protocol suite presented here will guarantee a consensus independent of timing assumptions, as long as fewer than a third of all pingers behave in an actively malicious fashion. For additional security, one can still add tests along the lines of the Mixminion protocol to detect inconsistencies and alert the administrators; however, if more than a third of the pingers are actively corrupt, the system is in a sufficiently bad state that its survival is questionable.

6.1 The Tools

In this section, we introduce the basic building blocks needed for our protocol. Due to lack of space, we do not give a formal model here. The protocols we choose were mostly developed within the MAFTIA project [3]; using new randomization techniques, these protocols are the first practical protocols that can deal with the maximum possible number of corruptions, and do not require any timing assumptions.

Asynchronous Binary Byzantine Agreement. The basic problem behind coordinating mutually untrusting parties is the problem of *Byzantine Agreement* [15]. Given a set of participants with different (binary) input values, a Byzantine agreement protocol allows parties to agree on one common output that has been proposed by an uncorrupted pinger [6], in spite of the presence of undetected traitors that try to disrupt the agreement and unpredictable network delays. For our application, we also require a transferable proof of the outcome, so one single pinger can prove to an external party what the outcome of a particular protocol instance was.

Verifiable Multivalued Byzantine Agreement. A multivalued Byzantine agreement protocol [5] extends the binary protocol to allow the pingers to agree on any bit-string, rather than only a binary value. As there is a (potentially) infinite universe of possible values, a multivalued Byzantine agreement protocol can no longer guarantee that the output of the protocol is the input of some uncorrupted pinger – this would only be possible if all uncorrupted pingers propose the same value. It is, however, possible to enforce that all pingers verify the value prior to agreement, and thus guarantee that it satisfies some properties, e.g., that it has the expected format, or contains proper signatures that certify its validity.

We will denote with $y = mult\_BA(x)$ the invocation of a multivalued Byzantine Agreement protocol with input x, yielding the output y.

Broadcast protocols. Broadcast protocols [5] are used to send messages from one sender to a number of receivers. In the simplest version, the sender simply sends the message to every other party (Note that in the Mixminion protocol, the receivers poll the messages, rather than the sender pushing them; for our purpose, this does not pose a significant difference). This simple protocol does not give any guarantees to the receivers, however; some may receive different messages, or no message at all.

A *consistent broadcast*, or c-broadcast, guarantees that all pingers that do receive a particular broadcast receive the same value [17]. It does not, however, guarantee that all pingers on the group receive anything in the first place.

A *reliable broadcast*, or *r-broadcast*, additionally guarantees that all pingers receive the broadcast [2]. Our model being asynchronous, there is no guarantee about the time; the only guarantee is that if one uncorrupted pinger receives the broadcast, eventually all other ones will.

An *atomic broadcast* [7, 23] primitive additionally guarantees that all uncorrupted pingers receive all messages in the same order. This is a rather powerful synchronization mechanism, that deals with many uncertainties of the asynchronous network and the attackers. In principle, it is possible to build the entire database on top of such a protocol; for this paper, however, we have chosen more efficient dedicated protocols.

Threshold signatures. Threshold signatures [26] allow pingers to issue shares of a signature, which then – given enough shares are available – can be combined into one single signature. The nice property is that a threshold signature outputs the same constant length signature, independent of the actual number of parties or the subset of parties that did the actual signing. This not only preserves space and bandwidth, but also allows for easy key distribution. A client does not need to know the public key of any individual pinger, nor the identity of the set of pingers, but can verify that a certain message was signed by a certain number of pingers by verifying against one single, static, public key. The disadvantage is that the internal management of the group of pingers becomes more complex. If an old pinger is disabled, its key share must be invalidated. Similarly, a new pinger needs to get a new key-share, and all thresholds need to adapt.[7]

We will denote with *tsign_generate(x)* the generation of a signature share of a threshold signature scheme, while *tsign_combine*($x_1, ...x_n$) combines n such shares into the final signature.

6.2 The Database Update Functions

Due to the different character of the data in the database, the pingers need four different protocols to maintain their databases in a consistent state.

Update set of mixes. The main functionality of our protocols is to maintain a consistent view about the set of mixes. Furthermore, a client should easily be able to obtain that set, i.e., each pinger can prove that he gives out the correct set. This is where the threshold signatures are used; there is only one signature for all pingers, but a minimum of two thirds are needed to generate the signature. Thus, a client only needs one public key to verify she got a correct set of mixes, without needing to know which parties are in the actual set of pingers.

As an input for the protocol, each pinger P_i has its own set $\mathcal{L}_i$ of mixes it considers valid. The protocol then has 2 phases. In the amplicifaction phase, each party collects the signed input of $n - t$ of other parties, and thus — as less than a third of all parties are corrupt — of at least $t + 1$ uncorrupted parties.

In the agreement phase, each party proposes its collection of inputs for a multivalued Byzantine agreement protocol, agreeing on one such set. It is possible that this set was proposed by a malicious party; however, as the initial sets

[7] The convenience afforded by the single threshold signature verification makes the threshold signature scheme, and its corresponding key data, a strong target for attack. Care must be taken that the implementation of the threshold signature scheme is performed securely, and that the algorithm itself is not weak.

where signed by the sender, any acceptable outcome set must contain at least $t + 1$ proper lists proposed by uncorrupted parties.

In the final step, a set of valid mixes is generated. To that end, each mix in the set of lists proposed by only t pingers is removed; thus, only pingers proposed by at least one honest pingers remain. As the final list contains the input of at least $t + 1$ pingers, every mix that is in the list of all uncorrupted pingers is guaranteed to be in the final set. The final list is then signed by a combined signature, allowing a party to prove that it is the proper list resulting from the protocol.

Protocol UpdateMixes for pinger P_i.

```
r-broadcast new (signed) list $\mathcal{L}_i$ of mixes.
wait for $(n - t)$ valid r-broadcasts.
let $\mathcal{L}' = \mathcal{L}_{j_1}, ....\mathcal{L}_{j_{n-t}}$ be a set of received $n - t$ lists.

$\mathcal{L}'' = mult\_BA\ (\mathcal{L}')$.
let $\mathcal{L}'''$ be the set of mixes in $\mathcal{L}''$ that have been proposed by $t + 1$ pingers.
$\sigma_i = tsign\_generate(date,\ \mathcal{L}''')$.

r-broadcast $\sigma_i$.
wait for $(n - t)$ such shares $\sigma_{j_1}, ..., \sigma j_{n-t}$.
$\sigma = tsign\_combine(\sigma_{j_1}, ..., \sigma_{j_{n-t}})$.

output $(date, \mathcal{L}''', \sigma)$
```

Fig. 1. The update protocol used by the pingers to obtain a consistent view of the mix network.

All subprotocols used in Figure 1 require a (expected) constant number of rounds and have an overal (expected) message complexity in $O(n^2)$, i.e., $O(n)$ messages to be send by each pinger. Consequently, the update protocol also terminates in (expected) constant rounds with a similar message complexity.

Update set of pingers. The protocol that updates the set of pingers is essentially the same as the one that updates the set of mixes. However, for this protocol it is important to also update the shared keys. This is to prevent the old parties from participating in any signing process, while the new parties need to get shares that allow them to participate. An appropriate re-sharing protocol is described in [4].

Update database (externally). With this function, the pingers can update information about a mix, most prominently its performance data. In the simplest case, this data is binary; in this case, a simple binary Byzantine agreement can be performed to determine a common value that has been proposed by at least one honest party. To avoid communication overhead, the agreements needed for all data on all mixes can be bundled; this leads to a protocol with message size linear in the total number of data items, and a running time and message complexity logarithmic in the number of pingers.

Update database (internally). This function is used to allow a mix to update database information about itself. Most commonly, this will be used to install a new key pair once the old one expires. It is relatively straightforward to implement this functionality, as the mix already has a key to authenticate itself. Assuming this is implemented properly (i.e., the signed messages are tagged properly), this can safely be used for database updates.

The update protocol would be a simple *r-broadcast* of a signed message requesting the update. This way, it is guaranteed that all pingers receive the same request, and the database stays consistent. To avoid race conditions, the mix also needs to maintain a serial number, so that all parties can be assured to receive all updates in the same order; due to the properties of the *r-broadcast* protocol, all pingers will eventually receive all update requests from the mix, so malicious manipulation of the serial numbers can only lead to a temporary inconsistency. Note that there exist protocols that can also enforce that all pingers receive all update requests in the same order; however, the those protocols are rather complex, so implementing this here may be overkill.

6.3 Attack Model

It is known that a simple problem such as agreement is quite hard in an asynchronous environment if some parties crash or otherwise do not follow the protocol. The only way around is to either rely on timing assumptions, or to use a randomized algorithm.

Our choice of implementation is for randomization, for two reasons: Firstly, the randomized model appears to be better adapted to the Byzantine setting, where the corrupted parties actively try to disrupt the protocol. Secondly, we can expect a realistic attacker in our scenario to launch denial of service attacks on the network, which timing based protocols have difficulties dealing with.

The price to pay for the fully asynchronous network model is a lower tolerance. It has been shown that even binary agreement is impossible once a third or more of all parties are corrupted [15]. In the mix-net scenario, the main parties for the agreement protocols are the pingers; they need to agree on a consistent status of the mixes and then deliver it on request to the clients. As the clients should not be forced to contact more pingers than needed, each honest mix should be able to prove that the result it forwards to the client is in fact the genuine outcome of the agreement.

6.4 The Functionality

The primitives we have described can be utilized by the mix-net's independent components to perform basic network maintenance operations. Mixes can announce their existence to a small selection of pingers. After the pingers perform several instances of the UpdateMixes protocol, all the pingers will have learned the address of the new mixes and independently confirmed their validity by sending the mixes a query which will result in the automatic return of their keys. After verifying a new mix exists, adding the mix's keys (which have been obtained directly from the mix itself), and confirming that the mix is properly forwarding packets by sending pings through it, the pingers will add the new mix to their list of known mixes. Once enough pingers list the new mix and its operational details, it will be included in the consensus directory. The current Echolot pingers are able to add and remove mixes without human intervention, and Leuchtfeuer has been designed to integrate with the current pinger behavior.

Leuchtfeuer restricts the information provided by pingers in one area where Echolot and the previously deployed pingers were unconstrained: in order to achieve consensus on the data associated with a mix, latency must be represented as one of a limited set of values, as opposed to being directly reported in units of time. Pingers should categorize individual mixes as being either "high" or "low" latency, and report them as such.

Pingers using Leuchtfeuer will record reliability and performance information about the mixes as they currently do, though the interval between publication of updates available to mix-net clients will increase significantly. As opposed to being updated every five minutes, as is the current default in Echolot, Leuchtfeuer pingers will create a threshold signature on the consensus directory and publish the signed directory for the clients every 12 hours. While this potentially increases the risk of lost packets due to a mix going offline immediately after a consensus directory in which it was still listed is published, it limits the ability of a passive attacker to perform intersection attacks based on short-term pinger result fluctuations.

Current mix-net clients do not perform any authentication on the data obtained by pingers, while in the Leuchtfeuer protocol, clients will need to verify the threshold signature to confirm that the consensus directory is authentic. This will not add noticeable additional complexity to the user experience.

7 Conclusions and Future Work

We have described a device called a pinger, a necessary component of anonymity networks based on unreliable mixes. As background, we presented a summary of the pinger software that has been created since the inception of the public anonymous remailer networks.

We have detailed a number of techniques used to ensure the results reported by a pinger are accurate and comprehensive, and emphasized specific technical requirements necessitated by the economic considerations of the public anonymous remailer networks. We have implemented and released our own pinger software, Echolot, which incorporates these techniques and as a result has become the dominant pinger solution used with the Mixmaster network.

We have designed an agreement protocol suitable for use in the asynchronous setting presented by the public remailer networks, which enables mutually-untrusting pingers to come to present a unified view of the state of the remailer network, including the names, network addresses, and public keys of the existing mixes, which can be authenticated by the mix-net client by verifying just one cryptographic signature on the consensus data. Our protocol greatly restricts an attacker's ability to exploit information about a user's information service or directory to perform intersection attacks against him, and reduces the impact that pingers operated by an adversary can have on the mix-net.

Acknowledgments

Peter Palfrader would like to thank Lucky Green and Colin Tuckley for authoring and continuing to update Echolot's end user documentation, Orange admin for work on the HTML templates that make up Echolot's output, and BiKiKii Admin, noisebox Admin and many nameless testers for providing valuable feedback during Echolot's development.

Len Sassaman would like to thank Bill Stewart, Nick Mathewson, and numerous attendees of the erstwhile San Francisco Bay Area Cypherpunks meetings for their insightful discussions of the mechanics of pingers. We also thank Meredith L. Patterson and Cat Okita for reviewing a draft version of this paper and for providing helpful comments.

This work 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. MySQL AB. Mysql. http://www.mysql.com/.
2. Gabriel Bracha. An asynchronous $[(n-1)/3]$-resilient consensus protocol. In *Proc. 3rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 154–162, 1984.
3. Christian Cachin, editor. *Specification of Dependable Trusted Third Parties.* Deliverable D26. Project MAFTIA IST-1999-11583, January 2001. Also available as Research Report RZ 3318, IBM Research.
4. Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *ACM Conference on Computer and Communications Security*, pages 88–97, 2002.
5. Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. Cryptology ePrint Archive, Report 2001/006, March 2001. http://eprint.iacr.org/.
6. Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. In *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 123–132, 2000. Full version available from Cryptology ePrint Archive, Report 2000/034, http://eprint.iacr.org/.

7. Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *Proc. Third Symp. Operating Systems Design and Implementation (OSDI)*, 1999.
8. David Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 4(2), February 1981.
9. cmeclax. pingstats, 2000 - 2003. http://ixazon.dynip.com/˜cmeclax/pingstats.html.
10. 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.
11. Yves Deswarte, Laurent Blain, and Jean-Charles Fabre. Intrusion tolerance in distributed computing systems. In *Proc. 12th IEEE Symposium on Security & Privacy*, pages 110–121, 1991.
12. Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.
13. Hal Finney. New remailer... Mailing list post, October 1992. http://cypherpunks.venona.com/date/1992/10/msg00082.html.
14. Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.
15. Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.
16. Raph Levien. rlist, 1995. ftp://ftp.zedz.net/pub/crypto/remailer/rlist.tar.gz.
17. Dahlia Malkhi and Michael K. Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, 1998.
18. Christian Mock. remlist – remailer config and ping tool, 2001. http://www.tahina.priv.at/˜cm/hacks/index.en.html.
19. Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster Protocol — Version 2, December 2004. http://www.abditum.com/mixmaster-spec.txt.
20. Peter Palfrader. Canonical List of All Pingers. http://www.noreply.org/allpingers/.
21. Peter Palfrader. Latency reports on Mixmaster remailers. http://www.noreply.org/latency/.
22. Peter Palfrader. Echolot: a pinger for anonymous remailers, 2001-. http://www.palfrader.org/echolot/.
23. Michael Reiter. Secure agreement protocols: Reliable and atomic group multicast in Rampart. In *Proc. 2nd ACM Conference on Computer and Communications Security*, 1994.
24. Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4), December 1990.
25. 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.
26. Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Advances in Cryptology: EUROCRYPT 2000*, volume 1087 of *Lecture Notes in Computer Science*, pages 207–220. Springer, 2000.