Echomix: a Strong Anonymity System with Messaging

Ewa J. Infeld   David Stainton   Leif Ryge   Threebit Hacker

Abstract—**Echomix is a practical mix network framework and a suite of associated protocols providing strong metadata privacy against realistic modern adversaries. It is distinguished from other anonymity systems by a resistance to traffic analysis by global adversaries, compromised contacts and network infrastructure, quantum decryption, and statistical and confirmation attacks typical for multi-client messaging setting. It is implemented as Katzenpost, a robust software project, and used in multiple deployed systems, and features relatively low latency and bandwidth overhead.**

The contributions of this paper are: (1) Improvements on leading mix network designs, supported by rigorous analysis. These include solutions to crucial vulnerabilities to traffic analysis, malicious servers and active attacks. (2) A cryptographic group messaging protocol with strong metadata protection guarantees and reliability. (3) Hybrid post-quantum nested packet encryption.

1. Introduction

Protecting metadata is as crucial a concern for privacy as protecting the content of communications. They are a primary stock of data brokers [1] and qualifier in large dataset analysis. Surveillance actors take advantage of metadata [2]–[4] even to exert lethal force. [5], [6]

Anonymity systems in use today offer incomplete protections against the most powerful class of surveillance adversaries. The research field of anonymity is robust, yet many academic designs disregard real-world Internet conditions, or explicitly declare as non-goals the resistance to a wide variety of practical attacks on user data and metadata. Second-party anonymity is seldom considered.

We describe a novel, implemented, practical and reliable mix network design, with a threat model that allows for sophisticated, global, active adversaries who may compromise network elements and a user's contacts, have access to a quantum computer, and do powerful cryptanalysis. We overcome weaknesses of leading anonymity systems, and introduce protocols which provide strong security guarantees even in the case of persistent messaging between users. The design and software is used in a growing number of deployed systems, [7], [8] as well as our own chat client. [9]

This is in contrast to systems such as Tor [10], which does not protect against a global adversary [11]–[13] and is vulnerable to an array of confirmation and traffic attacks [14], [15], some of which have been exploited by surveillance actors [16]. The leading mix network model today is Loopix [17], which is a basis of Nym's system [18]. Our design eliminates its many shortcomings, including vulnerability to traffic analysis, receiver observability, and vulnerability to malicious service providers. We support these claims with rigorous analysis.

We additionally introduce a messaging protocol which is suitable for anonymous group messaging with a realistic threat model, and provides reliability without forcing interactivity. We then present a quantum resistant packet format appropriate for mix networks. Finally, we provide latency and bandwidth overhead evaluation, to demonstrate that this system is practical.

2. Threat model

We consider a realistic modern adversary, such as a government surveillance agency, a large technology corporation, or a criminal organization. The adversary is:

Global. The adversary can see all or a significant portion of connections of the entire global internet and is capable of statistical analysis of gathered data. For many attacks that are typically attributed to a global adversary, it is enough if the adversary has a view of a target population of users of the network.

Active. The adversary can disable parts of the network, and plant or take over some devices in the network to inject malicious code and gain access to the information available to them. This can happen by technical means, exercising legal or extralegal forms of coercion, or subterfuge. The adversary can compromise a client's contacts' devices, resulting in a need for *second-party anonymity* in the system. We refer to a compromised contact as a second-party adversary, or 2PA. We minimize metadata shared with contacts and avoid various forms of forced interactivity such as automatic delivery and read receipts.

Many attacks typically attributed to an active adversary are also a concern with a passive adversary able to observe network disruption events. If a user's microwave oven turns on and causes a brief connection disruption for their WiFi, that event can be visible to various unrelated internet services which know the user's identity. In particular, connection disruptions must not be revealed to contacts.[1]

Echomix is not secure against an active adversary who compromises the entire system, or close to the entire system, such as a majority of directory authorities or a critical combination of node types on the client's path, as described in section 6.

Sophisticated. The adversary has large computational resources, and is capable of cryptanalysis on par with frontier research. The adversary has access to a quantum computer, or will have access to one in the near future.

Has context. The adversary can supplement collected data with rich context of already gathered data on all users from other sources.

---

1. These network disruption confirmation attacks have been used by surveillance actors. [19], [16] In particular, [19] is an example of a 2PA.

We will define *strong anonymity* for classification purposes as an ability to withstand a sophisticated, **global passive adversary (GPA)**. As demonstrated, the Echomix threat model goes further to include active adversaries of significant, but not absolute, power.

We define metadata broadly, to include all observable distinguishing characteristics of a user's activity. This can be sender and receiver identity, timing information, volume and type of communication, social network etc.

2.1. Anonymity systems in light of the threat

Several existing types of communication systems offer anonymity protections. We survey these briefly and note weaknesses with respect to our stated threat model.

2.1.1. VPNs. VPNs relay a user's connection while minimizing latency and maximizing bandwidth. Traffic can be correlated by a passive adversary observing network traffic at the the VPN, or the source and destination traffic.

Some VPNs are overlayed on mix networks and/or employ decoy traffic [20], [21]. The strongest of these designs attempt to transport VPN traffic with uniform bandwidth by clients, necessitating both a large bandwidth overhead and an upper bandwidth limit. However, Internet protocols facilitated by VPNs allow for confirmation attacks through forced interactivity, and so interruptions to client traffic can be correlated, either by passive observation or active interruptions of clients.

2.1.2. Tor. Tor is much more sophisticated than a VPN, and does not rely on a single or few points of failure. In the absence of a GPA, we consider Tor to be state-of-the-art for practical anonymous communication and it is able to provide users with an experience of browsing the Internet comfortably. However, a passive observer able to see two endpoints can correlate connections [11]–[13], [16]. As with VPNs, because Tor is a general purpose tool used to transport protocols which force interactivity, we consider it impossible to extend Tor's threat model to include protection against a GPA. [14], [15]

2.1.3. Mix networks. Mix networks, or *mixnets*, are an anonymous communications network paradigm distinguished by striking a balance between practicality and protection from strong adversaries. A mix network consists of network devices, typically referred to as mix nodes, that relay messages between clients in such a way that an adversary is unable to determine which clients are communicating with each other. This is done by reordering or mixing multiple indistinguishable messages.

There must be some trade-off in latency - packets must not be forwarded from a node immediately, in order to be *mixed* with other packets, and some trade-off in bandwidth - packets must be padded to a common size. They also have to be re-encrypted at each hop. [22] The notion that both latency and bandwidth overhead must be non-zero was formalized in [23]. Additional trade-offs occur with adding common anonymity strategies, such as decoy traffic. It is thanks to these combined strategies and trade-offs that carefully constructed mix networks can offer protection against global adversaries. Because of these trade-offs mix networks are not able to simulate browsing

the Internet comfortably as a VPN might, and are typically considered for use with some services, such as messaging.

Many mix network designs and protocols have been published, starting in 1979 [22], and we observe a rise in commercial efforts to build mixnets [7], [18], [24]. Much of the focus of these efforts is in incentivisation mechanisms for mix node operators, and financial privacy. The largest of these, Nym [18], is based on the Loopix model. [17] Due to a subtle oversight in design, Loopix is vulnerable to a GPA as described in subsubsection 3.4.3. It also has a single point of failure in a user's Service Provider, making it easy for an active adversary to compromise a target user.

2.1.4. Non-mixnet theoretical systems with strong anonymity. There exist theoretical designs providing strong anonymity, some of which have information-theoretic guarantees, which Echomix does not. They all carry significant overheads which have so far made them impractical for real world deployment at a scale. These include DC-nets and other k-anonymity-based systems [25]–[28], as well as PIR [29] designs.

3. The Echomix design

The Echomix system contains three server node types: gateways, mix nodes and services. None of these has a persistent relationship with clients. Service nodes are positioned on the far side of the network, and, in contrast to leading mix network designs, every interaction with them is a round trip - an *echo*. When a client connects to a random gateway, the Sphinx [30] packets sent by the client are relayed through the three layers of mix nodes, and then to a service. A reply is sent, which can confirm delivery or contain send query results. A service can send this reply without knowing the location of the client thanks to a Sphinx Single-Use Reply Block (SURB).

Figure 1: A client's interaction with the service node is a round-trip, with a packet's forward route marked in purple, and service's confirmation in green. The intermediate node layers are pictured from top to bottom.

In the context of a messaging application, a message travelling from Alice to Bob requires two *echos* to the far side of the network, one from Alice to write a message to a service, and a second initiated by Bob to retrieve it. This symmetry and a careful construction of protocols, allows us to out-perform other anonymous communication systems in resisting traffic analysis, malicious providers, and confirmation attacks.

The clients generate a stream of *echos* to uniformly random, or pseudorandom - in the case of some applications, including message streams - services, independently of whether a client requests a service or not. Most of these packets will be decoy traffic. The stream is a single Poisson process, with delays at each hop sampled from an exponential distribution, as in [31] and [17]. The advantages of this strategy are described in section 3.4.1. Unlike in Loopix [17], the amount of sent and received traffic is fully independent of whether the packets are application traffic or decoys.

From the point of view of network architecture, Echomix, and its implementation - Katzenpost [32], is an Internet overlay mix network atop TCP/IP or QUIC/IP. The subsequent layers are governed by the following protocols.

1) PQ Noise [33] transport protocol.
2) Sphinx [30] or PQ-Sphinx routing protocol.
3) Application layer, such as Pigeonhole messaging.

Katzenpost is the first software implementation of the PQ Noise. As a transport protocol it enforces the network topology, e.g. mix nodes in layer 1 are only allowed to downstream-connect to the mix nodes in layer 2. All PQ Noise messages are padded to a uniform size.

Since all application protocols are built based on packet round-trips to services positioned on the far side of the network, the Sphinx ability for the service to send a reply without knowing the location of the client is used throughout the design. This is done with Single Use Reply Blocks (SURBs). We describe our post quantum updates to the Sphinx format in section 7.

The Katzenpost PKI is an adjacent protocol at the root of authority within the mix network. Directory authorities publish PKI documents every epoch, distributing network connection information and public key materials to the nodes. This system is similar to Tor's. [34] Providing all nodes and clients with a uniform view of the network allows us to resist epistemic attacks. [35] An adversary who compromises a majority of directory authorities compromises the entire mix network. We therefore employ a decentralised PKI design, relying on multiple independent directory authority operators. The consensus producing the PKI document uses the post quantum hybrid signature scheme of Ed25519 [36] combined with Sphincs+ [37].

3.1. Gateway nodes

In contrast to [17], we believe that the nodes on the edge of the network should have as little information on the client behavior as possible, as they are the ones that can identify the client. Gateways have no persistent relationship with the client, and no knowledge of whether any of the client's packets are decoys or not.

If the client traffic crossing the node is low, the gateways additionally generate decoy traffic. We propose a new heuristic, the *Coupon Collector's bound* in order to ensure that all links of the network are active, resulting in a reliable mixing of packets.

Coupon Collector's Bound. *If packets released from a node are directed to one of the nodes in the next layer uniformly at random, the expected number of messages that node has to release before at least one message*

has been directed to each node in the subsequent layer behaves like $\Theta(n \log(n))$.

Proof: This is a direct consequence of the Coupon Collector's Problem. □

Let:
- $\mu = 1/\lambda$ be the mean time that a packet lingers in a node due to memoryless mixing,
- n be the maximum number of nodes in a layer,
- g be the number of gateways.

We aim to achieve a high probability that during each period μ, there is at least one packet crossing a given link in the network. The Coupon Collector's Problem tells us how many packets we need to cover links from the gateways to the first layer of mixes. In order for all links between two layers of size n to be reliably active, we should additionally multiply the desired output of a gateway by n/g. Therefore in order for all links to be active with high probability, and therefore for best mixing of the packets in the network, each gateway should be outputting at least an average of $\Theta(n^2 \log(n)/g)$ packets in the time μ. In practice, this bound is significantly lower than the number of packets travelling through the nodes.

3.2. Decoy traffic and application traffic

An *echo* decoy packet is a Sphinx round-trip through the mix network to a service node sampled uniformly at random. A service request to a server sampled in a way indistinguishable from uniformly random, followed by a service response sent back with the SURB, is unobservably coupled to echo decoy traffic.

Traffic coupling. *Let sequences A and B be two probabilistic processes on states $S = \{s_1, s_2, \ldots, s_n\}$. If both A and B are indistinguishable from uniformly random, then any process C which selects either A or B to sample the next state according to any algorithm $\mathcal{A}$ is also indistinguishable from uniformly random.*

Proof: Let $\mathcal{H}$ be any state history in C, and $s_j \in S$ be any state. Then for any values of $p = \mathbb{P}_{\mathcal{A}}](A|\mathcal{H})$ and $p' = \mathbb{P}_{\mathcal{A}}(B|\mathcal{H})$, the probability of the next element being from sequence A and sequence B respectively, the probability of the next sampled state being s_j is

$$p \times \mathbb{P}_A(s_j|\mathcal{H}) + p' \times \mathbb{P}_B(s_j|\mathcal{H}) = (p + p') \times \frac{1}{n} = \frac{1}{n}.$$

$\mathcal{A}$ *does not have to be independent of history $\mathcal{H}$.* □

If and only if we maintain the correct coupling, the user's application traffic is fully unobservable within decoy traffic, and causes no anomalies, avoiding the shortcomings of leading systems described in subsubsection 3.4.3. For receiving messages, maintaining unobservability requires care, since not only are the queries correlated with the related send requests, but we may have to contend with the dangers of multiple queries to the same server, which may be statistically significant and therefore visible to a passive observer. The introduction of *couriers* (section 5) allows us to efficiently maintain unobservably coupled traffic. It also prevents *SURB floods* - an attack in which a malicious service might collect multiple linkable SURBs and send them in a burst in order to identify a client.

3.3. Service nodes

Service nodes are positioned behind the mix network and handle functionality requested by the client. This could be storing messages, publishing information outside of the mixnet, interfacing with a blockchain node etc. They also process and execute the SURBs of decoy packets. Any application should be constructed so that the following conditions are met:

1) Separate service requests of a client are unlinkable. Repeating the same request may be linkable.
2) Services are treated uniformly by the client, with no persistent relationship.
3) The traffic from a client to the service is correctly coupled with the decoy traffic.

The last condition means that either the service is chosen in a way that is independent from traffic history and indistinguishable from uniformly random for each query to a service, or the packet will replace a decoy packet that was meant to go to the specific service. Since the latter introduces additional latency, we achieve the former in several ways in sections 4 and 5.

3.4. Iterating on prior research

So far, we described original additions to the field of mix network design. For the remainder of this section, we will discuss how we implement and improve upon established concepts, and motivate the new design by pointing out vulnerabilities of preceding systems. Beginning with section 4, we list remaining original contributions which allow us to extend our security guarantees to persistent messaging, and resisting quantum adversaries.

Katzenpost grew out of the Panoramix project [38], and was previously an evolution of Loopix [17], which combined ideas of sampling an exponential distribution (memoryless mixing) to delay at each hop [31], monitoring system health with heartbeat traffic [39], organizing nodes in a layered topology [40], assigning persistent Service Providers for each user at the edge of the network, and wrapping data in Sphinx packets [30], a packet format designed specifically for mix networks.

3.4.1. Memoryless mixing. Echomix adopts node delays sampled from an exponential distribution, as introduced in [31] and used in Loopix [17]. This distribution has the advantage of being *memoryless* - at each point in time, each message sitting in a mix will have the same probability distribution of the remaining delay, independently of how long it has already been waiting. This means that for an external observer the probability distribution of which message will be sent next is uniform at all times. For a constant parameter $\lambda > 0$ with the mean $1/\lambda$, the delays approximate the probability distribution function:

$$f(x_{\geq 0}, \lambda) = \lambda e^{-\lambda x}.$$

However, [31] claims incorrectly that the behavior of a resulting mix node is Poisson distributed. This is then reproduced in [17], culminating in nodes of these type being called *Poisson mix*es. The behavior of a sum of multiple Poisson processes is in fact only Poisson if the set of processes being summed doesn't change over time.

We propose to call this type of mixing *memoryless mixing* instead, while noting that the overall behavior of the node itself is not memoryless and a node should not be called a *memoryless mix*.

3.4.2. Heartbeat packets. All nodes in the mixnet generate *heartbeat packets* [39], which are sent through the system before returning to their origin. This allows each node to take stock of the functioning of the network - if any segment of the route is disrupted, packets crossing through that segment will not come back in the expected time window. This allows the system to detect an $n-1$ attack [41], or any other attack that involves a disruption of the network. However, heartbeat packets not originating from the clients are only effective in detecting faulty or malicious behavior in some mix nodes. Malicious nodes on the edge of the network can distinguish between mix heartbeats and user requests and decide to forward any heartbeat packet that is scheduled to return to a mix node, while dropping user traffic meant to exit the network.

Additionally, no node or client in the network should act on these measurements by itself. Such a process could result in different clients having different views of the network and opening the system to *epistemic* [35] and *compulsion* [42] attacks. Instead, nodes in Echomix process the collected measurements [43] and rate the links' health, and then upload the ratings to directory authorities. The directory authorities establish a consensus and distribute the updated structure of the network to all parties in the next PKI document, once per consensus epoch.

Finally, both [39] and [17] describe rating nodes in the mixnet. It is more helpful to rate links on the route, rather than nodes. Not only does it provide finer data, and allow for detection of $n-1$ attacks directly, but in practice many networking problems happen specifically between two locations. A separate argument in favor of indexing over links can be found in [44] for batch mixes.

3.4.3. Vulnerability to traffic analysis in Loopix. This Loopix design vulnerability has not been addressed elsewhere. A legitimate message traveling from the last layer of mixes is sure to go to the receiver's designated provider, as opposed to decoy traffic from the last layer of mixes, which is uniformly distributed among providers. Application traffic at the last hop is therefore independently overlayed on decoy traffic.

Figure 2: In Loopix, as Alice communicates with Bob, the increase in traffic to Bob's Provider is observable.

This means that application traffic at that hop is observable to a passive network adversary, as long as it

is statistically significant, and especially if there is any regularity to it. In this situation, the other traffic at this hop is not an effective cover, it's noise, and in many real-life situations a signal processing analysis could easily do away with it. This is dangerous even in the short term, since the low latency between the sender and the receiver's Provider opens this system to correlation attacks and SURB floods, and any advantage that comes from asynchronicity is lost.

A naive solution if one wanted to retain the rest of the Loopix design might be to wait until a decoy packet is going to the right provider, and send the message instead. This would significantly reduce the available bandwidth by effectively dividing it by the number of Providers. In sections 4 and 5, we carefully designe a correct coupling of decoy and application traffic which does not reduce the bandwidth.

3.4.4. Providers. Loopix features persistent *Service Providers* that the clients connect to directly. The Provider is a significant point of failure, with access to a trove of a user's information, including the message receiving pattern. It was suggested in [17] that returning heartbeat traffic may double as decoy traffic for received messages, but this is incorrect. These streams are independently overlayed, and can be similarly decoupled as above when a client is online. When the receiver is not online, loop traffic is not present at all. In Echomix we only include gateways on the perimeter of the network, with service providers positioned behind it, and do not include persistent providers or gateways for a user at all. Interactions of the same user are unlinkable by the service provider.

3.5. The challenges of persistent multi-client messaging

Persistent messaging between multiple clients comes with an additional set of statistical and active attacks, and a need for second party anonymity. The design described so far is appropriate for a wide array of applications, including one-to-all anonymous publishing, interfacing with blockchain nodes, and other sender-only use cases. It is a significant improvement on the security properties of previously published mixnet designs. In order to further extend it to persistent messaging, we introduce the BACAP and Pigeonhole suite of protocols to allow for messaging while maintaining our security guarantees. These are described in sections 4 and 5.

4. BACAP

We have established in subsection 3.3 that we require services to be accessed by clients uniformly at random, or in a way indistinguishable from uniformly random. We also require multiple interactions of the same client to be unlinkable by a service. But if users are to be able to exchange messages, they need to know where from and how to retrieve them. Our goal is therefore to implement a form of *private distributed hash table*, wherein users can leave each other messages in pseudorandom locations, with servers able to verify the validity of the message, and read ability, without being able to determine that two messages belong to the same conversation.

BACAP (Blinding-and-Capability scheme) allows us to deterministically derive a sequence of key pairs using blinding, built upon Ed25519 [36], and suitable for unlinkable messaging. It enables participants to derive *box IDs* and corresponding encryption keys for independent, single-use *boxes* using shared symmetric keys.

A box consists of an ID, a message payload, and a signature over the payload. There are two basic *capabilities* - one that lets a party derive the box IDs and decrypt the messages, and one that additionally lets the holder derive private keys to sign the messages. The signatures are universally verifiable, as the box ID for each box doubles as the public key for the signatures.

In the context of a messaging system, the protocol is used by Alice to send an infinite sequence of messages to Bob, one per *box*, with Bob using a separate, second instance of the protocol to send messages to Alice. Alice will use a root private key to derive a root public key shared between participants. The root key and a CSPRNG instantiated from recursive KDF applications are then used to obtain a sequence of context-specific values for exercising and verifying a capability. A context value ctx, which is a hash of a universally public value, will be used as additional input. It can, for simplicity, be a hash of the name of the storage network, or can be bound to a specific period of time, e.g., the *long epoch SRV* published by the Echomix directories at regular intervals, similar to how Tor uses its *SRV* in [45]. The context value makes it safe to *unlinkably* relocate messages to a different network.

All parties (and adversaries) know public constants:

- B the ed25519 base point
- ℓ the prime defined in [36]
- ctx hashed network context

In the section below we will use the syntax $B \cdot x$ to denote "scalar multiplication" of the point B and scalar x, and $a \times b$ to denote natural number multiplication. The following values are generated by Alice and constitute her *"write capability"* for the sequence:

- $S_R \in \mathbb{Z}_\ell$: root private key,
- $P_R = B \cdot S_R$: root public key.
- $i_0 \in \mathbb{Z}_{2^{63}}$: random initial index counter.
- $H_{i_0} \in \mathbb{Z}_{2^{256}}$: random initial KDF state.

Alice sends the following *read capability* to Bob out-of-band:

$$P_R, \ H_{i_0}, \ i_0.$$

Our i will be encoded as an unsigned 64-bit integer, and serves to define an ordering for boxes and to enable applications to refer to boxes uniquely. Initializing it with an upper bound of 2^{63} ensures that a sequence can contain at least $2^{64} - 2^{63} = 2^{63}$ boxes. We define no mechanism for extending sequences to more than 2^{63} boxes, but applications could use a box to communicate a new read capability if such an extension were required.

Both Alice and Bob can now derive a sequence of KDF symmetric keys H_i, location blinding factors K_i, symmetric payload encryption keys E_i, box IDs M_i, and increment $i \rightarrow i + 1$. The i is used as additional input in the H_i KDF as control input to reduce the risk of incorrect implementations generating valid KDF outputs for incorrect i indices.

$$H_i, \ i \underset{\text{KDF}}{\to} H_{i+1}, \ E_i, \ K_i.$$

Both parties generate the message encryption key E_i^{ctx}:

$$E_i \ , \ \text{ctx} \underset{\text{KDF}}{\to} E_i^{\text{ctx}},$$

and the blinding factor K_i^{ctx}, and the blinded public key M_i^{ctx} which will serve as the *box ID*:

$$K_i \ , \ \text{ctx} \underset{\text{KDF}}{\to} K_i^{\text{ctx}},$$
$$M_i^{\text{ctx}} = P_R \cdot K_i^{\text{ctx}},$$
$$\equiv B \cdot S_R \cdot K_i^{\text{ctx}}.$$

Alice derives a M_i^{ctx}-specific secret scalar S_i^{ctx}, and encrypts message m_i as ciphertext c_i^{ctx} using key E_i^{ctx}, and signs it as s_i^{ctx} using S_i^{ctx}. This signature ensures unforgeability (EUF-CMA) from adversaries that possess a read capability for the sequence, enabling the use of BACAP in group settings with multiple readers:

$$c_i^{\text{ctx}} = \text{AES-256-GCM-SIV-ENCRYPT}(m_i \ , \ E_i^{\text{ctx}}),$$
$$S_i^{\text{ctx}} = S_R \times K_i^{\text{ctx}} \pmod{\ell},$$
$$s_i^{\text{ctx}} = \text{Ed25519-SIGN}(c_i^{\text{ctx}} \ , \ S_i^{\text{ctx}}).$$

Alice sends her *message* to the server:

$$M_i^{\text{ctx}}, \ c_i^{\text{ctx}}, \ s_i^{\text{ctx}}.$$

The server verifies the *write capability* to ensure it was sent by a sequence *writer*, as opposed to a *reader*.

$$\text{Ed25519-VERIFY}(M_i^{\text{ctx}}, \ c_i^{\text{ctx}}, \ s_i^{\text{ctx}})$$

Bob requests M_i^{ctx} from the server, verifies and decrypts:

$$\text{Ed25519-VERIFY}(M_i^{\text{ctx}}, \ c_i^{\text{ctx}}, \ s_i^{\text{ctx}}),$$

$$m_i = \text{AES-256-GCM-SIV-DECRYPT}(c_i^{\text{ctx}}, E_i^{\text{ctx}}).$$

The original Ed25519 [36] signing algorithm works on *private keys* that are not scalars, but SHA-512 hash preimages created as part of the signing operation. We refer to a modified Ed25519 signing algorithm that skips the hash step and instead operates directly on private scalars as Ed25519-SIGN.

4.1. Choice of encrypt/decrypt functions

Using an authenticated symmetric encryption scheme prevents a *third-party* quantum adversary from forging c_i^{ctx}, separate from the s_i^{ctx} signature, the private key for which is obtained by the adversary. Such an adversary can forge signatures over garbled ciphertexts, but not plaintexts, and can't authenticate the ciphertexts. We use an authenticated cipher scheme, AEAD_AES_256_GCM_SIV [46] (using M_i^{ctx} as *nonce*), to make implementation less error-prone, and to enable replacing a c_i with a *tombstone*, a term we will define in subsubsection 5.4.3.

4.2. Forward-security

BACAP achieves *computational post-quantum forward-security* by the irreversibility of the KDF function. Parties may choose to leverage this by throwing away state associated with H_{i-1} once done with it. Instead of sending the *read capability* at i_0, Alice may choose to instead reveal a later $\{P_R, H_{i_n}, i_n\}$ which would only enable a newcomer Charlie to read the sequence starting from i_n. We sample the original i_0 randomly instead of starting at 0, to avoid indirectly revealing to Charlie how many boxes preceded i_n.

We separate E_i and K_i, as Alice may want to keep K_i to be able to derive S_i^{ctx} at a later point, for example to sign a message to delete the box, without being able to decrypt the message payload encrypted under E_i^{ctx}. Separating the two permits a protocol instantiation to selectively have forward-security only for E and the resulting c ciphertexts.

The knowledge of $S_i^{\text{ctx}} \equiv S_R \times K_i^{\text{ctx}} \pmod{\ell}$ and the factor K_i^{ctx} makes it trivial to recover

$$S_R \equiv S_i^{\text{ctx}} \times (K_i^{\text{ctx}})^{-1} \pmod{\ell}.$$

As a result, sharing the derived signing keys with third-parties is not safe because the recipient can recover the effective signing key for the whole sequence.

4.3. Unlinkability

Suppose that an adversary $\mathcal{A}$ knows the values in BACAP available to the server. Let X be the event that box IDs M and M' belong to the same sequence $\{M_i^{ctx}\}_i$, and X' that they don't. Let $X_{\mathcal{A}}$ be the event that the adversary guesses that they do. Then M and M' are *unlinkable* iff

$$|\mathbb{P}[X_{\mathcal{A}}|X] - \mathbb{P}[X_{\mathcal{A}}|X']| \leq \delta,$$

for some sufficiently small $\delta \geq 0$.

It is not possible for anyone without *read capabilities* to determine whether two messages in different boxes belong to the same sequence. The symmetric encryption of m_i under key E_i^{ctx} (unknown to adversaries) results in *unlinkability* even against *chosen-plaintext attacks*.

The messages in BACAP remain unlinkable to a quantum adversary. While the security of the Ed25519 signing scheme, and thus the distinction between BACAP *read* and *write* capabilities, relies on the hardness of the elliptic curve discrete logarithm problem (ECDLP),[2] the *unlinkability* does not:

Solving ECDLP for B in $M_i = B \cdot S_R \cdot K_i^{\text{ctx}}$ yields a scalar $S_R \times K_i^{\text{ctx}} \in \mathbb{F}_\ell$ (the effective signing key) for each box M_i^{ctx}. An adversary who knows S_R or a K_i^{ctx} can trivially compute the modular multiplicative inverse, but because S_R and K_i^{ctx} are independent pseudo-random elements of $\mathbb{F}_\ell$, there is no unique solution to the equation $M_i^{\text{ctx}} = S_R \times K_i^{\text{ctx}} \pmod{\ell}$ that lets the adversary solve for S_R. Such a solution would enable them to link $M_x^{\text{ctx}}, M_y^{\text{ctx}}$. Therefore, the *post-quantum unlinkability* of BACAP relies on:

1) The indistinguishability of $K_i^{\text{ctx}} \pmod{\ell}$ from uniformly random elements of $\mathbb{F}_\ell$ with negligible bias. That is, an assumption that the KDF is secure, and that the reduction $\pmod{\ell}$ has negligible bias.

---

2. Solved with Shor's algorithm on a quantum computer.

2) The computational difficulty of enumerating the K_i^{ctx} (keyspace roughly 2^{256}).
3) The property that for any S_R guess, enumeration of $H_0, \text{ctx} \to K_x^{\text{ctx}}, K_y^{\text{ctx}}$ is required to find two Ks such that $M_x^{\text{ctx}} = B \cdot S_R \cdot K_x^{\text{ctx}}$ and $M_y^{\text{ctx}} = B \cdot S_R \cdot K_y^{\text{ctx}}$.

4.4. Post-compromise security

A basic implementation of BACAP does not provide post-compromise security. A simple way to achieve post-compromise security would be to rotate BACAP sequences frequently. It is worth noting that deriving new H_n, i_n from a KDF (without communicating a new S_R/P_R) provides *post-compromise security* with respect to *unlinkability* since the adversary would still not be able to link K_is or obtain E_is.

A quantum adversary can impersonate writes by solving ECDLP for $S_R \times K_i^{\text{ctx}} \pmod{\ell}$ (but still not obtain new K_i^{ctx}), so nothing new is learned. In the classical setting, adversary knowledge of a compromised S_R does not help the adversary obtain the K_i^{ctx} required to compute $S_R \times K_i^{\text{ctx}} \pmod{\ell}$.

5. Pigeonhole storage

The goal of this section is to extend our system to provide messaging functionality expected by today's Internet users without compromising our security goals, and to futher strengthen our resistance to service nodes being compromised. This is achieved with Pigeonhole servers, which provide a time-limited storage capacity to implement asynchronous, unidirectional, single-writer, multi-reader BACAP messaging channels with strong metadata protection for clients against both passive and active adversaries, including authorized readers who are malicious or compromised. As Alice exchanges messages with Bob, they rely on pseudorandom shared sequences of BACAP boxes as storage locations. Boxes in the same sequence are linkable to users with read or write capabilities for the sequence, but are cryptographically unlinkable to the storage servers thanks to the properties of BACAP.

5.1. Additional concerns in interactive messaging

5.1.1. Increased vulnerability to statistical disclosure. Messaging comes with a number of statistical pitfalls. These include user behavior being vulnerable to correlation, the number of queries to particular boxes revealing that they belong to the same BACAP sequence with the same number of readers, and the difference in time between writes and reads. In the case of two users being more likely to be online at the same time if they are talking to each other, little can be done unless users themselves are mindful of the correlation potential and able to adjust their habits. In the case of other correlations, *pigeonhole services* employ a number of powerful mitigating tactics.

5.1.2. Reliability vs forced interactivity and lossiness. A mix network can be expected to drop a small number of in-flight messages to load-shed [47]. For applications requiring reliable delivery, this necessitates a reliability mechanism. There is a tension between reliability and

metadata protection due to the information that is revealed by sending acknowledgements or retransmitting missing information. Expecting a user to acknowledge receipt of messages might make her vulnerable to confirmation attacks and correlation. Some mixnet-based systems [18] automatically retransmit unacknowledged messages. This occurs transparently (to the user/sender) and repeats until acknowledgement is received. Others, like Karaoke [48] acknowledge the risk of repeated events, and terminate the users' conversation if message loss is detected.

The tension applies not only to the short-term reliability needs of ensuring that a write from a user to a service node was received, but also to the long-term reliability problem where a writer needs to retransmit older writes based on the their perception of the reader's state. In subsection 5.4 we address both facets of this problem with *couriers*, which are aware of retransmits but are not aware of the box IDs they are associated with, and we're able to provide reliability without automatic acknowledgements.

5.1.3. Long-term channel resilience. If a BACAP message becomes unavailable before the intended recipient(s) retrieves it, the recipient may continue trying to read the message forever. Messages can become unavailable either due to server failure, or as part of scheduled garbage collection to make room for new messages to be stored. To ensure *eventual consistency*, we will introduce a mechanism to redeliver messages with *all-or-nothing backfills*.

5.2. All-or-Nothing Design Philosophy

Suppose a user sends two or more related messages. If each message can fail independently, and the adversary (such as the contact) observes some messages appearing, it alerts them to the intention of sending additional messages. The failure of these messages to be observed can be correlated with a physical connectivity issue. We follow a principle that actions should either *succeed completely* or *fail unobservably*, and describe our solutions to these problems in subsection 5.6.

5.3. Replicas and sharding

To further mitigate the risks of service nodes in a messaging application being compromised, we split their functionality between *storage servers* or *replicas*, and *couriers*. Couriers maintain fixed-throughput connections to replicas, as do replicas with each other.

Figure 3: Replication is marked in blue, Alice's write operation green, Bob's read operation red, and Couriers' fixed-throughput connection to the replicas in purple.

Suppose we have n pigeonhole storage servers, or *replicas*, and we want a subset of k of them to store each box. The storage system is sharded using a consistent hashing [49] scheme which allows entities with knowledge of a the box ID (BACAP's M_i^{ctx}) to deterministically select the two servers that are currently responsible for that box. The k servers responsible for storing a given box are then that box's *replicas*.

The consistent hashing method makes this efficient. For a given box ID, one derives a set of k servers by sorting the list of servers by a hash of their public key concatenated with the box ID and use the result to select a permutation of the set of n servers for each box ID. We can then choose the first k of them. This results in a set of k designated replicas indistinguishable from being chosen uniformly at random. When a storage server goes offline, or a new server joins only proportion of k/n boxes need to be retransmitted.

5.4. Couriers

The second type of service, couriers, can be seen as a mix network service layer and are responsible for acknowledging the client's requests, but do not learn the box ID. They communicate with the replicas over fixed-throughput direct connections outside the mix network and avoid revealing the existence of a retransmission to the storage server.

Commands sent through couriers are encrypted by the client to the target replicas' NIKE keys, which are published in the PKI and rotated each epoch to provide forward secrecy. This encryption can optionally be post-quantum if the NIKE key is a hybrid NIKE as is used in our PQ NIKE Sphinx construction described in subsubsection 7.1.1.

5.4.1. Writing messages. To store a BACAP message in the network, Alice first generates an ephemeral NIKE keypair which will be used for the envelope, and a symmetric encryption key ("envelope key") which will be used to hide the envelope contents from the courier. She then randomly picks a *courier*, and two *intermediate replicas* that will receive write operation from the courier. The *intermediate replica* that receives an envelope decrypts it, commits it to disk, and acknowledges to the courier that the replica takes responsibility for delivering the message to the final replicas. The courier then sends an acknowledgement to the client.

These *intermediate replicas* are chosen independently of the two *final replicas* for that box ID which are derived using the sharding scheme. The reason Alice designates *intermediate* replicas, as opposed to addressing the *final* replicas directly, is to avoid revealing to the courier which shard the box falls into. In a system with many replicas, this partitioning would otherwise allow a courier to collude with one of Alice's contacts to execute SURB floods.

5.4.2. Reading messages. To read from a box for the first time, a client generates an ephemeral keypair for a new *read envelope*. It selects one of the replicas responsible for that box according to the sharding scheme, and creates a *read request* containing the ephemeral public key, the replica ID, the box ID encrypted with the ephemeral

private key and the replica's public key. It chooses a random courier, and sends it the read request.

Upon receiving a read request, the courier forwards the read envelope to the designated replica. who decrypts the envelope, and checks if it has an entry in its database for the referenced box ID. If an entry does exist, the replica encrypts a reply with the BACAP message encrypted to the envelope public key. It also encrypts a reply encrypted to the envelope public key, but containing a negative acknowledgement instead of the desired BACAP message. The nature of these responses should be indistinguishable to the courier.

If the box was empty but is written to in the near future, the replica schedules new replies to each of the listeners. Each reply is independently delayed, with delays sampled from a uniform distribution to mitigate the courier's ability to infer links between responses that pertain to the same box.

The delay also serves to hide relationships between writers and readers, to avoid a pending read from being fulfilled immediately upon the courier sending a write envelope to a replica. This mitigation is needed to address the cases where writer/reader or reader/reader pairs are using the same courier, but ultimately it does not address all concerns, which we will elaborate on in section 6. The courier caches a listener response associated with a particular envelope for a time.

A client that receives a negative acknowledgement may poll the box again in the near future. To prevent informing replicas of the precise rate of read requests sent by a client, which could link client behavior information with the box ID, the client sends the same read envelope to the same courier, and the courier will refrain from re-sending that envelope to the replica more than once. When the courier sees a duplicate envelope, it uses the new SURB associated with the recent read request to repeat the replica's latest response for that envelope.

Couriers for retried read requests are rotated frequently to mitigate correlation attacks where the courier could otherwise link a number of concurrent readers ceasing to poll at the same time and infer that they were interested in the same box.

5.4.3. Tombstones. BACAP messages come with a fixed-size payload c_i. To allow users to delete messages after sending them, we selectively break the unlinkability guarantees provided by BACAP with *tombstones*, which are BACAP messages with empty c_i. When a replica gets a tombstone for an M_i that it has an existing c_i, s_i for, the tombstone takes precedence and the replica deletes the old c_i, s_i pair. Tombstones are also used in subsection 5.6 for end-to-end reliability.

5.5. Server context and storage duration

BACAP provides for adding *blinding contexts* used when deriving keys, and in the Echomix PKI. [43] provides a Shared Random Value [50] and a set of previous *Weekly Shared Random Values* (WSRV). Clients use the last WSRV as a blinding context so that the box addresses they are querying rotate, and the period for which a Pigeonhole server is able to observe query patterns for a particular box is bounded.

Pigeonhole storage servers collect boxes in per-week buckets, and discard the oldest bucket upon entering a new week. A minimum guaranteed storage time can be calculated from the total storage capacity and the maximum total possible network throughput.

5.6. End-to-end reliable group channels

Sometimes a reader is offline longer than the replicas' data retention period, or the two replicas responsible for a box both fail. We therefore need an end-to-end reliability mechanism. This requires some form of acknowledgement from readers to writers and retransmissions from writers to readers, with a user knowingly disclosing information to contacts in these rare cases. To make sure a malicious writer can't probe whether a reader of their channel is online, acknowledgements are never sent automatically. Instead they are sent opportunistically with the next message the acknowledging user sends on their own channel. Likewise, the retransmission operation is never performed automatically, but only when the user sends a message.

We perform retransmissions in an all-or-nothing fashion by introducing a new courier *copy* command, to prevent a malicious reader from learning about network disruptions which affect a targeted writer while they are retransmitting a series of messages. To write or rewrite multiple messages to a channel, the writer creates a new pigeonhole channel to temporarily store the encrypted write requests for writing to the boxes which it ultimately wants to write to. After completing all of the writes to the temporary channel, it sends a *copy* command to a random courier, which contains the write capability for the temporary channel. The courier derives the read capability for it, reads the encrypted write commands from it, and executes each as it would normal write commands. The new message which the user wanted to send, which triggered the retransmission operation, will be the last write operation in the temporary channel.

The courier does not learn the box IDs of the boxes in the long-term channel it is writing to, because the write operations are encrypted to the replicas chosen by the client. The courier uses the temporary channel's write cap to write tombstones to delete the copied temporary boxes.

These end-to-end reliable channels can be thought of as a single extremely resilient multi-writer channel, which enables group communication applications which are robust against the loss of all data stored by replicas. Communication can even be resumed on a whole new set of Pigeonhole servers without users needing to re-bootstrap their channels.

5.7. Future work and variations on the design

PIR. Both read and write requests provide replicas with a rough, probabilistic ordering of the box IDs, and the guesses get better with each reader. At the cost of efficiency, this could be addressed with a private information retrieval (PIR) system to protect read operations. PIR could streamline our reliable channel mechanism: each channel could have a single *recovery box* in a PIR system where they can write one message per SRV to catch readers up to a point where they can find our messages, instead of writing and reading a large number of tombstones or old messages to reestablish communication with a contact who has been offline for a long time.

Push SURBs. Absent a PIR scheme, there are variations of Pigeonhole where read requests include *two* SURBs: one for immediate acknowledgement of the request, and another to be used later to send the payload after the write subsequently happens. To preserve receiver unobservability with respect to the gateway, this may also necessitate *long echoes*: decoy messages containing two SURBs, one of which is used in the future to provide inbound cover traffic after the user has disconnected.

SURB burning. Using the echo service to intentionally invalidate (unreceived) SURBs previously sent to couriers, to limit exposure to SURB floods.

6. Security properties of Pigeonhole messaging over Echomix

The analysis in this section assumes an Echomix network where each service node is exclusively running a Pigeonhole courier, and an echo service for echo decoys, without a use of unrelated services.

We consider several active network roles, and the effects of collusion between the combinations of two of them, as well as a GPA. Our goal is that collusion between any two of these active roles is insufficient to meaningfully compromise metadata confidentiality. However, an adversary who can compromise some combinations of two or more of these roles is able to perform some useful attacks.

6.1. Single-role adversary capabilities

Global Passive Adversary. Learns which clients are connecting to the mixnet, and learns the traffic patterns between all elements of the network. **In all following adversary descriptions, we assume GPA capabilities implicitly.**

Contact (reader). When they see a message, the reader can infer that the writer was online within the period of time defined by the latency parameters of the network. This can enable an intersection attack, singling out the clients that were connected around the time of each write.

Gateway. Sees the incoming and outgoing packets, but can't distinguish between decoys and pigeonhole messages.

Replica. Learns which box IDs are being written to the replica, and when. Learns which box IDs are being read, and when. Can withhold envelope replies temporarily or indefinitely. Can lie about having messages (either sending bogus responses or denying having stuff they have received). Learns which courier was responsible for a read.

Courier. Learns the rate of resends for a given enveloped write message, and that they come from the same client. Can withhold responses temporarily or indefinitely. Withholding a response results in the client eventually retrying the send. The courier can link these, and could use them

in correlation attacks, but the courier alone does not know which boxes the envelopes are related to. Can drop envelopes or reads, denying a client the services they requested, but cannot target a speciic client.

6.2. Capabilities of colluding pairs of compromised network elements

We assume GPA capabilities. *Mixes* assumes all three intermediate mix nodes on the path are compromised.

Gateway + Mixes. Since this adversary cannot distinguish echo traffic from courier traffic, and service nodes are picked at random, this does not yield useful information.

Gateway + Courier. A courier receiving a copy request can accumulate SURBs as the client retries, and send them in a burst. At the gateway, it can see this burst and link a user to that copy request. This adversary knows the length of the backfill operation, but not the destination box IDs.

Gateway + Replica. Although the courier pinning mitigates it somewhat, this adversary does get to perform a long-term intersection attack by observing which clients are connected when certain boxes are being read.

Gateway + Contact. A gateway and contact can do a long-term intersection attack, as the contact knows in which epoch a user sent a message. This adversary can more efficiently confirm if the user is a client of the gateway by dropping or delaying users messages and observing when messages are received by the contact, when the contact is using the compromised gateway.

Mixes + Courier. Similar to the capabilities of *Gateway, Courier*, except they link the operations to a specific gateway rather than a specific user

Mixes + Replica. No useful attacks.

Mixes + Contact. No useful attacks.

Courier + Replica. Learns which boxes are being written and read. Coupled with observations of a user population and their network disruptions (GPA) can identify the client writing to, and clients reading from, a specific box, when both reader and writer pick the compromised courier+replica combination. Can link box IDs in backfills involving the compromised replica.

Courier + Contact. No useful attacks.

Replica + Contact. Knows when boxes are read/written. For 1:1 conversations, they learn when the other party was online, even though they were just doing a read. For groups, they learn when *someone* in the group was reading, but can't necessarily distinguish readers from each other.

7. Post-quantum security

We have added quantum-resistant cryptographic primitives in a hybrid scheme that combines their guarantees with those of elliptic curve cryptography. These are all implemented in golang cryptographic libraries which are generalised and accessible to other projects. [51]

1) The consensus producing the PKI document, uses the post quantum hybrid signature scheme of Ed25519 [36] combined with Sphincs+[3] [37].
2) We use an upgraded Noise protocol [52], [53], PQNoise, as described in [33]. It uses KEM [4] [54] as opposed to the EC Diffie-Hellman in Noise. Katzenpost [32] is the first implementation of PQNoise.
3) We have two hybrid post-quantum updates to the Sphinx packet format [30]. One that adds cryptographic agility to the classical Sphinx so that it can use any NIKE [5]. We then use the hybrid post quantum NIKE of X25519 [55] combined with CTIDH [56]. The other is a KEM–based nested encryption packet format. It can likewise use any KEM. It is significantly faster, but has larger packet headers.

7.1. Post-quantum mixnet packets

We will now elaborate on the updates to Sphinx. The Echomix/Katzenpost packet encryption has two interchangeable ways to achieve post quantum security.

7.1.1. Post-quantum NIKE Sphinx. Our implementation of NIKE Sphinx uses a generic set of NIKE [57] interfaces that allow any NIKE, adding cryptographic agility to classic Sphinx. We use a hybrid NIKE consisting of CTIDH512 and X25519. This comes at a relatively high computational cost, and so it is appropriate for latency-tolerant implementations with a lower messaging frequency. It preserves the compactness of classic Sphinx by using its *blinding trick* [30].

As in classic Sphinx, the body plaintext contains an integrity tag and is nested encrypted with an SPRP[6] as the payload δ, while the header is composed of three parts:

- α : A NIKE public key,
- β : Symmetrically encrypted routing information section,
- γ : A MAC.

Suppose we have a mix node n, with private key x_n. It transforms the Sphinx packet by replacing α, β, γ with α', β', γ'. The Sphinx blinding trick lets the client compose several NIKE public keys where each key is generated by a node from the last one using the blinding operation. In particular, we generate α':

$$\alpha, x_n \xrightarrow[DH]{} S,$$

$$\alpha, b(S) \xrightarrow[blind]{} \alpha'.$$

A shared secret S is computed using the packet header's public key and the mix node's private key. A

---

3. Not to be confused with the Sphinx packet format.
4. KEM: key encapsulation mechanism
5. NIKE: non-interactive key exchange
6. SPRP: strong pseudo-random permutation, or a wide-block cipher.

KDF is used to generate several other secrets, including a blinding factor $b(S)$. α' is computed by blinding α with $b(S)$. And so we don't need to include separate public keys for different hops, but we are doing additional calculations.

Other operations performed by the node are as follows: it will use S to compute a hash of β, and compare it to γ to verify the integrity of the header. Then it will strip a layer of encryption from the payload, and obtain β', γ':

$$\beta,\ p(S) \xrightarrow[\oplus]{} \beta',\ \gamma',\ n',$$

where n' is the identity of the next node. It will then send off α', β', γ' and the payload to n'.

This is a straightforward implementation of Sphinx with added cryptographic agility. We will now compare it to KEM Sphinx.

7.1.2. Post-quantum KEM Sphinx. We will now introduce KEM Sphinx, our KEM-based [54], Sphinx revision which uses a generic set of KEM interfaces. Similar to our NIKE Sphinx, KEM Sphinx is meant to be used with a hybrid post quantum KEM in order to achieve post quantum security. A good default choice could be Xwing [58]. However our implementation makes available a more general purpose way to compose PQ KEMs using a generic secure KEM combiner that lets one combine an arbitrary number of KEMs while preserving IND-CCA2 security if at least one of the underlying KEMs has IND-CCA2 security. [59]

The KEM ciphertexts are stored in the β section of the header, which makes it significantly larger. They are nested encrypted, and the original stream cipher *xor*ed padding scheme is obeyed. In the routing slot for each hop, the first element is always the KEM ciphertext. The packet still consists of the following parts:

- α : A KEM ciphertext,
- β : Symmetrically encrypted routing information and KEM public keys,
- γ : A MAC,
- δ : The packet's payload.

Figure 4: A circuit diagram of unwrapping a KEM Sphinx message $((\alpha, \beta, \gamma), \delta)$ into $((\alpha', \beta', \gamma'), \delta')$ at mix n.

Since we do not use the blinding trick, the header is less compact than that of classic Sphinx. It is most appropriate when a given usage is able to compensate for the header overhead by making the packet payload bigger.

$$\alpha, x_n \xrightarrow[\text{decap}]{} S,$$
$$\beta,\ p(S) \xrightarrow[\oplus]{} \alpha',\ \beta',\ \gamma',\ n',$$

7.1.3. Speed. Unwrapping KEM Sphinx packets is roughly twice as fast than the classical NIKE Sphinx since it involves one public key operation rather than two. We no longer calculate the group element for the next hop by blinding the current group element. Instead, we extract the new KEM ciphertext from the encrypted routing information section of the Sphinx packet header. The following table compares the header size to Sphinx unwrap speed on a 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz processor for Sphinx variants.

Sphinx type	resistance	ns/op	header size
X25519 NIKE	ECC	151,383	476
X448 NIKE	ECC	254,966	500
X25519 KEM	ECC	57,611	636
X448 KEM	ECC	208,326	780
Xwing KEM	hybrid	175,732	7,164
MLKEM768-X25519 KEM	hybrid	182,334	7,164

TABLE 1: Sphinx variants speed in nanoseconds per operation, and header size in Bytes. It should be pointed out that X25519 KEM Sphinx is nearly three times as fast as the standard X25519 NIKE Sphinx, but the header is only one third larger.

7.1.4. Related work. Another approach to using KEM with nested encryption packets can be found in the recently published EROR packet format [60]. However, this format assumes doubling the payload overhead. The goal of this is to add protection from a tagging attack, in which an adversary who controls both a node on the way and a service can corrupt the payload ciphertext and link the packet with a packet that arrives at a service when it doesn't decrypt. However, since the adversary learns nothing else about the packet this way, and the information gained on other packets from this is negligible, we propose that in the context of a mixnet like Echomix this is not a practical trade-off. Since a practical implementation of KEM nested encryption may benefit from offsetting the large header size with a large payload size, doubling the payload size appears to be additionally costly.

8. Latency and bandwidth overhead

If a packet's journey is comprised of k steps, each incurring a mean delay $\mu = \frac{1}{\lambda}$, then their sum will follow the Erlang distribution,

$$f_{k,\lambda}(x) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!}.$$

Since a single delay has both mean and standard deviation equal to μ, the sum will quickly approach a normal distribution with mean $k\mu$ and standard deviation $\sigma = \sqrt{k}\mu$:

$$\lim_{k \to \infty} \frac{\lambda^k x^{k-1}}{(k-1)!} e^{-\lambda x} = \frac{\lambda}{\sqrt{k} 2\pi} e^{-\frac{\lambda^2}{2k}((x-1/\lambda)^2)}.$$

In the Echomix design the round trip involves 9 steps (gateway - 3 mix nodes - service - 3 mix nodes - gateway), not including the client's initial sending scheduler, which is a separate parameter. The round-trip latency obeys the Erlang distribution for $k = 9$:

$$f_{9,\lambda}(x) = \frac{\lambda^9 x^8 e^{-\lambda x}}{8!},$$

with cumulative distribution function:

$$F_{9,\lambda}(x) = 1 - \sum_{n=0}^{8} \frac{1}{n!} e^{-\lambda x} (\lambda x)^n.$$

This means that the total expected round-trip latency is 9μ, and the probability of exceeding 20μ is about 0.002.

For example, in the Echomix deployment by Zero Knowledge Network [7] the average delay per hop is $\mu = 0.2s$, resulting in average round-trip latency of 1.8s, and a 0.2% chance of exceeding 4s.

The client bandwidth use is a function of the packet size and send frequency, and cryptographic primitives used, and the size of the PKI document which the clients download from gateways. Apart from the PKI document, which is downloaded at most every 20 minutes, the client's overhead is independent of the number of nodes in the mix network. The size of the PKI document is linear in the number of nodes in the network.

In the example of $\emptyset$ Knowledge Network, each packet's user payload size is 30kBs, using an X25519 NIKE Sphinx with an additional 1kB of header and SURB size. With an average of 2.5 packets sent per second, the clients send and receive about 77kBs, which means that a client connected continuously will send and receive about 6.7GBs of data per day, with up to 96% of this memory being usable payloads. These parameters are practical for messaging, medium bitrate audio transmission and interfacing with many internet services such as cryptocurrency blockchains.

Sphinx type	resistance	header + SURB size
X25519 NIKE	ECC	1,082
X448 NIKE	ECC	1,130
CTIDH1024 NIKE	post-quantum	2,030
CTIDH1024-X448 NIKE	hybrid	3,226
X25519 KEM	ECC	1,402
X448 KEM	ECC	1,690
Xwing KEM	hybrid	14,458
MLKEM768-X25519 KEM	hybrid	14,458
MLKEM768-X448 KEM	hybrid	14,746

TABLE 2: Per-packet bandwidth overhead in Bytes on a mix network with a round-trip of 9 hops for different Sphinx variants. A comprehensive list can be found in Appendix IV.

The PKI document grows with the size of the network, since it has to include each node's public key information. This document is typically downloaded once per 20 minute epoch. The directory authorities can also produce a smaller document detailing changes from the previous epoch, so that clients only need to download the full PKI document when they first connect.

Sphinx	dirauths	nodes	replicas	size
X25519	3	10	0	159,901
X25519	9	10	0	459,421
X25519	3	500	5	167,076
X25519	9	500	100	1,071,855
CTIDH1024-X448	3	10	0	159,901
CTIDH1024-X448	9	10	0	459,421
CTIDH1024-X448	3	500	5	167,836
CTIDH1024-X448	9	500	100	1,087,055

TABLE 3: Example size of the PKI document in Bytes. Directory authorities are assumed to use Ed25519 and Sphincs+ hybrid signatures, and replicas are assumed to use Xwing.

9. Conclusion

The Echomix mix network design surpasses the state of the art systems by providing stronger metadata privacy and resistance to global adversaries who compromise users and parts of the network infrastructure, and eliminating multiple vulnerabilities of previously published systems. In particular, it uses symmetry and unobservable traffic coupling to meaningffully protect against traffic analysis, avoiding the mistakes of its predecessors. Echomix is implemented as a robust real-world open source software project, Katzenpost [32], elements of which have been used by several systems, including Zero Knowledge Network, Cloaked Services and our own chat client, Katzen.

In order to extend rigorous security guarantees to the difficult case of persistent multi-user messaging, we introduce the blinding-and-capability (BACAP) cryptographic protocol, which allows users to unlinkably interface with pseudorandom nodes. Pigeonhole storage, in conjunction with BACAP, provides additional powerful privacy properties and extends the messaging protocol to include functionality expected by today's users, such as both short and long term reliability and deleting messages, while maintaining the anonymity-first design philosophy. Together, and implemented on top of Echomix, they are suitable for low-latency, interactive group messaging in the presence of realistic adversaries. It is the first practical messaging system design with such strong threat model.

We also introduce KEM Sphinx and cryptographic agility to the Sphinx packet format, achieving hybrid post-quantum security in both KEM and NIKE Sphinx. KEM Sphinx is faster than NIKE Sphinx, but carries more bandwidth overhead.

Finally, we demonstrate that this design is practical, efficient, with manageable latency and bandwidth overhead and can yield a comfortable user experience for many of today's Internet services.

Acknowledgment

References

[1] Siri Jodha Singh Khalsa. Data and metadata brokering: Theory and practice from the bcube project. *Data Science Journal*, Jan 2017.

[2] Bryce Newell and Joseph Tennis. Me, my metadata, and the nsa: Privacy and government metadata surveillance programs. 03 2014.

[3] Belarus classifies social media channels as 'extremist'. https://www.aljazeera.com/news/2021/10/29/belarus-classifies-social-media-channels-as-extremist.

[4] Nsa can map your movements, determine your fellow travelers with cell data. https://shorturl.at/QnPIg.

[5] Lee Ferran. Ex-nsa chief: 'we kill people based on metadata'. https://abcnews.go.com/blogs/headlines/2014/05/ex-nsa-chief-we-kill-people-based-on-metadata.

[6] Sam Biddle. This undisclosed whatsapp vulnerability lets governments see who you message. https://theintercept.com/2024/05/22/whatsapp-security-vulnerability-meta-israel-palestine/.

[7] https://0kn.io/.

[8] https://www.cloaked.io/.

[9] Masala. https://github.com/katzenpost/katzen.

[10] https://torproject.org.

[11] Alfonso Iacovazzi and Yuval Elovici. Network flow watermarking: A survey. *Commun. Surveys Tuts.*, 19(1):512–530, jan 2017.

[12] Brian Neil Levine, Michael K. Reiter, Chenxi Wang, and Matthew K. Wright. Timing attacks in low-latency mix systems (extended abstract). In *Financial Cryptography*, 2004.

[13] Yossi Gilad and Amir Herzberg. Spying in the dark: Tcp and tor traffic analysis. In *International Symposium on Privacy Enhancing Technologies*, 2012.

[14] Ishan Karunanayake, Nadeem Ahmed, Robert A. Malaney, Rafiqul M. D. Islam, and Sanjay Kumar Jha. De-anonymisation attacks on tor: A survey. *IEEE Communications Surveys & Tutorials*, 23:2324–2350, 2020.

[15] Florentin Rochet and Olivier Pereira. Dropping on the edge: Flexibility and traffic confirmation in onion routing protocols. *Proceedings on Privacy Enhancing Technologies*, 2018:27 – 46, 2018.

[16] Anonymisierungsdienst tor angreifbar: Snowden-effekt verpufft. https://www.ndr.de/fernsehen/sendungen/panorama/aktuell/Anonymisierungsdienst-Tor-angreifbar-Snowden-Effekt-verpufft,tor192.html.

[17] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The loopix anonymity system. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1199–1216, Vancouver, BC, August 2017. USENIX Association.

[18] Claudia Diaz, Harry Halpin, and Aggelos Kiayias. The Nym Network https://nymtech.net/nym-whitepaper.pdf.

[19] U.S. vs Jeremy Hammond complaint. https://www.justice.gov/archive/usao/nys/pressreleases/March12/hackers/hammondjeremycomplaint.pdf.

[20] https://nymvpn.com/.

[21] https://github.com/CloakedServices/CloakedNetworkPoC/tree/main/katzensocks.

[22] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, feb 1981.

[23] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency—choose two. Cryptology ePrint Archive, Paper 2017/954, 2017. https://eprint.iacr.org/2017/954.

[24] https://hoprnet.org/.

[25] Luis von Ahn, Andrew Bortz, and Nicholas J. Hopper. k-anonymous message transmission. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, CCS '03, page 122–130, New York, NY, USA, 2003. Association for Computing Machinery.

[26] Gerrit Bleumer. *DC Network*, pages 313–315. Springer US, Boston, MA, 2011.

[27] David Isaac Wolinsky, Ewa Syta, and Bryan Ford. Hang with your buddies to resist intersection attacks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer &; communications security - CCS '13*, CCS '13. ACM Press, 2013.

[28] Emin Gün Sirer, Sharad Goel, Mark Robson, and Doundefinedan Engin. Eluding carnivores: file sharing with strong anonymity. In *Proceedings of the 11th Workshop on ACM SIGOPS European Workshop*, EW 11, page 19–es, New York, NY, USA, 2004. Association for Computing Machinery.

[29] Raymond Cheng, William Scott, Elisaweta Masserova, Irene Zhang, Vipul Goyal, Thomas Anderson, Arvind Krishnamurthy, and Bryan Parno. Talek: Private group messaging with hidden access patterns. Cryptology ePrint Archive, Paper 2020/066, 2020. https://eprint.iacr.org/2020/066.

[30] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *2009 30th IEEE Symposium on Security and Privacy*, pages 269–282, 2009.

[31] Dogan Kesdogan, Jan Egner, and Roland Büschkes. Stop- and-go-mixes providing probabilistic anonymity in an open system. volume 1525, pages 83–98, 04 1998.

[32] https://github.com/katzenpost.

[33] Yawning Angel, Benjamin Dowling, Andreas Hülsing, Peter Schwabe, and Florian Weber. Post quantum noise. *IACR Cryptology ePrint Archive*, 2022, 2022.

[34] https://spec.torproject.org/dir-spec/index.html.

[35] George Danezis and Paul Syverson. Bridging and fingerprinting: Epistemic attacks on route selection. volume 5134, pages 151–166, 07 2008.

[36] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, pages 124–142, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[37] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. Sphincs: Practical stateless hash-based signatures. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 368–397, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[38] Privacy and accountability in networks via optimized randomized mix-nets, grant agreement id: 653497. https://cordis.europa.eu/project/id/653497.

[39] George Danezis and Len Sassaman. Heartbeat traffic to counter (n-1) attacks: Red-green-black mixes. In *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*, WPES '03, page 89–93, New York, NY, USA, 2003. Association for Computing Machinery.

[40] Claudia Diaz, Steven J. Murdoch, and Carmela Troncoso. Impact of network topology on anonymity and overhead in low-latency anonymity networks. In Mikhail J. Atallah and Nicholas J. Hopper, editors, *Privacy Enhancing Technologies*, pages 184–201, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[41] Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a trickle to a flood: Active attacks on several mix types. volume 2578, 02 2003.

[42] George Danezis and Jolyon Clulow. Compulsion resistant anonymous communications. pages 11–25, 06 2005.

[43] David Stainton, Yawning Angel, and Masala, 2022. https://github.com/katzenpost/katzenpost/blob/main/core/pki/document.go.

[44] Hemi Leibowitz, Ania Piotrowska, George Danezis, and Amir Herzberg. No right to remain silent: Isolating malicious mixes, 09 2018.

[45] Tor rendezvous specification - version 3. Tor Project, November 2013. https://github.com/torproject/torspec/blob/main/rend-spec-v3.txt#L2302-L2307.

[46] Shay Gueron, Adam Langley, and Yehuda Lindell. AES-GCM-SIV: Nonce Misuse-Resistant Authenticated Encryption. RFC 8452, April 2019. https://www.rfc-editor.org/info/rfc8452.

[47] Matt Welsh, David Culler, and Eric Brewer. Seda: an architecture for well-conditioned, scalable internet services. *SIGOPS Oper. Syst. Rev.*, 35(5):230–243, oct 2001.

[48] David Lazar, Yossi Gilad, and Nickolai Zeldovich. Karaoke: distributed private messaging immune to passive traffic analysis. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, page 711–725, USA, 2018. USENIX Association.

[49] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, page 654–663, New York, NY, USA, 1997. Association for Computing Machinery.

[50] David Stainton, Yawning Angel, and Masala, 2022. https://github.com/katzenpost/katzenpost/blob/main/core/pki/sharedrandom.go.

[51] https://github.com/katzenpost/hpqc.

[52] Trevor Perrin. The noise protocol framework. *noiseprotocol, Protocol Revision*, 34, 2018. https://noiseprotocol.org/noise.pdf.

[53] Benjamin Dowling, Paul Rösler, and Jörg Schwenk. Flexible authenticated and confidential channel establishment (facce): Analyzing the noise protocol framework. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *Public-Key Cryptography – PKC 2020*, pages 341–373, Cham, 2020. Springer International Publishing.

[54] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *Advances in Cryptology — EUROCRYPT 2002*, pages 45–64, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[55] Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.

[56] Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. CTIDH: faster constant-time CSIDH. Cryptology ePrint Archive, Paper 2021/633, 2021. https://eprint.iacr.org/2021/633.

[57] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. Non-interactive key exchange. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *Public-Key Cryptography – PKC 2013*, pages 254–271, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[58] Manuel Barbosa, Deirdre Connolly, João Diogo Duarte, Aaron Kaiser, Peter Schwabe, Karoline Varner, and Bas Westerbaan. X-wing: The hybrid KEM you've been looking for. Cryptology ePrint Archive, Paper 2024/039, 2024. https://eprint.iacr.org/2024/039.

[59] Federico Giacon, Felix Heuer, and Bertram Poettering. KEM combiners. Cryptology ePrint Archive, Paper 2018/024, 2018. https://eprint.iacr.org/2018/024.

[60] Michael Klooß, Andy Rupp, Daniel Schadt, Thorsten Strufe, and Christiane Weis. EROR: Efficient repliable onion routing with strong provable privacy. Cryptology ePrint Archive, Paper 2024/020, 2024. https://eprint.iacr.org/2024/020.

Appendix I: Machine-checkable proofs

For the reader's convenience, we provide the machine-checkable proofs in their original electronic format, ready for verification. They can be found at https://github.com/katzenpost/research/. along with the full lists of Sphinx geometry overheads and PKI document sizes.

Appendix II: Structure of courier read/write requests

A Sphinx [30] payload destined for a courier contains (for *read* and *write* requests):

Shared fields:
1) The sender's ephemeral hybrid public key:
   a) x25519 public key
   b) CTIDH-1024 public key
2) for each designated replica:
   a) 256-bit DEK encrypted to the replica's public key
3) enveloped message which the courier can't decrypt

For **write requests, couriers see:**
1) (shared fields)
2) SURB for courier to ACK receipt of request once at least one replica has accepted it

For **read requests, couriers see:**
1) (shared fields)
2) Immediate-use SURB, for courier to ACK receipt of the encrypted (to the client) reply from the designated replica
3) replica/shard id designating each replica to contact

Appendix III: Structure of replica envelopes

Replicas see **write requests** as:
1) sender's ephemeral public key
2) envelope DEK encrypted with shared secret between sender private key and replica public key
3) enveloped message, encrypted with DEK, containing a BACAP message:
   a) BACAP box ID (M_i^{ctx})
   b) BACAP payload (c_i^{ctx})
   c) BACAP signature (s_i^{ctx})

Replicas see **read requests** as:
1) sender's ephemeral public key
2) envelope DEK encrypted with shared secret between sender private key and replica public key
3) enveloped message, encrypted with DEK, containing a BACAP box ID:
   a) BACAP box ID (M_i^{ctx})