Mixnet Research Review

E. J. Infeld and D. Stainton

April 15, 2024

Abstract

We provide a selected overview of decades of research on Mixnets, with special focus on the modern solutions implemented in multiple actively developed projects, such as Katzenpost, HOPR, 0KN or Nym. The purpose of this work is to provide a resource on the general knowledge in this robust field. Some parts were withheld from this version pending publication. This work is supported by a grant from the Wau Holland Foundation.

Contents

Why Mixnets? The need for systems that can protect from powerful adversaries.

The strife for privacy in the modern world is inseparable from our need for freedom and sovereignty. It is no longer controversial to say that we face very powerful adversaries in this strife. These could be state, corporate or criminal actors, vying for our information to use as means of making profit, manipulating us and others, gaining leverage, strengthening their authority, or persecution. In many contexts, we have little hope for non-technical solutions due to lack of sufficiently powerful pressure in favor of privacy.

In our quest for technical solutions, we need equally powerful tools. In the case of communication tools, the Internet's bread and butter, we would like to allow users to interact and exchange information with reasonable expectation of both the content and metadata of their communication, and personal information such as a user's social graph, being protected from such adversaries.

If we hope for our work to be relevant in the modern world, we can no longer settle for weak threat models. Therefore, consider an adversary capable of the following:

- The adversary can see the traffic of the entire global internet, and in particular between devices in the network, and is capable of intricate statistical analysis of gathered data.
- The adversary can disable parts of the network.
- The adversary can plant or take over some devices in the network to inject malicious code and manipulate the functioning of the network or to gain access to the information available to them. The takeover could happen by technical means or by exercising force outside of the network.
- The adversary has very large, but not infinite, 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 a quantum computer in the near future.
- The adversary can supplement collected data with rich context of already gathered data on all users from other sources.

We aim to rise up to the challenge with modern Mixnets.

Mixnet basics

A mix network is a collection of network devices, referred to as *mix nodes*, that attempt to relay multiple messages from a set of senders to a set of receivers (these are typically the same set) in such a way that a third party observing the network is unaware of which sender is sending a message to which receiver. We will refer to these senders and receivers as either *users* or *clients* of the network.

A *batch mix* could for example have regularly scheduled rounds in which messages are collected and routed through several devices (*hops*) in the network where each hop shuffles the messages. Therefore, it can be said that in a batch mix, the primary source of *mix entropy* comes from the message shuffling, and indeed the entropy or the measure of the uncertainty a passive network observer would have in trying to link input messages with output messages, is a function of the batch size. On the other hand, *continuous time mixing* strategies differ from batch mixnets because each message is delayed independently from the other messages which allows for a more favorable latency tradeoff as is the case with the Sto-and-Go design [M1] used in Loopix [M2]. In such a strategy, the sampled delay must be probabilistic in nature, since a deterministic delay could be reconstructed by an adversary and used to correlate messages.[1]

We would like to impress on the reader, that when designing networking protocols with strong adversaries in mind, every decision comes with trade-offs and consequences. Let us focus for now on a simple batch mix with a single mix node, that is relaying messages with the hopes of hiding from a network observer which of the incoming messages corresponds to which of the outgoing ones. Suppose we consider a setup in which Alice only has the mix node's public key, and encrypts her message to Bob with that, and the mix node decrypts it and then encrypts it again with Bob's public key. A straightforward improvement on that comes with nested encryption - if Alice wishes to send a message to Bob, she can encrypt the message with Bob's public key, add a note for the server where to send it, and then encrypt the encrypted message together with that note with the server's public key. The server can then pass the message onto Bob without knowing its content. The difference between the two simple setups is illustrated below - as Alice sends a message to Bob, and Charlie sends a message to Diane, and the mix server releases them both at once. A casual observer might not know whether that's what's happened, or if Alice sent a message to Diane, and Charlie to Bob.

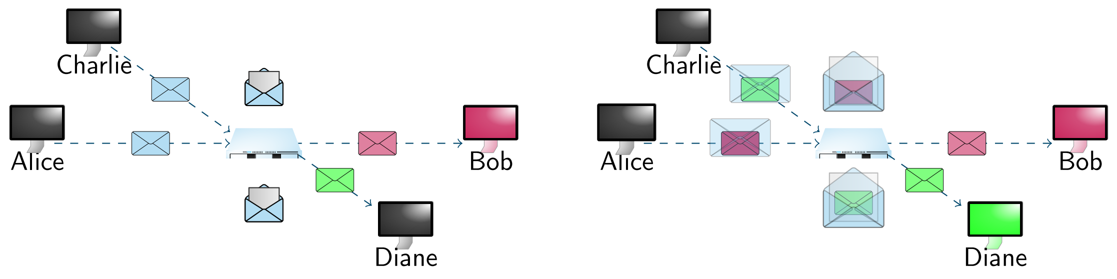


Figure 1: Blue envelopes are encrypted with the server's public key, red with Bob's and Green with Diane's.

This has the advantage of the server not learning the content of the messages, but it also has certain trade-offs. For example, each sender has to retrieve the final receiver's public key, possibly from the server. Also, a network observer could take a message going to Bob and try to recreate what it would have looked like coming into the server by using the server's public key, and then match it to a message sent by Alice. Alice can prevent this by adding a little randomness before encrypting the outer layer of the message, which the server will then strip. Similarly, for every improvement or addition we make to the system we will keep careful track of its consequences.

[1]Randomness is in the eye of the beholder. It's enough if the delay is sampled in a way that appears random to the adversary.

Compromising on latency and compromising on bandwidth

Let's now compare two simple *mixing strategies* that one may encounter with mix nodes. A mixing strategy is the method which a mix node employs to attempt to hide from an adversary which of its users are communicating with each other.

One, a mix node might collect many messages from multiple users, and then release them all at once, a batch mix. The hope is that an adversary will not be able to match a message being sent to a receiver to a message coming in from a sender. The batch mix scheduling and batch mix size (messages per batch) are tuned so that the desired latency/bandwidth tradefoff is achieved where the main cost is bandwidth since batch mixes always send an entire batch of messages. Since the adversary sees messages being sent, but doesn't know which outgoing messages correspond to which incoming ones, this strategy achieves sender-receiver *unlinkability.*

Two, a mix node might send decoy traffic between itself and the users with the overall traffic pattern independent of the amount of messages the users are sending, and the messages forming an indistinguishable subset of the traffic. In this method the main trade-off is the amount of bandwidth it uses. The network adversary does not observe when a message is being sent, and therefore we say this strategy achieves *unobservability.* We sometimes see protocols that have asymetric properties, where, for example, messages being sent are unobservable but messages being received are not, and vice versa. We then talk about *sender unobservability* or *receiver unobservability*, respectively.

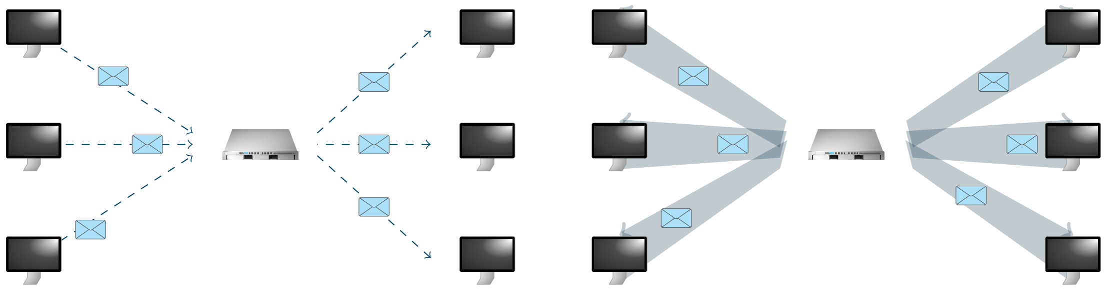


Figure 2: The system on the left compromises on latency. The system on the right compromises on bandwidth.

For a more sophisticated system, we would want to employ multiple independent mix nodes, and send a message through a sequence of them to prevent any one actor from correlating senders and receivers. This is where we go from a single mix node to a system deserving of the name *mix network*. We would want to protect the messages with layers of encryption for each node, in much the same way it is done in Tor. However, Tor attempts to simulate the standard internet browsing experience, and therefore minimize latency and optimize bandwidth. It cannot afford to have its relays delay connections to properly mix them. It also cannot effectively employ decoy traffic or padding, since it wants its users to be able to take full advantage of their bandwidth at any given time, and padding to that level would be prohibitively expensive. This makes it vulnerable to adversaries capable of watching the network and correlating users' behavior. This is a *statistical disclosure attack*, and we will elaborate on it in the next section. This vulnerability in Tor cannot be fixed if it is to keep its desired functionality.

By contrast, a Mixnet does make these compromises in order to thwart powerful adversaries. Early designs attempted to ensure anonymity by compromising on latency rather than badwidth, while most modern Mixnet designs usually attempt find a sweet spot with the use of both. This means that some amount of *decoy traffic* is being sent by a user, ideally independently of their actual traffic needs. The amount of traffic a mixnet allows for is also an upper limit to the user's bandwidth, since setting it too high would quickly add up to a significant drain on their resources.

This recalls the Anonymity Trilemma - a rule that to maintain anonymity guarantees, you have to compromise either on the bandwidth overhead or on latency. It should be more accurately said that you need to in fact compromise on both at least a little bit - at the very least, the network packets need to be padded to a constant size, and there needs to be some delay in the mix node or no mixing is achieved. It was systematized in [E1]. Some modern systems allow for tuning of both the latency and the bandwith overhead, and are therefore somewhat versatile. This includes Loopix [M2], where one could compensate for shortening the latency with increasing the bandwidth overhead and vice versa. In practice, user experience concerns are likely to constrain these choices.

Because of these compromises, some communication activities lend themselves to Mixnets more readily than others. Non-synchronous messaging, as with email, is a natural use case for this kind of an anonymity system. On the other hand, browsing the internet comfortably, as with Tor, appears to be impossible.

Mixnet evolution

Topology

Over several decades, there have been significant improvements in theoretical Mixnet design. One of the first questions one might ask is, how do we structure our network from the point of view of traffic flow. We would like to structure our network, so that:

- Messages going through the network are *mixed* with as many other messages as possible, or in the case of a network with decoy traffic, as much traffic in the network as possible.
- We would like to limit the number of active links in the network, since they diffuse the traffic and if we have to populate them with decoy traffic, increase cost.
- We would like to not have few points of failure - and adversary should not be able to compromise the network by taking over a small number of devices.

For a high probability of mixing, a straightforward design is one that has a bottleneck - a node all messages have to go through. However, that is then a point of failure, should an adversary compromise that node, they can disable the network. The opposite strategy, allowing messages to take any route between all nodes, ensures a lot of resilience in the network, but requires that all $\binom{n}{2}$ links may have to be populated with decoy traffic, where n is the number of nodes, and in both directions, which makes $n(n-1)$ populated[2] links. This strategy is typically referred to as *free routes.* Additionally, if the volume of traffic in outgoing links is generated depending on the volume of incoming traffic, as is modelled in [M3], a network structure that allows for graph cycles will see them locked in a non-decreasing pattern. However, that model of generating decoy traffic is not directly relevant for the Mixnets being built today.

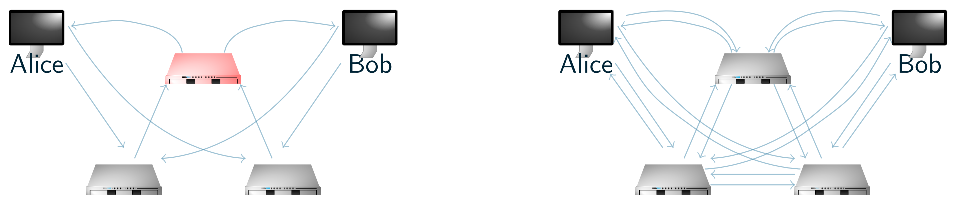


Figure 3: A Mixnet with a bottleneck (in red, left), and a Mixnet with free routes (right.)

[2]Much of the literature talks about *padding* the traffic to a regular pattern, in which case we would say these are *padded* links. To ensure correctness we will typically use the term *decoy traffic* rather than *padding*, since in most modern designs the generated traffic pattern is not regular. In some contexts, the term *padding* remains appropriate.

It turns out [M3], that a stratified topology, that is, one where mix nodes are divided into an ordered set of disjoint subsets that are treated as *layers* of the network, with each message picking one mix node in each layer, has certain advantages. So suppose for simplicity, that we have $n = m \times l$ mix nodes, and divide them into l sets $L_0, L_1, \ldots, L_{l-1}$ of m nodes each.

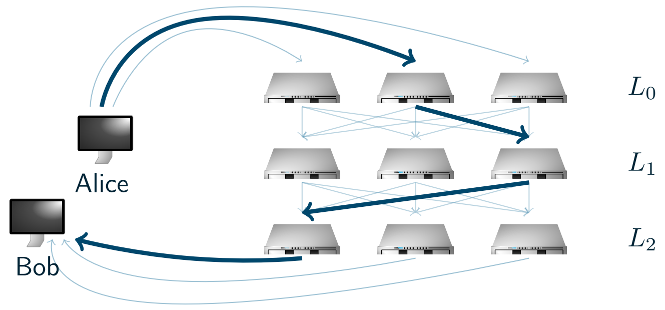


Figure 4: Alice sends a message to Bob through a Mixnet with a stratified topology with $m = l = 3$.

Given the above optimizing goals, it is an efficient setup for most traffic parameters. Only $(l-1) \times m^2$ links need to be populated inside the network, with users supporting m links to and m from the network. In a fully connected model, this would instead be $2 \times \binom{n}{2} = n(n-1)$, plus n links to and n links from the network for each user. While the network is consistently populated, this graph guarantees that any two connections are mixed, since any node in a higher layer is connected to any node in a lower layer.

In a general network setup, and assuming constant traffic, whether two connections are mixed is an interesting question dealing with graph shadows[3], due to the transitive nature of the *mixing* relation. For a stratified, padded mix network like the one pictured, the lower shadow of any entry node includes all the nodes below it, and the upper shadow of any exit node includes all the nodes above it. The question of mixing in general graphs may be more of a mathematical curiosity than a practical concern, but some of this line of analysis is touched on in [M4].

In a network with variable decoy traffic, where we cannot assume links are populated at all times, we would talk about specific packets being mixed, rather than connections, and the connectedness relation has to be treated as probabilistic. One such design is Loopix [M2]. We hope, however, that for any populated link the decoy traffic is consistent enough that we can approximate the probability to be 1, as is done implicitly in all existing analysis.

The challenges in constructing the cryptography and a packet format for Mixnets

Mixnets are particularly vulnerable to certain attacks. For example, in a system that employs nested encryption, an attacker might take a message outgoing from a node, and recreate what it might have looked like coming into the node by encrypting it with the node's public key. This would not be a concern in a system that does not attempt to use relays to hide metadata, but in a Mixnet it allows the attacker to match a message to its previous hop. One way to address this problem is to add some randomness before encryption, to make it harder to match the actual pre-image and thus find its origin. This simple solution turns out to not be enough in the case of straightforward RSA implementations, where the attacker can exploit the multiplicative homomorphism to take advantage of the information gained from sending a message's product with a well chosen factor through the node [A1].

[3]In a directed graph with no cycles, a lower shadow of a node consists of all nodes that one can get to from that node. Similarly upper shadow are all nodes that have this node in their lower shadow.

When wondering how to encrypt messages travelling through a mix network, we have the following general concerns. We would like the node to find an encrypted payload and a header with instructions on what it should do with the packet next. The properties we're looking for in a packet format are:

- The packet needs to get to the destination, after a series of hops, and each relay on the way should be able to find out the next hop only.
- A network observer shouldn't be able to link an outgoing packet with an incoming packet.
- The packets should be padded to a standard size.
- The packets should, ideally, stay the same size throughout their entire route. Even in a stratified topology this is may become important if, say, heartbeat [M5] packets originate from the network interior and yet must be indistinguishable from other packets originating from the network perimeter.
- We would like integrity protections, for example a MAC[4]. Ideally, it should be checked at every stage on the way.
- We would like this process to be efficient in both space and processing requirements.
- We would like to be reasonably confident about the security of the format, for example by a proof of security.
- Allowing for replies while the sender does not need to disclose their identity or location is a plus. This can be achieved by sending cryptographic reply instructions. Often such reply may not be able to have a MAC, since the payload of the reply is not known in advance and therefore we can't pre-generate it. We could, however, try to rely on homomorphic encryption schemes to generate a MAC.

Sphinx

The Sphinx [M6] packet format was a breakthrough in Mixnet functionality, since it at least partially fulfilled all of the above requirements. The name comes from the fact that the body is encrypted with the Lioness cipher [M7]. A more modern parameterization of Sphinx can simply replace Lioness with a modern SPRP[5]. In this way, the Sphinx packet format's MAC only covers elements within the header. Sphinx uses SURBs (Single Use Reply Blocks) so that Bob can send a reply to Alice without knowing her location. Alice sends a message to Bob containing a SURB, Bob uses the SURB to compose a Sphinx packet which he sends to Alice without knowing the route the message takes on the way back to Alice.

SURBs were first introduced by the Minx [M8] nested encrypted packet format which was used by Mixminion. However, Minx had many design flaws [A2] and Sphinx was created to replace it.[6] All nest encrypted packet formats prior to Sphinx had various design, security and privacy flaws. None of them had security proofs and Sphinx is the first packet format provide one.[7]

The Sphinx packet format is compact, however this compactness comes at a cost of computational efficiency. Prior packet formats such as Mixmaster stored one RSA public key inside the header per hop, creating very large packet headers. This is essentially how it's done with a PKE[8] or KEM.[9] However, the original Sphinx was accomplished using a NIKE[10] and therefore was able to use the *blinding trick*.

[4]MAC: Message Authentication Code

[5]SPRP: Strong Pseudo Random Permutation

[6]It is our understanding that the Python reference implementation of Sphinx was meant as a drop in replacement for Minx in Mixminion but it was never used because the Mixminion project was abandoned by its authors in favor of their new anonymous communication network, Tor.

[7]It is not a proof in a rigorous mathematical sense, as these are typically lacking in Mixnet literature.

[8]PKE: Public Key Exchange

[9]KEM: Key Encapsulation Method

[10]NIKE: Non-Interactive Key Exchange

It's easiest to introduce the Sphinx blinding trick by discussing how Sphinx packets are decrypted as they travel through the network. A Sphinx packet is composed of a header and a body. The body plaintext contains an integrity tag and is nested encrypted with an SPRP while the header is more complicated and composed of three parts:

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

Suppose we have a mix node n, with private key x_n. It must cryptographically transform the Sphinx packet where the goal is to replace $\alpha,\ \beta,\ \gamma$ with $\alpha',\ \beta',\ \gamma'$. The Sphinx blinding trick lets the client compose a Sphinx packet with several NIKE public keys where each key is generated from the last one using the blinding operation. In particular, α' is generated like so:

$$\alpha, x_n \underset{DH}{\longrightarrow} S$$

$$\alpha, b(S) \underset{blind}{\longrightarrow} \alpha'$$

A Diffie–Hellman shared secret S is computed using the packet header's NIKE public key and the mix node's NIKE private key. The shared secret is used with a KDF[11] to generate several other secrets, including a blinding factor $b(S)$. And finally, we compute alpha' by blinding alpha with the KDF generated blinding factor. 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: 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 $\beta',\ \gamma'$:

$$\beta,\ p(S) \underset{\oplus}{\longrightarrow} \beta',\ \gamma',\ n'$$

where n' is the identity of the next node. It will then send off $\alpha',\ \beta',\ \gamma'$ and the payload to n'.

Mix nodes do not check the integrity of the payload unless that mix node is the final destination of the Sphinx packet. Only the final destination of the Sphinx packet is allowed to decrypt the payload plaintext which contains an integrity tag. The SPRP which encrypts the payload, also known as a wide block cipher, would have destroyed the integrity tag had the payload ciphertext been mutated during transport, which is considered enough of an intergrity guarantee.

We will introduce our KEM-based, post-quantum Sphinx revision in an upcoming paper. It does not use the blinding trick, and so has a bigger packet header. The KEM ciphertexts are stored in the beta section of the header, 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. We have also constructed a CTIDH-based post-quantum update to Sphinx. Both of these solutions have their trade-offs. In particular, where the KEM one has large headers, the CTIDH one has a large computational overhead.

A protocol design limitation to keep in mind when composing with the Sphinx packet format is that the forward and reply routes are selected by the sender of the Sphinx packet. Therefore special care must be taken to compose a messaging protocol where participants have strong location hiding properties. In particular we'd like to compose protocols where the sender does not know the final destination of the message because the overall route is composed by multiple client entities, namely the sender and recipient.

[11]KDF: Key Derivation Function

Dropping the reply requirement

If we do not have the requirement that the sender should be able to allow for replies without disclosing their identity or location, things get easier. There would be no need to complicate the packet with the use of the SPRP, since it is only there to accommodate the SURB construction. In a setting where SURBs are not at all needed, the payload could be encrypted with an AEAD[12], where the KDF is used to deterministically generate the key and nonce. This has been touched on in [A3]. We propose to call this forward-only version of Sphinx "Shark."

In the context of a threat model which includes adversaries willing and capable of performing compulsion attacks, SURBs are a risk, as their recipients can be discovered. The compulsion attack on forward Sphinx packets and on SURBs is only partially mitigated by periodic mix key rotation, and even that assumes the adversary will never have access to a cryptographically relevant quantum computer. We elaborate on this in the attacks' mitigation section.

Forward Mix Security and Other Design Elements

Forward security assumes that the mix node destroys the private key material immediately after mixing. Both the client and the mix must keep track of their own states and when to use them with which node. If the client doesn't have an established ratchet state with each mix node in the network, then the possible routes will necessarily be restricted. This is discussed in [M9]. It is an interesting paper that discussed some powerful ideas for composing mixnet protocols such as *interdependent SURBs*, *path burning messaging*, and *plausible deniable routing*.

The topology of a Mixnet that employs SURBs can be designed with *crossover points* - nodes on the route that keep Alice's SURB and generate a new one for the rest of the way. We then talk about the messages route as being composed of two *fundamental routes*, before and after the crossover point. There could of course exist a chain of arbitrarily many fundamental routes.

If we were to employ a Post Quantum Hybrid NIKE Sphinx using CTIDH, it would be very slow and involve a huge amount of computational overhead. This can be at least partially mitigated by having an optional section of the mix network that only routes via hybrid CTIDH NIKE Sphinx. Crossover points would be used to send packets into and out of that network subsection.

Loopix

Loopix [M2] is a layered Mixnet design introduced in 2017. It uses the Sphinx packet format, although without the full sender anonymity provided by SURBs. The three characteristics that place it in the larger context are as follows:

1. Using an exponential probability distribution to sample independent delays of each message at each node, inspired by [M1].
2. Some of its decoy traffic are packets travelling in loops - their final destination is the same as origin, which can be either an end-user or a mix node, inspired by [M5].
3. Service providers act as intermediaries between end-users and the mix network, providing offline storage and easing the resource burden on the users.

[12]AEAD: Authenticated Encryption with Associated Data

Delay Sampling

A key characteristic is the cute idea introduced in [M1] to use an exponential probability distribution to sample the delays imposed on each message at each node. These delays are sampled in advance by the sender. This distribution has the advantage of being *memoryless* - at each point in time, a message that is sitting in a mix will have the same probability distribution of the remaining delay, regardless of how long it has already been waiting. This means, in particular, that for an external observer, the probability distribution of which message will be sent next is uniform at all times. For a constant parameter $\lambda > 0$, such that the target mean is $1/\lambda$. the delays approximate the probability distribution function

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

If a message has already been sitting in a node for an amount of time s, the probability that it will leave the node in the next time period t is independent of s. This can be written as:

$$\forall t, s \geq 0, \quad \mathbb{P}[x \leq t] = \mathbb{P}[s \leq x \leq s + t \mid s \leq x].$$

This property is easy to verify. The right hand side of this equation is:

$$\frac{\int_s^{s+t} f(x,\lambda)dx}{\int_s^{\infty} f(x,\lambda)dx} = \lambda \frac{\frac{-1}{\lambda}e^{-\lambda(s+t)} - \frac{-1}{\lambda}e^{-\lambda(s)}}{\frac{-1}{\lambda}e^{-\lambda\infty} - \frac{-1}{\lambda}e^{-\lambda s}} = \lambda \frac{e^{-\lambda s}(1 - e^{\lambda t})}{e^{\lambda s}} = \lambda(1 - e^{\lambda t}),$$

and so we see that the result is independent of s, and equal to $\mathbb{P}[x \leq t]$.

The advantage of this is that at any point, any one of the messages sitting in the node can be the next to be released with uniform distribution, and the expected time to a particular message being released is the same. The expected time to the next message being released will depend on the number of messages currently in the node. It is however worth pointing out that the first, far more relevant, of these properties can also be achieved more simply by setting up the node to release messages according to any time distribution, and then choosing which of the messages it's storing should be released uniformly at random. If the time distribution in question is sampling exponential delays, the expected time to the next message being released will be constant, but the expected time to a particular message being released will vary.

The authors call this a *Poisson mix*, since they assume the overall time distribution of a node sending messages looks like a Poisson process, a mistake carried over from [M1]. This is only true if we can argue the expected number of messages released in any time period is constant, which is not the case here, since the number of messages in a node fluctuates with the traffic in the network. Both papers appeal to the fact that a sum of Poisson processes is a Poisson process, which would be true if you were adding specific, well defined Poisson processes, but it's not if you keep changing which and how many processes you're summing up. In particular, since the behavior of the mix depends on the amount of traffic, it is not memoryless. More packets released in one second is correlated with more packets released in the next. Unfortunately, [M2] bases many of its conclusions on this claim, and so much of the paper has to be discarded as false.

Even if the mix was correctly defined as Poisson, this name wouldn't correspond well to the valuable properties of this model, and in fact a Poisson behavior can be achieved in multiple ways that would not be desirable, in particular without messages being treated independently, which is the strength of this model. The simplest such construction would be to put messages in a FIFO queue, and release them in order with exponential delays between them. This will in fact be a Poisson process, while not mixing the messages at all.

Another disadvantage of calling it a *Poisson mix* is that a developer implementing this design might be misled into sampling delays from a Poisson distribution, rather than exponential distribution. In fact the Poisson behavior (or not) of the node is not something that's relevant to the implementation at all. We will therefore insist on calling it the *Memoryless mixing* instead. Importantly, we should not call the mix itself Memoryless, but the model still benefits from the individual message delays being sampled from a memoryless distribution. We provide an intuitive introduction to these distributions in the Extended Commentary section at the end of this work.

Decoy Traffic

Another key characteristic of the Loopix design is the way it generates traffic in the network. Each end-user generates three regular types of traffic $\Lambda_P,\ \Lambda_L,$ and Λ_D, each distributed in time as a Poisson process. This is achieved by sampling the delay between two messages in each $\Lambda_P,\ \Lambda_L,$ and Λ_D from the exponential distribution. Λ_P is there to accommodate real messages sent by the user, and whenever no messages are sent it generates dummy messages to randomly selected end-providers, thus padding the pattern to a steady process. The end providers drop these dummy messages upon decryption, and so they are called *drop messages.* Additionally, the stream Λ_D consists entirely of drop messages at all times.

Finally, Λ_L is the loop traffic, inspired by [M5]. These packets have the same origin and destination, and provide a way for the user to check that the network is operating correctly, and that he is not being targeted with an $n-1$ attack, or any other attack that involves a disruption of the network. This loop traffic is also the only source of cover for messages being received by the user, however it is not effective for that purpose. As this stream has a known time-distribution, and messages being received are being sent independently of it, then as long as the amount of messages is statistically significant compared to Λ_L, the system is still vulnerable to statistical disclosure attacks due to receiver observability. Mix nodes and providers also generate loop traffic inside the network in order to take stock of the functioning of the network and detect $n-1$ attacks (see page 12).

The Role of Providers

In the Loopix design, each user chooses a service provider that acts as a stable intermediary between the user and the Mixnet. This has many advantages - the provider not only ensures offline storage, so a user can receive messages without being constantly connected, but also takes on much of the filtering of incoming traffic and resource burden imposed by the cryptographic protocols used. In particular, packets are decrypted and processed at provider level, and most of the cover traffic is dropped. The model does not provide receiver unobservability, since the amount of legitimate traffic going to a receiver is independent of the decoy traffic, which at that hop is only the user-generated loops which may be removed as noise, unless the ratio of the remaining traffic fits in a statistical error of the user-generated loops. The provider is also a significant point of failure, and any Loopix-based system should pay special attention to the risk to the user should their provider be compromised.

[next paragraph is withheld pending publication, as is much of the EC section]

A summary of security concerns in a general Mixnet

There exists a rich body of work analyzing how one might disrupt the functioning of a Mixnet or circumvent its anonymity protections. We have endeavored to categorize these attacks in the following table. It does not include detailed attacks that arise from some specific networking choices. For the case of Katzenpost, these are detailed in the threat model document where the corresponding table is longer.

Mixnet attack type	Attack description	Necessary adversary capabilities
Intersection, Statistical Disclosure Attacks [A4] [A5]	Over time, adversary can glean statistical information that makes the probability distribution of who Alice is communicating with non-uniform. Law of Large Numbers implies the anonymity set tends to the set of clients with identical probability in the long run to the actual recipient.	The adversary is able to see messages entering and leaving the network. This is customarily treated as a PGA, despite only requiring a view of the network's perimeter. Must be able to distinguish messages from dummy traffic with better than uniform probability, or observe when users are active.
$n-1$ Attack [A6]	The adversary causes the mix to contain only messages sent by the adversary, except one. This often means that the adversary drops or delays other messages until the mix is empty before the target message enters the mix. The adversary sees the target message exit the mix to its next destination.	The adversary must compromise routers which are upstream from a target mix node, as well as be able to tell when a target message passes through them.
Epistemic Attack [A7]	The fact that a client is issued only a subset of the mix nodes' directory and encryption keys can leak information to the adversary.	The adversary has knowledge of the target client's view of the network which distinguishes them among clients. This could happen via a zero day or a design flaw such as not implementing PIR for discovery.
Denial of Service Attack	The adversary is able to disrupt the functioning of the service, often by overwhelming its resources.	The adversary has sufficient network and computational resources to overwhelm the network.
Sybil Attack	The adversary plants a large number of malicious nodes, and is therefore able to glean partial or complete information to follow a message through the mix and disrupt the network.	The adversary has sufficient resources to take over the network, and the network's design allows for the creation of a large number of malicious nodes.
Compulsion Attack [A8] [A9]	The adversary compels enough honest node operators to disclose information to follow a message through the mix or disrupt the functioning of the network.	The adversary has the necessary force to compel a sufficient number of honest actors to do the adversary's bidding.
Timing Attack [A10] [A11]	An active adversary manipulates the timing of the packets passing through compromised routers, or passive adversary exploits timing information that is leaked despite padding.	The passive attack could happen via a zero day or design flaw. The efficacy of the active attack needs to be analyzed with respect to the specific design.
Cryptographic Attacks	The adversary is able to forge a signature, generate a second hash preimage, decrypt cyphertext or do other damage assumed to be prevented by the use of cryptography.	The adversary can break the security of one or more cryptographic primitives through a cryptographic zero day or sufficient computational resources, or exploit a flaw in their implementation.
Tagging and Replay Attacks	The adversary gleans information or distrupts service by manipulating the packets.	The adversary is able to compromise integrity protections through a zero day or a design flaw.
Endpoint Security	The adversary breaches the security of a user's device via an attack not directly related to the mixnet.	The adversary is able to exploit a technical flaw in the user's device or compel the user to grant him access.

Common mitigation strategies

Intersection, Correlation and Statistical Disclosure Attacks

This attack typically assumes a global passive adversary who watches Alice's interactions with the mix network, but it's worth pointing out that a view of the perimeter of the network is enough. Whenever Alice sends a message, a set of potential recipients are noted by observing which clients receive a message shortly after Alice sends her message. After a period of time of noting these sets of potential recipients, an intersection among these sets may reveal the set of recipients Alice sends messages to.

Suppose, in a simple scenario, that Alice sends a message to Bob and an observer sees an anonymity set of k possible recipients, including Bob, where the other $k-1$ recipients are chosen uniformly at random from a set of n users. The next time Alice sends a message to Bob, with another anonymity set of $k-1$ other users again chosen uniformly at random, the probability that another user who was in the first set will also be in this one is only $\frac{k-1}{n-1}$. This illustrates that the over time, Bob's anonymity set shrinks dramatically, and soon will shrink to just Bob with probability approaching 1. In practice, the other users are not chosen uniformly at random and the set sizes vary, but as long as there is no stable anonymity set of users who are always present, the adversary will still, in time, identify Bob.

The example we described is called an *intersection attack*, but this can be generalized to situations where a user being in an anonymity set is not binary, but some users are more likely to be the recipients than others. These recipients will still be identified in time due to the Law of Large Numbers, a fundamental theorem of Probability, as long as over time they are the most likely recipients. This more general attack is referred to as a *statistical disclosure attack.*

The only way to prevent statistical disclosure attacks is to provide reliable sender and receiver unobservability. A typical strategy is to use decoy traffic to create traffic patterns independent of whether a user is in fact either sending or receiving messages. This is extremely difficult to achieve in practice. It would require a user's behavior patterns, such as when they use the system, to be independent of whether they have messages to send or receive. In a world with extremely powerful corporate and state surveillance actors one cannot expect that, especially with casual users sending personal communication. One can, however, attempt to make these attacks take more time by reducing the amount of available information, for example by restricting the system to asynchronous messaging.

If at least one of the communicating parties is a service or user particularly concerned with anonymity, maintaining a stable traffic pattern and perceived behavior independent of whether a message is received, the hope of mitigating statistical disclosure attacks remains. For example, putting a SecureDrop[13] node on a Mixnet can be executed in such a way, as long as it is implemented with reliable padding - even if users sending messages into it have observable behavior patterns, provided the operator of that node does not. In particular, having several clients connected to the network in a continuous manner would make them a stable anonymity set, and even if a user sends multiple messages to one of these services, a statistical disclosure attack would not reveal which one, provided there are no other vulnerabilities.

It is worth pointing out that Loopix[M2] does not provide receiver unobservability and is therefore vulnerable to statistical disclosure attacks even for most careful users. [parts withheld] For the provider-client traffic, practical implementations of this design use padding to make sure that a provider's response to a client's query looks the same whether it contains a message or not.

[13] https://securedrop.org/

$n - 1$ Attack

An $n - 1$ attack consists of an adversary gaining control of nodes upstream from a target mix node, and using that advantage to follow a target message through that target node. The strategy is to manipulate incoming messages so that the messages which dwell in the target mix are only messages sent by the adversary, except one message, the target message. Depending on the network, he could do it by delaying or stopping all other messages, or flooding the node with messages that are recognizable to him. Against a continuous time mix as in Loopix, the n-1 attack requires the adversary to sufficiently delay or drop all non-target messages destined for the target mix.

Nodes and users taking stock of the functioning of the network by sending loop messages that are meant to come back to them is considered a detection tactic for $n - 1$ attacks, since if an attacker is delaying, stopping or altering messages, that would cause these decoy message to not be received at the expected time or to be droppe completely. This was introduced in [M5] and is a strategy that gives Loopix [M2] its name.

Epistemic Attacks

Many networks will issue the clients with only a subset of the node directory and keys, for usability reasons. This will result in the client then not taking advantage of the whole network and restricting messages to the subset they know, and if an adversary has knowledge of that, they can exploit it. For small enough networks, the straightforward answer to this is to issue the clients with the entire node directory and let them take full advantage of the system. Once the network grows, the trade-off of being exposed to epistemic attacks may still be too great to limit a user's view of the network and we do not recommend it. At the very least, one needs to implement PIR[14] for discovery.

Denial of Service Attacks

The Mixnet version of a DoS attack is an adversary sending many packets into the mix network to cause the mix nodes to become overwhelmed and begin dropping packets. This results in a a network outage until the adversary stops sending so much traffic. A typical response is rate limiting individual clients. However this only stops the DoS attack from being conducted by a single client entity. The adversary could still DDoS the network by using many clients to send packets. Within the network, individual links should also be rate-limited. This will typically ensure that DDoS attacks can only target the first layer of the network, since the traffic is then unlikely to be relayed further.

Sybil Attacks

The adversary plants a large number of malicious nodes, and is therefore able to glean partial or complete information to follow a message through the mix network. This can be mitigated by preventing mix nodes from automatically joining the network, and implementing some kind of trust requirement.

Compulsion Attacks

A compulsion attack involves an adversary exercising force outside of the network to compel node operators to turn over information of control over their nodes. A basic mitigation strategy here is to diversify the locations and control of the nodes, so that no one force can compel enough of them.

[14]PIR: Private Information Retrieval

Another way to reduce the impact of compulsion attacks, is to employ forward security, wherein the mix node destroys the private key material immediately after mixing. Both the client and the mix must keep track of their own states and when to use them with which node. If the client doesn't have an established ratchet state with each mix node in the network, then the possible routes will necessarily be restricted. This is discussed in [M9]. Once the node destroys the key information, compulsion attack on the node for those messages becomes impossible. Future protocols might employ *compulsion traps* combined with the path burning messages, such that the compulsion trap sends out a path burning message to destroy cryptographic key materials later in the route to prevent a compulsion attack. Jeff Burdges theorized (in an unpublished Lake/Xolotl paper) about forward secure mixes and developed a few ideas about post quantum forward secure mixes.

Cryptographic Attacks

The holy grail would be anonymity that is independent of the adversary being able to break cryptographic protocols. We refer to this as *information-theoretic security*. This is sometimes achieved with clever non-cryptographic protocols, as in Dining Cryptographers networks, and sometimes by destroying the relevant information, as in forward security. This is very costly and restrictive, and given the complexity of modern communications, we rely heavily on cryptography for most of the elements of our protocols.

Tagging Attacks

Attacks that involve an adversary manipulating the packet's code en route are typically mitigated with integrity protections: MACs and cryptography that prevents decrypting if a part of the ciphertext was altered. In protocols that employ crossover points, the 1-bit tagging attack performed on a forward message from a client to a service mix, could link the two for a successful adversary. This might serve as partial statistical confirmation that Alice is talking to Bob, depending on the design of the messaging system.

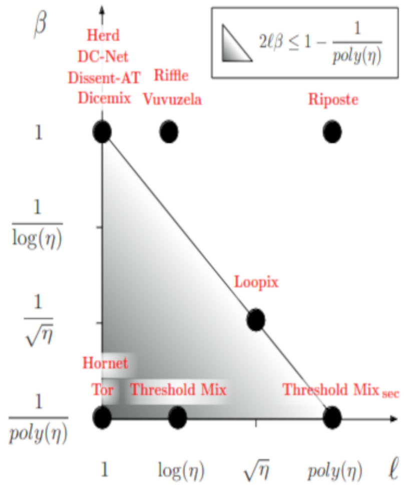
Evaluation of modern mixnets

Anonymity Trilemma

The intuitive premise behind the Anonymity Trilemma - that strong anonymity requires compromising on latency, bandwidth overhead, or both is a useful and widely accepted paradigm. [E1] claims to prove the relation and provide the following lower bound on bandwidth overhead and latency in terms of an anonymity factor η:

$$2\beta\ell \leq 1 - \frac{1}{P(\eta)},$$

where β is the bandwidth overhead, ℓ is latency and $P(\eta)$ is polynomial in η. The paper operates on a toy model, assuming the adversary does not employ statistical methods and only counts a user in an anonymity set or not, in a binary way. Elevating this analysis to a realistic mechanism remains an open problem. Nevertheless, we include a diagram from [E1] (right), which proposes a rating of various anonymity systems' efficiency and efficacy.



Privacy Notions

In the *Mixnets basics* section, we provided an intuitive introduction to terms like *unlinkability* and *unobservability*. We can define these terms formally. In published literature, these are typically referred to as *privacy notions*. Several papers endeavored to provide a formal framework for these notions. [E2] classified them in a hierarchy, the gist of which is "unobservability implies unlinkability, both-sides unlinkability implies each sender unlinkability and receiver unlinkability etc." Not all of the claims of the hierarchy are correct, and in the paper they are artifacts of the way the proofs are constructed as adversary-challenger games with some arbitrary choices made along the way. But to be fair, all papers exploring this question fall into that trap. The shortcomings of this paper are explored in the Extended Commentary section at the end of this work, including questionable framework, incorrect phrasing and straight up math mistakes.

We will now propose a straightforward definition of anonymity notions in terms of probability from the point of view of the observer. This is a direct application of Kolmogorov relations, although in anonymity literature it is sometimes attributed to [E3], crediting them for re-inventing the wheel.[15]

In probability theory, randomness is always in the eye of the beholder and so observer-specific uncertainty is an essential way to define it without arbitrary and controversial restrictions. Defining an anonymity notion obeys the following pattern. Suppose that the adversary $\mathcal{A}$ makes a guess whether an event X occurred based on the available information.

$\mathbb{P}[X_{\mathcal{A}}]$ is the probability that X occurred based on the information available to $\mathcal{A}$.

$\mathbb{P}[X_{\mathcal{A}}|X]$ is the probability that X occurred according to $\mathcal{A}$, provided that X actually occurred,

$\mathbb{P}[X_{\mathcal{A}}|X']$ as the probability that X occurred according to $\mathcal{A}$, provided that X did not occur.

We would like to say that for some $\delta \geq 0$ small enough to satisfy our anonymity requirements,

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

In a perfect system, we would like $\delta = 0$, which would mean that the adversary $\mathcal{A}$ has exactly the same knowledge whether the event X occurred or not with probability 1, which would make the event X and the event of the adversary guessing that X occurred independent events.

If $\mathbb{P}[X_{\mathcal{A}}|X] > \mathbb{P}[X_{\mathcal{A}}|X']$, we refer to the difference as the *adversary advantage.*

The event X could then be:

- $X :=$ message m exists. For message unobservability.
- $X :=$ User A communicates with User B. For sender-receiver unlinkability.
- $X :=$ User A sent the message m. For sender-message unlinkability.
- $X :=$ User A sent a message. For sender unobservability.
- $X :=$ User B received the message m. For receiver-message unlinkability.
- $X :=$ User B received a message. For receiver unobservability.

In the Extended Commentary section, we go through an example in detail, by calculating adversary advantage on the receiver observability in Loopix.

[15] Admittedly, this problem goes both ways as some of the greatest minds in probability theory claimed to introduce the entire field of anonymity in 2014 [E4], with a fresh but not very useful angle and a cute new name - *cryptogenography*.

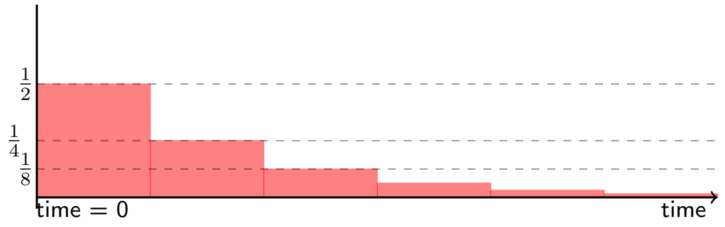
Extended commentary on the published research (by E. J. Infeld)

Loopix

(Failed) memorylessness of the mix

As promised in our summary of the Loopix design, we will now explore the relationship between the exponential distribution and Poisson processes, in order to convince the reader the name *Poisson mix* should be changed to *Memoryless mix**ing*** instead. And like much in probability theory, it all starts with a coin toss.

Suppose we are tossing a coin, and it lands on heads with $1/2$ probability and tails with the remaining $1/2$ probability. And suppose we will keep tossing it until the first time it lands on heads, and then stop. Let the total number of tosses, finishing with the first toss that lands heads, be the discrete random variable X. An astute student of probability will know, that the probability of the first toss landing heads is $1/2$, the first landing tails and then second landing heads is $1/4$, and so on, with the probability of the the first $i-1$ tosses landing tails and and ith landing heads being $(1/2)^i$. So the probability distribution of how many tosses it will take for us to stop tossing the coin decays exponentially. Suppose we are making these tosses at a rate of one toss per second.



The expected number of tosses we make is then:

$$\mathbb{E}[X] = \sum_{i=1}^{\infty} \frac{1}{2^i} \times i = \sum_{i=1}^{\infty} \frac{1}{2^i} + \sum_{i=2}^{\infty} \frac{1}{2^i} + \sum_{i=3}^{\infty} \frac{1}{2^i} + \dots =$$

$$= 1 \times \sum_{i=1}^{\infty} \frac{1}{2^i} + \frac{1}{2} \times \sum_{i=1}^{\infty} \frac{1}{2^i} + \frac{1}{4} \times \sum_{i=1}^{\infty} \frac{1}{2^i} =$$

$$= \left(1 + \frac{1}{2} + \frac{1}{4} + \dots\right) \sum_{i=1}^{\infty} \frac{1}{2^i} = 2 \times 1 = 2.$$

We can refer to this process formally as consecutive Bernoulli trials until first success. Now suppose that instead we are making coin tosses at the rate of every half a second, but the probability of the coin landing heads is $1/4$. The expected number of tosses, calculated the same way, is 4, but since our time increment is now half a second, the expected time to success is still 2 seconds.

$$\mathbb{E}[X'] = \sum_{i=1}^{\infty} \frac{1}{4} \times \left(\frac{3}{4}\right)^{i-1} \times i = \frac{1}{4} \times \sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^{i} \times \sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^{i} = \frac{1}{4} \times 4 \times 4 = 4.$$

Both of these distributions decay geometrically, and the expected time to success is the same. We can continue this process - as the time increments get smaller and smaller and we decrease the probability of a successful toss accordingly to keep the expected time to success the same - these discrete distributions will tend to a continuous limit. That continuous limit is the exponential distribution $f(x) = \lambda e^{-\lambda x}$ with parameter $\lambda = \frac{1}{\mathbb{E}[X]}$.

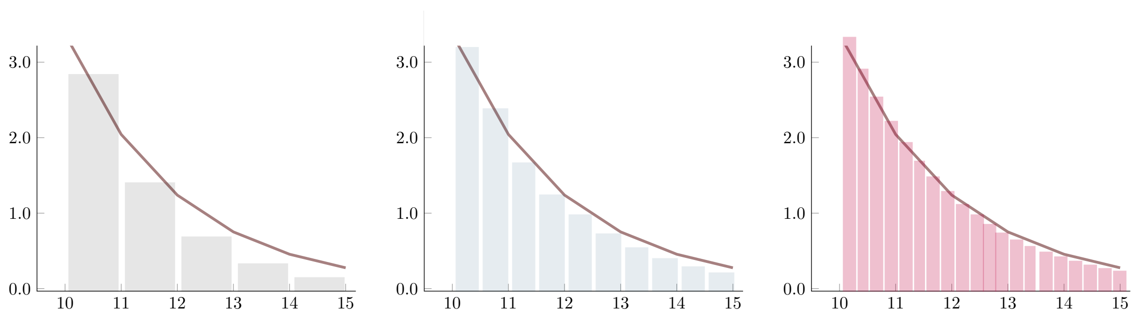
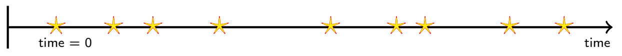


Figure 5: Probability distributions of first success in consecutive Bernoulli trials, starting at $t = 10s$ (because the first terms are messy and don't matter in the limit) for probability of success 0.5, 0.25 and 0.125 respectively, with time increments decreased accordingly, plotted against the exponential distribution with $\lambda = 0.5$.

The process of repeatedly tossing a coin until it comes up heads is memoryless - it doesn't matter how many times you already got tails, if you're still tossing, the probabilities for the next toss are the same. Similarly, if a mix node was repeatedly rolling dice to release a message in the next (very small) time increment with probability p, how long the message will be waiting till release would not depend on how long it has already been sitting there. If we imagine that at time t' there are k messages sitting at a mix node, and we're rolling dice independently for each to either release it or not, it would not matter in what order these messages arrived at the node, or how long ago. And so the first outgoing message could be any one of them with uniform probability. In its continuous approximation this is the mix that employs independent exponential delays.

Now, a Poisson process is a series of events of this type distributed in time. This again relates to the coin toss (or, more formally, Bernoulli trials.) If we were tossing the coin relentlessly, not stopping at any point, the event of the coin coming up heads could be marked on a time axis like so:



The limit of the process of tossing the coin with a set expected time to success at smaller and smaller time increments and decreasing the probability p proportionally, is called a Poisson process. There are countless things in nature that can be modelled this way, for example particle decay. The distances between consecutive events are sampled from the exponential distribution. On the other hand, a Poisson distribution describes how many events of this nature occurred in a time for which the expected number of events is some parameter λ. Since it's counting, the Poisson distribution itself is not memoryless.

Poisson processes are common, and likewise there are many ways to set up a relay releasing messages to look like a Poisson process. It could first roll the dice on whether to release a message or not, and then on success choose which message according to any algorithm. It could even be a queue, in which case we don't achieve any mixing. Or it could choose the message uniformly at random, in which case we do.

On the other hand, in the Loopix model, each message has its own independent exponential delay. The overall behavior of a mix node isn't necessarily a Poisson process, because depending on the number of messages in the system, and the node itself, the expected delay to the next message being released from a node is likely to fluctuate. The overall behavior will only be Poisson distributed if the number of messages in a node at each point can be approximated to be constant. So not only does the name *Poisson mix* not correspond to the memorylessness of the setup, it's not even true. That's why we insist on calling it a *memoryless mix* instead.

[Next two pages of commentary are temporarily withheld.]

Privacy Notions

I feel compelled to point out a few shortcomings of [E2]. The paper is very hand-wavy, which is unacceptable for a work that claims to provide a formal framework. Let's consider a paragraph already on page 3:

> (...) Thus, we need to show for the applied protection measure, that compared to any other sender of the message, it is not more probable that Alice is the sender. We analyze the worst case: in a group of users, let Charlie be a user for whom the probability of being the sender differs most from Alice's probability. If even these are too close to distinguish, Alice is safe, since all other probabilities are closer.

This paragraph should never have made it into a research publication. One, it should say "any other *possible* sender of the message." Two, it starts out saying "it is not more probable" to then assume it is in fact the most probable, but all other probabilities are withing an error margin, which it doesn't say explicitly. A reader would not be wrong to wonder, what if Charlie's perceived probability is, in fact, higher, which it fails to either account for or exclude, but which can very much happen in practice. It also fails to adequately define what "probable" means, even though the paper does mention an adversary observer much earlier, so we are meant to fill in the gaps.

Unfortunately, the rest of the paper isn't much better.On pages 5 and 6, we have a math error with "0" an "1" transposed. There is a claim that

$$|Pr[0 = \langle \mathcal{A}|Ch(\Pi, X, c, 0)\rangle] - Pr[0 = \langle \mathcal{A}|Ch(\Pi, X, c, 1)\rangle]| \leq \delta$$

is equivalent to the following two equations together:

$$Pr[0 = \langle \mathcal{A}|Ch(\Pi, X, c, 0)\rangle] \leq Pr[0 = \langle \mathcal{A}|Ch(\Pi, X, c, 1)\rangle] + \delta,$$

$$Pr[1 = \langle \mathcal{A}|Ch(\Pi, X, c, 1)\rangle] \leq Pr[1 = \langle \mathcal{A}|Ch(\Pi, X, c, 0)\rangle] + \delta.$$

But these two latter equations mean the same thing. The authors appear to have meant the last one to be:

$$Pr[1 = \langle \mathcal{A}|Ch(\Pi, X, c, 0)\rangle] \leq Pr[1 = \langle \mathcal{A}|Ch(\Pi, X, c, 1)\rangle] + \delta.$$

This is a trivial argument that is stretched to an entire section, one might as well try to get it right.

The paper's nomenclature is messy, inconsistent, and needlessly complicated, and uses words and symbols customarily used differently in mathematics. It defines some of the notions with arbitrary restrictions, say, from page 8 on it is clear that its definition of sender-receiver linkability requires that the adversary identifies a specific message from the sender to the receiver, which should not be a requirement, least of all in a probabilistic setting. You don't need to identify a specific message to know that they exist.

This is an artifact of the framework, and only one of the issues with it. Defining anonymity in terms of adversary-specific indistinguishability is natural, and has been widely explored in probabilistic combinatorics. But the game constructs in the paper are contrived and arbitrary. This ripples through the paper, and affects the resulting hierarchy. There are relations in it that are blatantly false, such as, to stick to the above example, a claim that sender-receiver linkability implies message observability, which the reader hopefully accepts is nonsense.

And even more outrageous problem is that the relation described in the hierarchy is incorrectly described as an implication, rather than a marking stricly stronger property, leading us to follow through the graph to conclusions like "message unobservability implies message observability," which is clearly false.

Bibliography

Mixnet design research

[M1] 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.

[M2] Ania Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The loopix anonymity system, 2017.

[M3] 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.

[M4] George Danezis. Mix-networks with restricted routes. In Roger Dingledine, editor, *Privacy Enhancing Technologies*, pages 1–17, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[M5] 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.

[M6] 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.

[M7] Ross Anderson and Eli Biham. Two practical and provably secure block ciphers: Bear and lion. In Dieter Gollmann, editor, *Fast Software Encryption*, pages 113–120, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[M8] George Danezis and Ben Laurie. Minx: a simple and efficient anonymous packet format. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, WPES '04, page 59–65, New York, NY, USA, 2004. Association for Computing Machinery.

[M9] George Danezis. Forward secure mixes. 10 2002.

Selected sources on attacks

[A1] Birgit Pfitzmann and Andreas Pfitzmann. "how to break the direct rsa-implementation of mixes.". In *Lecture Notes in Computer Science 373-381*, 1996.

[A2] Erik Shimshock, Matt Staats, and Nick Hopper. Breaking and provably fixing minx. In *Proceedings of the 8th International Symposium on Privacy Enhancing Technologies*, PETS '08, page 99–114, Berlin, Heidelberg, 2008. Springer-Verlag.

[A3] Christiane Kuhn, Martin Beck, and Thorsten Strufe. Breaking and (partially) fixing provably secure onion routing, 2019.

[A4] Carmela Troncoso and George Danezis. The bayesian traffic analysis of mix networks. pages 369–379, 11 2009.

[A5] George Danezis and Andrei Serjantov. Statistical disclosure or intersection attacks on anonymity systems. In Jessica Fridrich, editor, *Information Hiding*, pages 293–308, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[A6] Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a trickle to a flood: Active attacks on several mix types. volume 2578, 02 2003.

[A7] George Danezis and Paul Syverson. Bridging and fingerprinting: Epistemic attacks on route selection. volume 5134, pages 151–166, 07 2008.

[A8] George Danezis and Jolyon Clulow. Compulsion resistant anonymous communications. pages 11–25, 06 2005.

[A9] Johan Helsingius. https://en.wikipedia.org/wiki/penet_remailer.

[A10] Joan Feigenbaum, Aaron Johnson, and Paul Syverson. Preventing active timing attacks in low-latency anonymous communication. In Mikhail J. Atallah and Nicholas J. Hopper, editors, *Privacy Enhancing Technologies*, pages 166–183, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[A11] Vitaly Shmatikov and Ming-Hsiu Wang. Timing analysis in low-latency mix networks: Attacks and defenses. volume 4189, pages 18–33, 01 2006.

[A12] https://en.wikipedia.org/wiki/shor's_algorithm.

Mixnet evaluation

[E1] 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`.

[E2] Christiane Kuhn, Martin Beck, Stefan Schiffner, Eduard Jorswieck, and Thorsten Strufe. On privacy notions in anonymous communication. 2019.

[E3] Michael Backes, Aniket Kate, Praveen Manoharan, Sebastian Meiser, and Esfandiar Mohammadi. Anoa: A framework for analyzing anonymous communication protocols. In *2013 IEEE 26th Computer Security Foundations Symposium*, pages 163–178, 2013.

[E4] Joshua Brody, Sune K. Jakobsen, Dominik Scheder, and Peter Winkler. Cryptogenography. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science*, ITCS '14, page 13–22, New York, NY, USA, 2014. Association for Computing Machinery.