Aimless Onions: Mixing without Topology Information

Daniel Schadt
Karlsruhe Institute of Technology
KASTEL Security Research Labs
Karlsruhe, Germany
daniel.schadt@kit.edu

Christoph Coijanovic
Karlsruhe Institute of Technology
KASTEL Security Research Labs
Karlsruhe, Germany
christoph.coijanovic@kit.edu

Thorsten Strufe
Karlsruhe Institute of Technology
KASTEL Security Research Labs
Karlsruhe, Germany
thorsten.strufe@kit.edu

Abstract

Mix networks allow communication with strong anonymity guarantees. In theory, mix networks can scale indefinitely, as additional nodes can be added to the network to support new users. However, one factor that limits scalability in current designs is the need for all clients to know both the identity and the key of every available mix node.

In circuit-based onion routing, a mechanism that does not require this knowledge to be globally available exists, but it relies on the interactivity of the circuit construction to keep its security guarantees. We therefore set out to investigate whether we can transfer such a mechanism to the context of message-based mix networks.

In this paper, we propose Aimless Onions, the first mix format that enables clients to create onions in a mix network without knowing which nodes are available. Rather than downloading topology information, clients only need to acquire constant-size public parameters. Thus, Aimless Onions overcomes an important scalability limitation in mix networks, while retaining the same security guarantees as the state of the art. Using Aimless Onions, clients sending 25 messages per hour save 74% of bandwidth compared to using Spinx packets and topology information download, even at today's network sizes.

1 Introduction

Mix networks have become a widely used technique for practical anonymous communication networks (ACNs), as demonstrated by Loopix [22] or the deployed Nym system.[1] An architectural advantage of mix networks over other anonymity designs is that they can theoretically scale to support arbitrarily large user bases. As new users join the network, additional mix nodes can be deployed so that the load on all nodes remains constant. Good scalability not only improves usability, but also privacy, as a large user base allows clients to hide among a larger number of peers, and trust is spread across a larger number of nodes.

In practice, the scalability of mix networks is limited by the path selection process (among other factors). During path selection, it is important that the mix nodes for each path are chosen randomly from *all* available nodes, otherwise an adversary can fingerprint users based on their choices [12]. Current solutions require clients to download a complete list of all mix nodes along with their public keys. This is a suitable solution for today's ACNs, but even the current Tor network topology information is 632 KiB in size (compressed) and updated hourly. The size of the topology information grows linearly with the number of nodes, and if ACNs reach mainstream adoption, it is not unreasonable to expect it to be an order of magnitude larger than today.[2] At this point, alternative path selection methods will have to be in place.

To avoid downloading topology information in Tor, Komlo et al. propose *Walking Onions* [19]. In Tor, a circuit is first constructed and the authenticity of each relay on it is verified before it is used for sensitive data. With Walking Onions, the selection of the relays is also delegated to the (potentially malicious) relays, but verified by the client to be random and not maliciously biased. This provides clients with a factor of 10–16 in bandwidth savings.

In mix networks however, onions are prepared and sent by the client in one go, without establishing a circuit first. Such networks are advantageous in use cases with low interactivity, such as email or microblogging, where little data would be sent over the lifespan of a circuit. However, the Walking Onions method cannot be used here, as the onion already contains all sensitive data when it is created, and the sender relinquishes control once they submit it to the network.

Therefore, we investigate how the Walking Onions method could be transferred to a mix network design. To do so, we design *Aimless Onions*, an approach that avoids downloading topology information in mix networks and does not require interactivity during path selection. The challenge in Aimless Onions is to find a way for a client to properly encrypt and prepare an onion for a path of mix nodes whose addresses and keys it does not know, without giving malicious mix nodes the chance to influence the path selection.

At a high level, Aimless Onions defines a large namespace of virtual names, e.g., all 32-bit integers, as valid onion targets. A set of authorities, similar to those in Tor, maps each virtual name to a real mix node. The number of names a mix node receives is used to weight it, e.g., depending on its available bandwidth. Clients use *identity-based encryption* (IBE) [5] to encrypt onions directly to arbitrary virtual names in the namespace. This way, clients do not need to know which real nodes exist or what their public keys are. The authorities collectively generate the IBE secret key material, tolerating malicious minorities. The key material is distributed to the corresponding mix nodes so that they can decrypt incoming onions.

Aimless Onions makes the same trust assumptions as related work [19], with an anytrust model along the path and an honest majority among the authorities. The client only needs to download a single long-lived key per authority, which is independent of the

[1] https://nymtech.net/

[2] Compare Tor's 2 million active users to WhatsApp's 3 billion.

Proceedings on Privacy Enhancing Technologies 2025(4), 293–307

https://doi.org/10.56553/popets-2025-0131

number of existing mix nodes and has a small size of around 3 KiB (compared to the 632 KiB of Tor consensus). Thus, Aimless Onions solves the limitations of scalability in mix networks that are caused by the path selection process.

Through empirical evaluation, we further find that Aimless Onions saves bandwidth for clients even at today's anonymity network sizes for traditional mix network use cases such as email. While individual Aimless Onions are larger than their Sphinx counterparts by a factor of up to 11, the bandwidth saved by not downloading the network topology makes up for it. For example, given a Tor-like network with 9 authorities and a path length of 3, clients sending fewer than 105 onions per hour can save bandwidth.

In this paper we make the following contributions:

- We propose Aimless Onions, the first mix format that enables the construction of onions without needing network topology information.
- We provide security proofs for Aimless Onions based on Kuhn et al.'s formal model.
- We build an open-source implementation prototype of Aimless Onions.
- We evaluate the Aimless Onions prototype and compare it against state-of-the-art mix formats under realistic assumptions.

This paper is structured as follows: In Section 2 we introduce the necessary background and notation. In Section 3 we look at previous work related to Aimless Onions. In Section 4 we model the network and the adversary. In Section 5 we give a high-level description of our design and design choices. In Section 6 we describe our format in more detail. In Section 7 we describe how onions are formed at the client and processed at the mix nodes. In Section 8 we sketch a security proof of our format. In Section 9 we evaluate our format using our implementation. In Section 10 we discuss possible extensions to our format. Finally, in Section 11 we conclude our findings.

2 Background

In this paper, we work with bilinear groups. We denote the groups as $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, and the mapping as $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We assume that the l-wBDHI assumption [4] holds, which intuitively states that you cannot compute the mapping $e(g,h)^{(\alpha^l)}$ when given g, h and $g^{(\alpha^i)}$ for i from 1 to $2l$, but $i \neq l$.

We use κ as security parameter. Breaking security guarantees should require work that scales exponentially in κ.

When working with bitstrings $a, b \in \{0,1\}^*$, we write $a \,||\, b$ to mean the concatenation of a followed by b. Further, given $x, y \in \mathbb{N}$, we write $a_{[x..y]}$ to denote the substring of a spanning from x to y (exclusively).

2.1 Mix networks

Mix networks are a method of anonymous communication introduced by Chaum [9]. Mix networks guarantee that messages cannot be traced from sender to recipient by routing them through a series of *mix nodes*. Senders apply multiple layers of encryption to each message, which are successively removed by the mix nodes to ensure that messages cannot be linked by their content. This concept is known as *onion encryption*. Each mix node further shuffles the incoming messages before forwarding them to ensure that they cannot be linked by their order. A mix network operates under the *anytrust* assumption, meaning that a single mix node on the path must be honest for the security guarantees to hold.

Clients in a mix network must know the addresses and keys of available mix nodes. We use the term *consensus* to refer to the document that contains this information for the clients. The exact method to generate or disseminate a consensus is an implementation detail of the mix network, but we assume that each client joining the network needs to download the consensus and regular updates for it.

2.2 Pseudorandom generators

We make use of pseudorandom generators (PRNG). We define PRNGs based on Katz and Lindell's definition [17]:

Definition 1 (PRNG [17]). *A PRNG ρ is a polynomial time algorithm such that on input $s \in \{0,1\}^\kappa$, a string of length $l(\kappa)$ (with l being a polynomial) is output. We require that*

(1) *$l(\kappa) > \kappa$ for all κ.*
(2) *For all polynomial time distinguishers D, $|\Pr[D(r) = 1] - \Pr[D(G(s)) = 1]|$ is negligible when $r \xleftarrow{R} \{0,1\}^{l(\kappa)}$ and $s \xleftarrow{R} \{0,1\}^\kappa$.*

Intuitively, a PRNG takes a short input seed and outputs random looking bits. We usually use the PRNG to XOR its output onto another bitstring, in which case we write $a \oplus \rho(\ldots)$ to mean $a \oplus \rho(\ldots)_{[0..|a|]}$.

2.3 IBE and HIBE

Identity-Based Encryption (IBE) describes encryption schemes in which the key of a user can be derived directly from their identity.

Definition 2 (IBE [5]). *An IBE scheme consists of four algorithms:*

(1) *$\texttt{Setup} : 1^\kappa \to \mathcal{P} \times \mathcal{K}^*$ returns the system parameters and the master secret key. The system parameters are assumed to be an implicit argument to the following algorithms.*
(2) *$\texttt{Extract} : \mathcal{K}^* \times \mathcal{I} \to \mathcal{K}$ takes the master key and an identity and extracts the private key for the given identity.*
(3) *$\texttt{Encrypt} : \mathcal{I} \times \mathcal{M} \to \mathcal{C}$ takes an identity and a message and returns the message encrypted for the given identity.*
(4) *$\texttt{Decrypt} : \mathcal{I} \times \mathcal{C} \times \mathcal{K} \to \mathcal{M}$ takes a ciphertext and an identity together with the corresponding private key and decrypts the given ciphertext.*

For an IBE to be correct, it is required that

$$\forall i \in \mathcal{I}, (p,k) \leftarrow \texttt{Setup}, m \in \mathcal{M} :$$
$$\texttt{Decrypt}(i, \texttt{Encrypt}(i,m), \texttt{Extract}(k,i)) = m.$$

In a hierarchical IBE (HIBE), the identities form a hierarchy, and `Extract` can extract an identity's private key given either the master key, or the private key of the identity's parent.

2.4 Secret sharing

A (k,a) secret sharing scheme is a method for splitting a datum d (the *secret*) into a shares such that k shares are sufficient to recover the datum d, but $k-1$ shares or fewer do not reveal information about d [25].

Shamir proposed a method of secret sharing that works by constructing a polynomial of degree k with the secret as a free coefficient, and then evaluating it at a different points to produce the shares. Knowledge of k point-value pairs is enough to reconstruct the polynomial and thus the secret, but $k-1$ pairs are not enough to fully specify the polynomial.

We formally define a secret sharing scheme as follows:

Definition 3 (Secret Sharing). *A (k, a) secret sharing scheme consists of two algorithms:*

1. $\texttt{Split} : \mathcal{D} \to \mathcal{S}^a$ *takes a secret and produces a shares.*
2. $\texttt{Combine} : \mathcal{S}^k \to \mathcal{D}$ *takes k shares and produces the original secret.*

We define the security of secret sharing according to Boneh and Shoup's notion of security, which is a formalization of Shamir's original guarantee:

Definition 4 (Secret Sharing Security [6]). *A secret sharing scheme is secure, if for all parameters $0 < k \leq a$, for every $\sigma, \sigma' \in \mathcal{D}$ and every subset J of $\{1, \ldots, a\}$ of size $k-1$, the distributions $\{\Sigma_i\}_{i \in J}$ and $\{\Sigma'_i\}_{i \in J}$ are identical (where $\Sigma = \texttt{Split}(\sigma)$ and $\Sigma' = \texttt{Split}(\sigma')$).*

3 Related work

Aimless Onions's goal is to allow clients to participate in a message-based mix network without needing to know topology information. In this section, we discuss related work with similar goals.

Komlo et al. propose *Walking Onions* for Tor.[3] Like Aimless Onions, Walking Onions allows clients to participate without downloading the network consensus first. In Walking Onions, paths are successively built hop-by-hop, where the next hop is chosen randomly by the current one. However, its security inherently requires the client to verify the path before sending sensitive data, which is not possible in our message-based setting.

The Threshold Pivot Scheme [15] is a mix network that combines mix nodes into groups that share key material. Clients only need to know which groups exist (rather than knowing individual mix nodes) and they generate onions that can be processed by any one node of the group. While this significantly reduces the amount of topology information the client needs to know, it does so at the cost of anonymity, as a single malicious node taints the whole group.

Finally, there is a line of research which, like Aimless Onions, proposes to use identity-based encryption in onion routing [7, 8, 16]. They focus however on reducing the number of round trips needed for Tor's circuit creation. Clients still need to know the full network consensus.

4 System and threat model

We propose Aimless Onions as a generic mix format that can be used as the basis for new mix networks. Therefore, we leave the type of mixing, the amount of cover traffic, and the exact parameter choices as out of scope for this work. We do however assume a certain basic structure of the network.

We assume that the network consists of three groups of participants: Clients which use the service, mix nodes which shuffle and forward messages, and authorities which keep track of the available mix nodes in the system. We assume that the authorities have a consistent view of the network, but we do not require a specific method of synchronization.[4] This mirrors a simplified version of Tor's [13] model.

We assume that clients are the ones with the most limited resources like computing power and network bandwidth. As such, we require our format to be efficient at the client's side. We assume that clients want to send few but large messages, e.g., to communicate via emails.

We design our system in a "service model" [20], where the final recipient is not part of the anonymity network but merely a host on the internet. Instead, it is the task of the last mix node to relay the message according to the application protocol to the intended recipient. This corresponds to the function of exit relays in Tor, and matches how Sphinx and anonymous remailers like Mixminion [10] were designed to be used.

In order to deal with new mix nodes joining the network and old mix nodes going offline, we assume that the network can run in *epochs*. During an epoch, the set of active mix nodes does not change – new mix nodes will be included in the next epoch, while mix nodes that go offline will be removed. In practice, mix nodes may go offline unexpectedly during an epoch, which will lead to delivery failures for onions routed over those mix nodes. A similar approach is used by Tor, which produces a new consensus once per hour.

Instead of fixed mix cascades, we assume a free-route mix topology, in which each mix node can connect to every other mix node, and each mix node can appear at any position in the mixing path.

We assume that a certain fraction of mix nodes are malicious, meaning they cooperate with the adversary and share their secret keys. Additionally, we assume that there are malicious authorities, but that the *majority* of them are honest, again mirroring the threat model that Tor assumes. We also assume that the adversary can monitor the network traffic of all clients and mix nodes, malicious or not. Given those capabilities, the adversary tries to link messages to both their sender and their recipient.

5 Design

Our fundamental idea is to allow for clients to freely choose a sequence of mix nodes as the path for their onion, without them needing to download a consensus document first. Not knowing the identities nor keys of the available mix nodes, we employ a name-based assignment in which the client can simply draw random names from a circular namespace. Those names then relate to mix nodes that are online.

As keying material needs to be available in order to build the onion, but the client only draws random names, we allow the client to extract the necessary keying material from just the names. For this purpose, we leverage identity-based encryption (IBE) such that the client can locally derive all needed keys for the mix nodes. We design the circular namespace such that its names correspond to the IBE identities.

[3] Tor Proposal 323, https://spec.torproject.org/proposals/323-walking-onions-full.html

[4] In particular, we consider the exact method for agreeing on a network state to be out of scope. An obvious approach would be to use a majority vote as is done in Tor.

Now, IBE schemes assume the existence of a trusted third party which acts as the *private key generator*. This key generator is responsible for generating the correct secret keys for each identity using the master secret. In our system, we entrust the authorities with this task: The authorities need to keep track of the available mix nodes anyway, and mix nodes need to inform the authorities about their state. Therefore, having the key generation done by the authorities is a sensible choice.

With the authorities being in control of the master secret however, they also gain the ability to generate the secret keys for any mix node of their choosing. A malicious authority could therefore undermine the security of the whole system by revealing the secret keys of honest mix nodes. To prevent this attack, we ensure that a single malicious authority cannot generate enough secret key material to overrule the honest authorities. We do so by combining the IBE encryption with (k, a) secret sharing, such that at least k authorities need to cooperate in order to reveal secret information. With $k > a/2$, we retain the trust model in which an honest majority of authorities is enough to keep the system secure.

With the authorities responsible for generating the keys, we also need to ensure that a single authority cannot disrupt the network by not providing the mix nodes with their keys. By choosing the threshold lower than the total number of authorities ($k < a$), our design can tolerate authorities that are unavailable.

Further, with the design so far, the clients still operate on an abstract namespace, whose elements are not related to available mix nodes yet. Aimless Onions therefore requires a mapping $\iota : \mathcal{I} \to \mathcal{R}$ between the IBE identities $\mathcal{I}$ and the set of mix nodes $\mathcal{R}$ to be constructed. We again delegate this task to the authorities, as they have the list of available mix nodes. The mapping ι therefore relates the client's randomly drawn names to actual mix nodes.

Since the IBE identities $\mathcal{I}$ and the set of mix nodes $\mathcal{R}$ are independent of each other, there will not be a direct one-to-one mapping – and since the client chooses identities randomly without knowing which ones are actually in use, the authorities must ensure that all identities map to a mix node. As a result, a given mix node can have multiple identities assigned to it.

Additionally, since clients draw the mix node names randomly without additional information, they cannot weight their decision by node properties such as the available bandwidth. To retain this ability, the namespace is chosen to be very large, i.e. $|\mathcal{I}| \gg |\mathcal{R}|$. The authorities can then weight the mix nodes by assigning them a larger or smaller number of identities relative to the other mix nodes.

As a result of this technique, the identity space can get very large (e.g. $|\mathcal{I}| = 2^{32}$). This would require the authorities to generate just as many keys for the mix nodes, which would take a lot of computation time. To avoid generating every single key, Aimless Onions uses an IBE which supports hierarchical key derivation (a HIBE). Now the authorities can "combine" the keys for mix nodes by giving them a key higher up in the hierarchy, and the mix nodes can then locally derive the remaining keys. As keys in the hierarchy can only be combined if they share a common ancestor, we optimize the identity-mapping ι to group keys for a single mix node under a common parent as much as possible.

Finally, the authorities can create disjunct subsets of the identity space to group mix nodes by properties such as their ability to handle exit traffic or their jurisdiction. By having the prefix bits of the identity represent those properties, clients can ensure that a mix node with the right properties handles the onion, while the mix node's keys can still be combined using the HIBE technique.

6 Format

In this section we describe the format of our onions. In general, we follow the same principles as in existing formats like Sphinx [11] by using layered encryption to provide unlinkability, and MACs to prevent tagging attacks. The main differences in our format are that we combine HIBE and secret sharing to encrypt each layer, and that we include the payload in the MAC to avoid Sphinx's security pitfalls regarding tagged payloads [20]. As a result, we exclude reply blocks (SURBs) from our construction, and leave "aimless" repliable onions [21] open for future research.

As we ensure a constant fixed packet size, we impose a limit on the path length: A path may not contain more than V hops. This upper limit ensures that we can bound the size of a packet.

6.1 Packet structure

We consider a packet to consist of two parts, a header H and the payload Π. The header contains the information necessary to process a packet at a mix node, whereas the payload contains the (encrypted) payload data.

The structure of our format is shown in Figure 1: All cryptographic operations depend on a randomly drawn *master key*. This master key is provided to the mix nodes at the beginning of the header.

The remaining part of the header then contains the per-hop information, consisting of the ID of the next hop, a MAC to ensure hop-by-hop integrity and the header for the next mix node. In addition to those required fields, a fixed amount of additional data can be included here, for example to provide delays for Poisson mixing [22].

Finally, the packet contains an onion-encrypted version of the payload. As we design our format in the service model, this payload is revealed to the last mix node on the path, and as such, contains the address of the recipient and the message. We note that our format does not provide end-to-end confidentiality or integrity between the sender and the recipient, but only between the sender and the last mix node. To provide full end-to-end security, the message must be processed by e.g. PGP or S/MIME before being onion-wrapped.

6.2 Packet and field sizes

For a clearer protocol description, we denote the sizes of the different header components as given in Table 1.

With this, we get a per-hop header size of

$$|H_{\text{hop}}| = a \cdot |\Sigma| + |\mu| + |r|,$$

where a is the number of authorities. The total size of the header is then given as

$$|H| = V \cdot |H_{\text{hop}}|.$$

7 Protocol

We will specify our design, next. We define our name hierarchy on top of binary numbers: A number $x_0 \dots x_{j-1} \in \{0,1\}^j$ is the parent

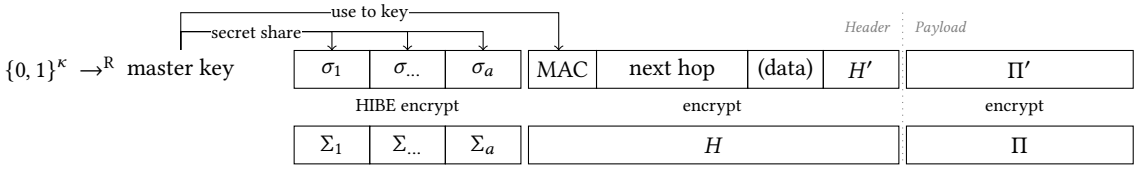

Figure 1: General layout of an aimless onion.

Size of …	Symbol
… a secret share	$\lvert\sigma\rvert$
… an encrypted share	$\lvert\Sigma\rvert$
… the MAC	$\lvert\mu\rvert$
… the next hop	$\lvert r\rvert$

Table 1: Symbols for the sizes of the header elements.

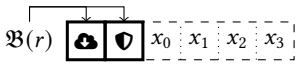

Figure 2: Split of the identity name into flag bits (thick rectangles) and weighting bits (dashed rectangles).

of $x_0 \dots x_{j-1}x_j \in \{0,1\}^{j+1}$. Further, we assume that for a given mix node $r \in \mathcal{R}$, $w(r)$ gives the mix node's desired weight.

7.1 Bucketization

To allow a client to choose a mix node with desired properties (such as allowing exit traffic), we split the name into two parts: We denote the first f bits of the name as *flag bits* and use them to represent a node's properties. The remaining j bits are freely assigned to the nodes by the authorities. We call this part the *weighting bits*. A client who wants to ensure that the selected node has the desired properties can thus fix the flag bits and only choose the weighting bits randomly.

A name is therefore represented by an element of $\{0,1\}^{f+j}$, where f are the number of flag bits, and j are the number of weighting bits. We call all names sharing the same first f bits a *bucket*, and we call $\mathfrak{B} : \mathcal{R} \to \{0,1\}^f$ the bucketization function which assigns each relay its prefix. A visualization of this structure is shown in Figure 2.

7.2 Identity allocation

The authorities first split all mix nodes into their respective buckets with the help of $\mathfrak{B}$. Within each bucket, the authorities then allocate the identities to the mix nodes. In the following, we detail the process for a single bucket.

For each bucket B, we denote the subset of $\mathcal{I}$ representing this bucket as $\mathcal{I}^B$. That is, each $n \in \mathcal{I}^B$ has the same prefix of f bits. Using that, we define how large each mix node's identity allocation should be, proportional to its weight $w(r)$ in its bucket:

$$i(r) = |\mathcal{I}^B| \cdot w(r) \cdot \left(\sum_{r' \in \mathcal{I}^B} w(r') \right)^{-1}.$$

Now, when choosing which $i(r)$ identities to allocate to a given mix node, we want to optimize for the number of needed HIBE keys, as we can only use the hierarchical key derivation for a set of 2^x keys sharing a common prefix. As the flag bits for a bucket are always the same, our allocation algorithm only concerns itself with the weighting bits within a bucket.

We implement an allocation system similar to the buddy allocator [18] to assign names from $\mathcal{I}^B$ to nodes:

First, the authorities allocate the full bucket $\mathcal{I}^B$ to the largest[5] mix node, corresponding to the case in which all identities of a mix node share no common prefix in weighting bits. If no mix node r with $i(r) \geq |\mathcal{I}^B|$ exists, they partition the current bucket in halves, one half $\mathcal{I}^B_0$ where all identities have a weighting bits prefix of 0 and one half $\mathcal{I}^B_1$ where the prefix is 1. They subsequently check if the size of those subsets matches the largest mix node.

If the allocation matches exactly, that is $i(r) = |\mathcal{I}^B_0|$, they allocate $\mathcal{I}^B_0$ to r and are done with r, so they can continue with $\mathcal{I}^B_1$ and the next mix node according to size.

If the mix node should be allocated even more identities, that is $i(r) > |\mathcal{I}^B_0|$, they also allocate $\mathcal{I}^B_0$ to r, but keep r with the reduced size $i(r) \leftarrow i(r) - |\mathcal{I}^B_0|$ in the list of mix nodes that need identities allocated.

If the size of the halves is still too large, that is $i(r) < |\mathcal{I}^B_0|$, they again split the subsets in halves by splitting them according to the next bit in the common prefix, giving the subsets $\mathcal{I}^B_{00}$, $\mathcal{I}^B_{01}$, $\mathcal{I}^B_{10}$ and $\mathcal{I}^B_{11}$.

In all cases, this allocation-and-splitting process is repeated until all needed allocations have been satisfied, or the subsets cannot be further subdivided. At the end, any unallocated subsets are distributed to the mix nodes with largest $i(r)$, even if that gives them more identities than desired. A pseudocode of this algorithm is shown in Algorithm 1.

With this algorithm we prevent that a mix node has many identities that do not share a common prefix, as that would prevent the HIBE mechanism to combine the keys. Instead, the algorithm ensures that a mix node is allocated large amounts contiguous identities which share a common prefix.

In other words, this algorithm treats the identities as leaves of a binary tree of depth $f + j$ and tries to allocate $i(r)$ identities to r in a way that those identities make up a large subtree. This is similar to load balancing using client IP wildcard patterns [26], and the resulting allocation corresponds to the wildcard rules.

The authorities then repeat this process for the remaining buckets.

[5] Largest by bandwidth, breaking ties by sorting their IP addresses lexicographically.

$R \leftarrow \mathcal{R}$; /* set of mix nodes */
$C \leftarrow [\varepsilon]$; /* current shared prefixes */
$s \leftarrow |\mathcal{I}^B|$; /* current subset size */
while $C \neq \emptyset$ **do**
  $r \leftarrow max(R)$ by $i(r)$;
  if $i(r) < s$ **then**
    if $s > 1$ **then**
      $s \leftarrow \frac{s}{2}$;
      $C \leftarrow [\mathtt{c}\ldots\mathtt{0}, \mathtt{c}\ldots\mathtt{1} | c \in C]$;
    else
      $i(r) \leftarrow i(r) - 1$;
      remove and allocate first of C to r;
  else
    $i(r) \leftarrow i(r) - s$;
    remove and allocate first of C to r;

Algorithm 1: Identity-Mix-Node allocation mechanism.

7.3 Primitives

For our construction, we need the following primitives:

- A hash function $h : \{\mathbf{M}, \mathbf{H}, \mathbf{P}, \mathbf{F}\} \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ to derive subordinate keys from the master key. The symbols $\mathbf{M}, \mathbf{H}, \mathbf{P}, \mathbf{F}$ are used to differentiate keys for the MAC ($\mathbf{M}$), the header encryption ($\mathbf{H}$), the payload encryption ($\mathbf{P}$) and the padding computation ($\mathbf{F}$).
- A message authentication code $\mu : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^\kappa$, which we model as a function that takes a key and input bits and outputs a tag.
- A keyed pseudorandom generator $\rho : \{0,1\}^\kappa \to \{0,1\}^*$ as defined in Definition 1.

7.4 Packet creation

To create a packet, the client first picks v random identities r_i from $\mathcal{I}$. These represent the path of the onion through the network.

The client now computes the onion with the following steps:

(1) The client has the recipient address Δ and the message M.
(2) The client draws v random keys $k_i \xleftarrow{R} \{0,1\}^\kappa$. These represent the master keys for each hop.
(3) The client computes the filler strings Φ_i like in Sphinx [11]:

$$\Phi_0 = \varepsilon$$
For $i = 1$ to v:
$$\Phi_i = \left(\Phi_{i-1} \,||\, \mathtt{0}^{|H_{\text{hop}}|}\right) \oplus \rho\left(h\left(\mathbf{H}, k_{i-1}\right)\right)$$

(4) The client fills the initial header with random bits:

$$H_v \xleftarrow{R} \{0,1\}^{|H| - a \cdot |\Sigma|}$$

(5) In reverse order of the path, the client now starts wrapping the onion by calculating for $i = v$ to $i = 1$:

$$\begin{aligned}
\sigma_{i,j} &= \mathtt{Split}(k_i) \\
\Sigma_{i,j} &= \mathtt{Encrypt}(r_i, \sigma_{i,j}, \mathrm{pk}_j) \\
\Pi_v &= (\Delta, M) \oplus \rho(h(\mathbf{P}, k_v)) \\
\Pi_i &= \Pi_{i+1} \oplus \rho(h(\mathbf{P}, k_i)) \\
\eta_i &= (r_{i+1} \,||\, H_{i+1})_{[0..|H| - a|\Sigma|]} \\
H_i &= \Sigma_{i,1} \,||\, \Sigma_{i,\ldots} \,||\, \Sigma_{i,a} \\
&\quad ||\, ((\mu\left(h\left(\mathbf{M}, k_i\right), \eta_i \,||\, \Pi_i\right) \,||\, \eta_i) \oplus \rho\left(h\left(\mathbf{H}, k_i\right)\right))
\end{aligned}$$

(6) The client can now send (r_1, H_1, Π_1) to the first node in the path.

7.5 Packet processing at mix nodes

When a mix node r receives a packet (r, H, Π), it proceeds as follows:

(1) It splits off the a encrypted secret shares Σ_j from the start of H: $\sigma_1, \ldots, \sigma_a, H' = H$
(2) It decrypts each share with the secret key from the j^{th} authority: $\sigma_j = \mathtt{Decrypt}(r, \Sigma_j, \mathrm{sk}_j)$
(3) It recombines the secret shares into the onion master secret: $k = \mathtt{Combine}(\sigma_1, \ldots, \sigma_a)$. If recombination fails, the onion is discarded.
(4) It pads and decrypts the header part: $\eta = \left(H' \,||\, \mathtt{0}^{|H_{\text{hop}}|}\right) \oplus \rho\left(h\left(\mathbf{H}, k\right)\right)$
(5) Now it can parse the MAC, the next address and the next header: $t, r_{+1}, H_{+1} = \eta$
(6) If $t \neq \mu\left(h\left(\mathbf{M}, k\right), r_{+1} \,||\, H_{+1} \,||\, \Pi\right)$ the mix node discards the packet, as it has been tampered with.
(7) Otherwise, the mix node forwards $(r_{+1}, H_{+1}, \Pi \oplus \rho(h(\mathbf{P}, k)))$ to the next node r_{+1}.

7.6 Node address retrieval

So far, we have addressed mix nodes by their identity number $r \in \mathcal{I}$. The identity needs to be translated to a proper network address during operation, so a client can send an onion to the first mix node, and mix nodes can forward onions to the following mix node along the path. For this, we describe two methods:

Clients and small mix nodes can simply ask an authority directly to resolve the single identity-address mapping, akin to doing a "DNS request" before sending the packet. Large mix nodes can download the whole mapping in advance, which avoids frequent round-trips to resolve node identities but incurs a large bandwidth cost. Such mix nodes can also act as "mirrors" which answer requests for small clients, allowing this lookup service to be distributed, fault-tolerant and scalable.

While such requests disclose information to a potentially malicious mirror, we note that within our threat model, this information is already know to the global active adversary: They can observe the link from client to first hop and thus learn which first hop a client has chosen. Further, in all mix network designs, clients are only anonymous within those who share a common prefix before a corrupted node, so learning the mapping between client and first node does not yield any advantage.

7.7 Key distribution

We consider the exact approach with which keys are distributed (e.g., through an HTTP API hosted by the authorities) to be out-of-scope, and dependent on the mix network that Aimless Onions is used in. However, we sketch a design of a possible implementation that fulfills our requirements:

Each authority has a long-lived key pair that it can use to sign data. The list of authorities and corresponding key pairs is embedded in the client software, like done in Tor. Using this key pair, the authorities can generate their master public key each epoch, and make a signed version of it available under their HTTP endpoint. Those keys can also be collected and mirrored by third parties, as the signature can be used to verify their authenticity.

The mix nodes also have the list of authorities and authority keys embedded in the software. They use this information to establish a secure and authenticated channel (e.g., with a TLS certificate) to the mix node to advertise themselves for the next epoch. The authorities gather all mix node advertisements and perform the identity allocation. Afterwards, the mix nodes can re-use the authenticated channel and retrieve their secret keys.

8 Security

In this section, we sketch our security argument and give an intuition behind it. The full formal proof can be found in Section A.

Our argument considers the *cryptographic security* of a single Aimless Onion, meaning that it cannot be tracked by its content. Attacks such as timing analysis, denial of service or equivocation attacks are not part of the formal model, and as such are not covered by our argument. However, we discuss their implications in Section 10.

Our argument is based on the formal model of onion routing by Kuhn et al. [20]. We adapt their model to include the notion of key-generating authorities, and allow for a minority of those authorities to be semi-honest.

Their first property, Tail Indistinguishability (TI), ensures that a malicious receiver cannot distinguish two onions that share the same path after an honest mix node. The property is modelled using a game between a challenger and an adversary. The challenger receives a message and a path from the adversary, and uses them to build an onion. Additionally, it builds a second onion that has a different prefix to the path. The challenger then randomly selects one of those onions, and gives its processed form to the adversary. The adversary must decide which onion has been processed.

We prove that Aimless Onions fulfills TI by making a hybrid argument: In our hybrid, the parts of the header that contain information about the path before the honest mix node are replaced by randomly chosen bits. This hybrid is indistinguishable from the original game, as there, the honest mix node encrypts this part of the header. We then look at the remaining pieces of information that the onion depends on and see that none depend on the challenger's choice of onion.

Kuhn et al.'s second property, Layer Unlinkability (LU), ensures that the adversary cannot recognize an onion after it has passed through an honest mix node. This property is also modelled as a game between a challenger and an adversary. The adversary again provides the challenger with a message and a path, of which the challenger creates an onion. The challenger also creates a stand-in onion that shares only the prefix of the path, but has a random destination and message. The adversary is given the processed form of the first onion, as well as a random choice between the first onion or the stand-in. Again, the adversary must decide which of the two onions it has received.

We prove that Aimless Onions fulfills this property via a hybrid argument. We successively replace values in the onion generation procedure with randomness, such that the final onion consists only of random values. The adversary however cannot notice those replacements as that would imply breaking the HIBE encryption, or alternatively breaking the secret sharing.

9 Evaluation

We evaluate our format in four aspects: First, we evaluate how accurately our identity-node mapping ι represents the mix nodes' desired weight, based on the size $|\mathcal{I}|$ of the namespace. Then we evaluate how the choice of $|\mathcal{I}|$ affects the computation time for the HIBE operations. Afterwards, we fix $|\mathcal{I}|$ and do an in-depth evaluation of the computational effort required by the clients to form onions, the mix nodes to process onions and the authorities to generate keys. Finally, we evaluate the bandwidth overhead that our format induces for each packet, and compare it to the overhead of downloading the list of mix nodes at the start of each epoch.

9.1 Weight representation accuracy

We first investigate how accurately our scheme can represent the different weights of mix nodes, depending on the size of the identity space $|\mathcal{I}|$. We expect that a larger identity space will allow for a more accurate representation of the weights, but will also cause the HIBE to be less efficient. Therefore, we want to find the smallest identity space that gives an acceptable representation accuracy.

To evaluate the representation accuracy, we use Tor's consensus to get an example of a real-life bandwidth distribution, and we assign each relay a weight proportional to its bandwidth. We then run our allocation mechanism (Algorithm 1) for increasing identity space sizes $|\mathcal{I}| = 2^d$, $d \in \{1, 2, \ldots\}$. For each size, we compute the mean absolute percentage error (MAPE) between the resulting weights and the initial weights. A MAPE of 10% for instance means that on average, each relay's assigned weight differed from the initial weight by 10%.

In addition to our allocation algorithm, we also run a modified version that can "skip" the final allocations and leave parts of the identity space unallocated. While this is not useful in practice (onions sent to those identities would be lost), it helps to better understand the scaling behavior of the allocation algorithm by providing a lower bound: This modified version has at most a MAPE of 100%, as the worst case for a relay is to stay unallocated.

From our results (Figure 3, top left) we can see that our expectations are correct: The baseline starts at a high MAPE, meaning that there is a large average deviation from allocated weight to desired weight of a relay. The deviation starts to decrease steeply at around $|\mathcal{I}| = 2^{10}$ and falls below 1% at $|\mathcal{I}| = 2^{27}$.

For the actual allocation algorithm, we see a second effect: It cannot leave identities unallocated, so with a small $|\mathcal{I}|$, a few mix nodes will have vastly overrepresented weights. This leads to a

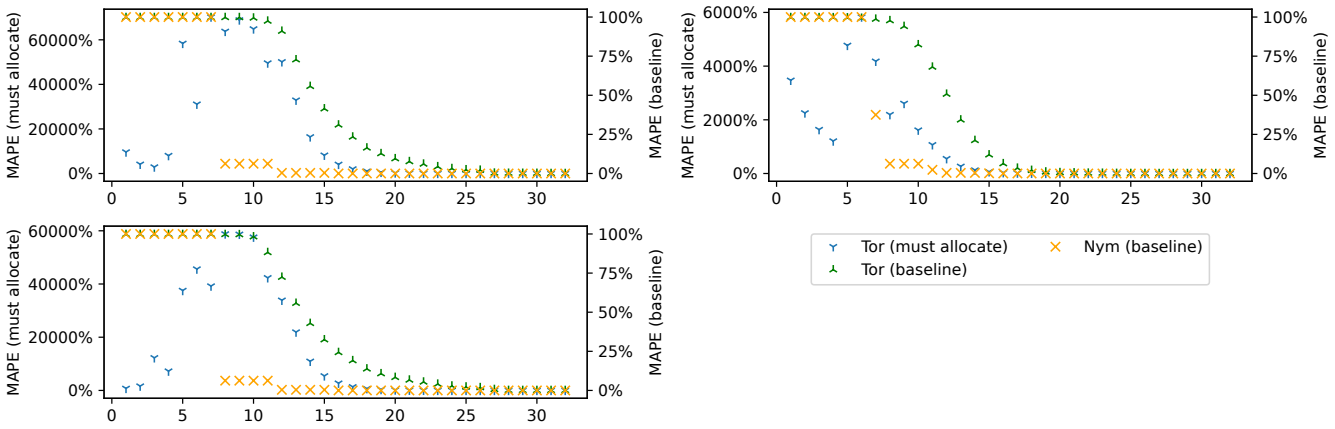

Figure 3: Mean absolute percentage error (MAPE) between allocated weight and actual weight for different identity space sizes $|\mathcal{I}| = 2^x$. The "must allocate" points represent Algorithm 1, while the baseline represents a version of the algorithm that can leave identities unallocated. Top-left represents the variant without buckets, top-right the variant with fine-grained buckets, and bottom-left the variant with coarse buckets.

very high MAPE (9666% at $|\mathcal{I}| = 2^1$). As $|\mathcal{I}|$ grows, so does the number of mix nodes with overallocations, leading first to a *decrease* in representation accuracy. However, once $|\mathcal{I}|$ is large enough to accurately represent the desired weights, the overall deviation also approaches zero in the same way the baseline does.

We run two more variants of this experiment to see the effect of the bucketization on the accuracy: As each bucket contains fewer mix nodes, we expect that fewer bits are required to achieve a good representation accuracy. The bits that we save in weighting bits can then be used as flag bits.

In the first variant, we use flags inspired by Tor: *Fast*, *Guard*, *Stable* and *Exit*, as well as the country (represented as an 8 bit integer) of the node. This gives 12 flag bits for a total of 4096 buckets. In this variant, we can see that the MAPE falls below 10% at 21 bits (Figure 3, top right). However, we can also see that only 368 buckets are non-empty, meaning that a lot of the identity space is wasted on empty buckets.

In the second variant, we coarsen the buckets: We now only distinguish exit and non-exit nodes, and we consider the continent instead of the country. This requires only 4 flag bits and leads to 12 (out of 14) non-empty buckets. In this variant, the MAPE falls below 10% at 30 bits (Figure 3, bottom left).

We repeat all three variants with a Nym-like network with 240 nodes, and find that a smaller identity space is sufficient. For the base variant (i.e. all nodes in a single bucket), we get a MAPE of below 10% at $|\mathcal{I}| = 2^{16}$. For the first variant (fine bucketization), we get a MAPE of below 10% at $|\mathcal{I}| = 2^{13}$, but only 46 buckets are non-empty. For the second variant (coarse bucketization), we get a MAPE of below 10% at $|\mathcal{I}| = 2^{15}$.

9.2 Rust implementation

As the basis for our performance benchmarks, we implemented a prototype of Aimless Onions in Rust. We chose Rust because it is a fast, memory-safe language with a rich ecosystem of available cryptographic libraries. In our prototype, we aim for $\kappa = 128$ bits of security, in line with the current NIST recommendation for symmetric key lengths [3].

We implement the HIBE of Boneh et al. [4], as it leads to small ciphertexts. Further, to avoid embedding our secret shares in $\mathbb{G}_T$, we use the HIBE as hybrid encryption. This gives us a ciphertext in $\mathbb{G}_1 \times \mathbb{G}_2 \times \{0,1\}^l$, where l is the size of the plaintext.

For the underlying curve, we use BLS12-381 as suggested by the IETF draft for pairing-friendly curves.[6] While the security of BLS12-381 is estimated to be only 126 bits [2], it is still adopted in practice due to its faster performance compared to other curves.

For the remaining primitives, we chose the `sha3`, `aes`, `ctr` and `hmac` crates to provide implementations for our symmetric primitives, as well as `bls12_381_plus` for the BLS12-381 curve and `shamirsecretsharing` for the implementation of Shamir's secret sharing.[7]

9.3 HIBE operations

All benchmarks in this section were run on a server with a 2.5 GHz CPU and 2 GiB of RAM.

In this experiment we want to evaluate how the size of the identity space $\mathcal{I}$ affects the performance of the HIBE operations. Together with the accuracy evaluation from Section 9.1, we use this to judge the right size for $\mathcal{I}$.

For this experiment, we use our Rust implementation to measure the time it takes to create an onion (corresponding to the HIBE encryption), to unwrap an onion (decryption), to derive a relay's key (hierarchical key derivation) and to generate a key by the authority (key generation with the master key). We fix all parameters except for the size of the identity space, which we vary from $|\mathcal{I}| = 2^1$ to $|\mathcal{I}| = 2^{64}$.

We expect the onion creation time to scale linearly in the depth of the hierarchy space, as each additional level increases the HIBE key size by one group element. Similarly, the key derivation for a

[6] https://datatracker.ietf.org/doc/draft-irtf-cfrg-pairing-friendly-curves/
[7] All crates can be found at https://crates.io/crates/$name

Format	a	Avg. Time [ms]
Sphinx	—	0.19 (±0.00)
Aimless Onions	1	105.42 (±0.00)
	3	316.32 (±0.03)
	5	527.19 (±0.09)
	7	737.92 (±0.06)
	9	948.70 (±0.05)

Table 2: Benchmark results of the onion creation procedure for Sphinx and Aimless Onions with a varying amount of authorities. The path length is fixed to 3 and the payload size is fixed to 1024 bytes. The number of authorities is denoted as a. Reported value is the mean ± standard error.

Format	a	Avg. Time [ms]
Sphinx	—	0.10 (±0.00)
Aimless Onions	1	2.74 (±0.01)
	3	8.29 (±0.01)
	5	13.65 (±0.01)
	7	19.06 (±0.00)
	9	24.52 (±0.00)

Table 3: Benchmark of the onion processing procedure at a mix node with varying number of authorities. The payload size is fixed to 1024 bytes. The number of authorities is denoted as a. Reported value is the mean ± standard error.

relay and the key generation for the authority each do more work for larger keys. On the other hand, we expect the onion decryption process to be independent of $|\mathcal{I}|$, as the derived HIBE key is only used to key a symmetric cipher.

Our results (shown in Figure 4) confirm our expectations. For all operations dependent on the key size, we see a clear increase in computation time. Doubling the depth leads to roughly a doubling in expected computation time.

With those results, and the results from Section 9.1, we choose to set $|\mathcal{I}| = 2^{32}$ for the following benchmarks. This gives a good weight representation and makes the identities fit into 32-bit integers.

9.4 Computation benchmarks

All benchmarks in this section were run on a ThinkPad E14 Gen 4 with an AMD Ryzen 5 5625U and 16 GB of memory.

We first benchmark the client's needed time to form an onion. For this experiment, we evaluate in three dimensions most likely to vary in different deployments: In one experiment, we vary the payload size, in a second one the path length and in a third one the number of authorities. In all cases, we compare against the implementation of Sphinx from the Nym project.[8]

We expect the payload size to have a negligible impact on the performance, as the complete payload is processed using a fast symmetric cipher. Further, we expect both the path length and the authority count to have a linear impact on the onion creation time, as each of those dimensions adds more expensive HIBE operations. Those HIBE operations form the dominating expensive part of the processing time.

Our results confirm our expectations. We see a linear increase both when increasing the number of authorities (Table 2), as well as when we increase the path length (Figure 5). The payload size had no impact on the onion creation time. In total, creating Aimless Onions onions is around 3 orders of magnitude slower than creating Sphinx onions, but still takes less than 1 second. For the message-based system we target, this does not cause a bottle neck, as users send messages infrequently: For example, Rosenfeld et al. state that the average WhatsApp user only sends 39 messages *per day* [23].

In the second benchmark, we evaluate the performance impact of Aimless Onions on mix nodes that unwrap onions. This consists of two parts: First, the mix node has to derive the correct key for the chosen identity. Then, the mix node has to actually decrypt and unwrap the onion. In both cases, we expect a linear dependency of the needed time and the authority count, as the processes need to be done once per existing authority.

For key derivation with $|\mathcal{I}| = 2^{32}$, we measure a time of 158 ms per key per authority (single threaded). For 9 authorities, this results in a time of 1473 ms per key. This process has to be done once for each identity, but the key can be derived in advance and re-used for all onions sent to the same identity. In the case of $|\mathcal{I}| = 2^{20}$, this time is reduced to 72 ms (per key per authority), leading to 648 ms for 9 authorities.

For onion unwrapping, our results (Table 3) also confirm a linear relationship between unwrapping time and authority count. We can see that Aimless Onions processing is around 450 times slower than Sphinx, but can still complete in less than 30 milliseconds per onion even for a setting with 9 authorities, giving a throughput of around 33 onions per second (single threaded). These numbers are again independent of the payload size, as the payload is processed with a symmetric cipher, which takes only negligible time. As a result, we do not consider the throughput of onion decryption to be the limiting factor in our design: Assuming 39 messages per day per user [23] and pre-derived keys, each mix node has enough throughput to handle the onions of over 70000 users per thread.

Finally, we determine how long an authority needs to generate all required keys for the mix nodes. We do so to assess whether the authority can keep up with the key generation for all nodes. For that, we have taken the 2023-11-08 14:00 Tor consensus containing 8584 relays and their advertised bandwidth as sensible real-world data. We then allocate those relays according to our algorithm, and measure the time it takes to generate the secret keys for all relays.

Our experiment shows that using a single core, the authority needs about 16 minutes to generate all secret keys. This process can be sped up linearly, such that it takes only $16/n$ minutes using n CPU cores. With epoch lengths of one hour, as used for example in Tor, the authorities in Aimless Onions are fast enough to generate all necessary keys during an epoch before the next one starts.

9.5 Bandwidth baseline

Before we evaluate the bandwidth usage of Aimless Onions, we first establish a baseline to compare against. To do so, we look at

[8] https://github.com/nymtech/sphinx

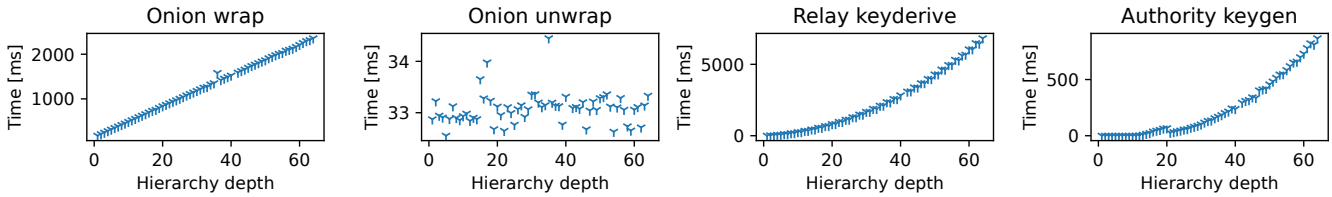

Figure 4: Computation time of the various steps for different identity space sizes $|\mathcal{I}| = 2^d$. We omit the standard error as it is too small to be seen in the plots.

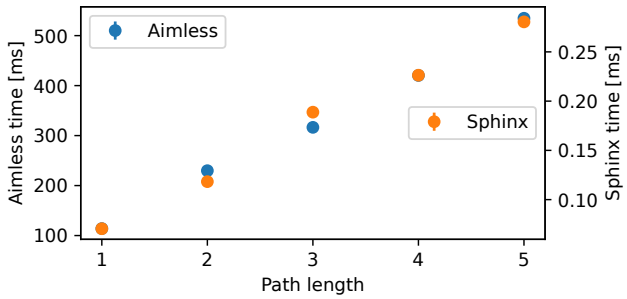

Figure 5: Benchmark of the onion creation procedure for a varying path length, fixed authority count of 3, fixed payload size of 1024 bytes and a hierarchy size of $|\mathcal{I}| = 2^{32}$.

Tor[9] and Nym as two existing and deployed anonymous communication networks, and we measure the bandwidth they use for bootstrapping the network topology.

For Tor, the list of relays is published in the *consensus* document. We can obtain recent consensuses from the CollecTor service[10] to find their size at around 3 MiB. However, the consensus can be compressed during transmission, so this does not reflect the real bandwidth usage.

To find out the actual amount of transmitted data, we use the Tor client and its diagnostic output. We start a fresh instance with no cached network information, wait for the bootstrapping to complete, and then look at the reported values in the client's log. In addition, we capture the bootstrapping process using `tcpdump` to verify the numbers that the Tor client measures.

With this technique, we can see that the Tor client downloads about 632 KiB for the consensus, 14 KiB for the authorities' certificates, and 11 MiB for the relays' microdescriptors. These values are averaged over 16 bootstrapping runs, and the `tcpdump` capture verifies those numbers. As we care about the consensus only, we will use the 632 KiB as a baseline for Tor, representing approximately 7400 relays.

In addition to a full consensus, Tor allows for clients to download consensus diffs. These diffs only contain changes between an old consensus and the current one, which reduces the amount of data transmitted for clients that have access to an old consensus document. We include a diff-based baseline to represent clients that stay connected to the network for longer.

To estimate the size of a consensus diff, we use Arti[11] to connect to a Tor relay, and download diffs for the last ten consensuses. From this, we see diff sizes ranging from 88 KiB (for the newest consensus diff) to 713 KiB (for the oldest consensus diff). We therefore use 415 KiB as the baseline for the diff size, which corresponds to the average over the last ten diffs.

For Nym, we examine their client to find the API endpoints where the network topology is downloaded from.[12,13] Those documents are 197 KiB and 49 KiB in size. As the Nym client does not use compression, we use the resulting sum of 246 KiB as the baseline for Nym, representing 240 mix nodes and 134 gateways. Again, we use `tcpdump` to verify those numbers.[14]

9.6 Bandwidth overhead

In our second experiment, we want to evaluate the overhead in onion size and therefore bandwidth usage when switching from Sphinx to Aimless Onions. In particular, we want to find the break-even point at which the Sphinx setup (loading a large initial consensus and sending small onions) becomes more efficient than our approach (loading a small consensus and sending large onions).

As Aimless Onions embeds more key material into the header, and relies on cryptographic primitives with large ciphertext expansion, we expect the onions to be larger than Sphinx onions. However, this overhead depends only on the path length and authority count, as the payload itself is encrypted using symmetric encryption and therefore does not contribute to the additional overhead of Aimless Onions compared to Sphinx.

We expect to see a linear dependency between the Aimless Onions onion size and the number of authorities, as each authority leads to an additional key share that needs to be embedded at each hop. Similarly, we expect to see a linear dependency between the onion size and the path length, as each hop leads to a fixed extra amount of information that is added to the header. In any case, we expect Aimless Onions to produce bigger packets than Sphinx, but this overhead will be offset by the fact that no list of mix node identities and keys (a "consensus") must be downloaded.

To get the values, we use the Sphinx and Aimless Onions implementations to output the actual onion size for various system

[9] https://torproject.org
[10] https://metrics.torproject.org/collector.html
[11] https://crates.io/crates/arti-client
[12] https://validator.nymtech.net/api/v1/mixnodes/active
[13] https://validator.nymtech.net/api/v1/gateways
[14] The client actually downloads the documents twice, but we only count them once.

parameters. In three different experiments, we first vary the authority count while keeping the path length and payload size fixed, then we vary the path length and keep the other parameters fixed, and finally we vary the payload size.

From our first experiment we see that an increasing number of authorities increases the Aimless Onions size linearly, while not affecting the size of Sphinx onions. The second experiment shows that the size of both Sphinx and Aimless Onions onions increases linearly with the path length. However, the per-hop increase is larger with Aimless Onions than with Sphinx. Finally, in the last experiment we confirm that both Sphinx and Aimless Onions scale linearly with the payload size with the same factor of 1.

While individually, Sphinx onions are always smaller than Aimless Onions, we see that we can save data when we consider the total bandwidth that a client spends, including the initial consensus download. This is shown in Figure 6 for different combinations of parameters. For Tor-like parameters with 9 authorities and a path length of 3, we see that clients sending less than 105 onions per hour benefit from Aimless Onions. When using consensus diffs, this number drops to 69 onions per hour. For Nym, the break-even point is at 27 onions, as Nym is a smaller network with less topology data, and uses a path length of 5. With the average instant messaging user sending around 39 messages *per day* [23], Aimless Onions is beneficial for the vast majority of messaging users.

10 Discussion

So far, we have outlined the base version of Aimless Onions. In this section, we want to outline some shortcomings of this design and how they can be improved.

Mix node double choosing. Tor ensures that a path does not contain the same relay (or two relays run by the same entity) twice. With Aimless Onions, we cannot guarantee this: even if the client picks different identities, it does not know whether those map to the same mix node.

For networks with many mix nodes, the chance of this happening is small. For a Tor-sized network with around 8600 nodes, Tor's bandwidth distribution and a path length of 3, the chance to double-pick a node is 0.10%. For a Nym-sized network with around 240 equally-probable nodes and a path length of 5, this chance increases to 4.13%. By using a stratified topology and separating the layers into different buckets, this possibility can be eliminated completely.

Balancing mix node and authority work. In our design, we describe a setup in which we interpret identities as binary numbers. This allows the authorities to make optimal use of the hierarchical derivation feature of the HIBE structure to combine many keys, but it leads to a tall hierarchy and a potentially large amount of keys that a mix node has to derive. By choosing a different basis for the identity space, we can balance this work better between the authorities and the relays. For example, if we use quaternary numbers as the basis for our identity space, we can keep the total number of identities $|\mathcal{I}|$ the same while halving the depth of the hierarchy. This increases the work for the authorities to generate the keys, but decreases the work for the relays to locally derive their keys.

Faulty and malicious authorities. In our design, authorities are responsible for generating the keys that mix nodes use for their operation. As a result, a faulty authority may not provide those keys in time. Due to the threshold secret sharing, our design can tolerate up to $a - k$ faulty authorities while staying operational.

Further, malicious authorities may partition clients by providing them with different public keys, thereby making the secret shares they encrypt recognizable. The key distribution method we describe in Section 7.7 cannot prevent such an attack. To solve this problem, a stronger agreement protocol (like Tor's voting mechanism) needs to be implemented between the authorities, and all keys must be signed by each authority. The design of such a Byzantine agreement protocol is out of scope for this paper.

Supporting replies. Assuming that all onions should look the same, hop-by-hop integrity is at odds with replies, as a sender cannot precompute the MACs for the reply onion [21]. Our design of Aimless Onions offers strong hop-by-hop integrity at the cost of not supporting anonymous replies via single-use-reply-blocks (SURBs). However, we can relax the hop-by-hop integrity guarantees and support SURBs in a Sphinx-style setup. We detail the changes necessary for that in Section B.

Denial of service attacks. While all anonymous communication networks are susceptible to *denial of service* (DoS) attacks to some extent, the potential for DoS attacks in Aimless Onions is exacerbated because the relays spend much longer to unwrap a packet than the sender needs to wrap it. We expect honest nodes to discard duplicate packets, and so the scope of such an attack is limited, as clients still have to work to build fresh packets. Like related work, we consider other protective measures to be out of scope for this paper, but point to Tor's proof-of-work or Nym's credential system as possible solutions.

Threshold HIBE. The requirements we identify in our design make the use of a threshold HIBE (T-HIBE) [14] seem natural. However, we note that the size of the ciphertext in T-HIBE scales with the depth of the hierarchy. As we use tall hierarchies, the T-HIBE overhead would be very large. We therefore use a traditional HIBE with a ciphertext whose size is independent of the hierarchy depth, and combine it with threshold secret sharing to obtain functionality similar to that of a T-HIBE.

11 Conclusion

Mix networks offer great scalability, as additional nodes can be added to support new clients. However, current designs require clients to maintain a consistent, up-to-date view of the global network topology in order to prevent epistemic attacks [12]. In practice, the distribution of this topology information, which grows linearly with the number of nodes in the network, limits scalability.

In this paper, we propose a novel mix packet format, *Aimless Onions*, which relieves clients of the need to download topology information without compromising anonymity guarantees. Instead, each client only needs to download constant-size public parameters, independently of the number of mix nodes.

We provide formal proof of Aimless Onions's security guarantees and evaluate its performance with realistic parameters from

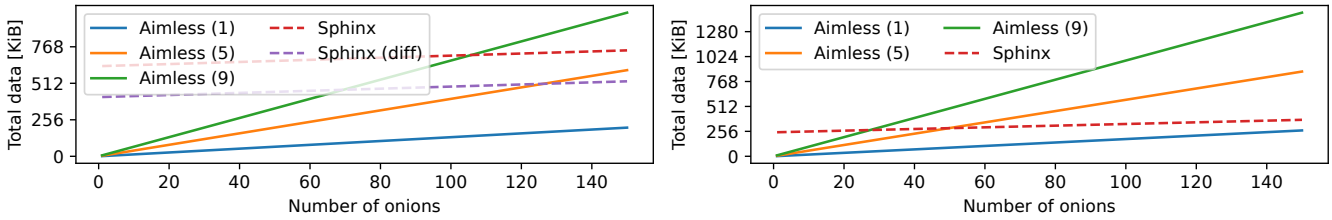

Figure 6: Total amount of data transmitted for sending n onions (including the consensus download). The Aimless parameters gives the number of authorities. The intersection between the lines is the break-even point. The left graph represents a Tor-like scenario with Tor's consensus size and 3 hops, while the right side uses a Nym-like scenario with Nym's consensus size and 5 hops. The left graph additionally contains a scenario in which the size of a Tor consensus diff is used.

the Tor and Nym networks. We find that individual packets in Aimless Onions are larger than packets in currently used formats, but in scenarios in which users send only a few messages, Aimless Onions saves bandwidth compared to downloading a consensus up front even at today's network sizes. We also find that our protocol requires more computational effort than existing protocols, but is still feasible to use in practice.

While we acknowledge the downsides of Aimless Onions in a deployed system, we see it as a step towards building mix networks that can scale to support mainstream adoption.

Acknowledgments

We thank the anonymous reviewers for their feedback. This work was funded by the Topic Engineering Secure Systems of the Helmholtz Association (HGF) and supported by KASTEL Security Research Labs, Karlsruhe. Funded by the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) as part of Germany's Excellence Strategy – EXC 2050/1 – Project ID 390696704 – Cluster of Excellence "Centre for Tactile Internet with Human-in-the-Loop" (CeTI) of Technische Universität Dresden.

Availability

We have made our prototype implementation and the evaluation tools publicly available at https://github.com/kit-ps/aimless-onions (tag `pets-2025`).

References

[1] Ross Anderson and Eli Biham. 1996. Two Practical and Provably Secure Block Ciphers: BEAR and LION. In *Fast Software Encryption*.
[2] Diego F. Aranha, Youssef El Housni, and Aurore Guillevic. 2023. A Survey of Elliptic Curves for Proof Systems. In *Des. Codes Cryptogr.*
[3] Elaine Barker. 2020. *Recommendation for Key Management: Part 1 - General.* Technical Report. National Institute of Standards and Technology.
[4] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. 2005. Hierarchical Identity Based Encryption with Constant Size Ciphertext. In *EUROCRYPT*.
[5] Dan Boneh and Matt Franklin. 2001. Identity-Based Encryption from the Weil Pairing. In *CRYPTO*.
[6] Dan Boneh and Victor Shoup. 2023. *A Graduate Course in Applied Cryptography.* Self-published.
[7] Dario Catalano, Mario Di Raimondo, Dario Fiore, Rosario Gennaro, and Orazio Puglisi. 2013. Fully Non-Interactive Onion Routing with Forward Secrecy. In *Int. J. Inf. Secur.*
[8] Dario Catalano, Dario Fiore, and Rosario Gennaro. 2017. A Certificateless Approach to Onion Routing. In *Int. J. Inf. Secur.*
[9] David L. Chaum. 1981. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. In *Commun. ACM.*
[10] G. Danezis, R. Dingledine, and N. Mathewson. 2003. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *IEEE S&P.*
[11] George Danezis and Ian Goldberg. 2009. Sphinx: A Compact and Provably Secure Mix Format. In *IEEE S&P.*
[12] George Danezis and Paul Syverson. 2008. Bridging and Fingerprinting: Epistemic Attacks on Route Selection. In *PETS.*
[13] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. *Tor: The Second-Generation Onion Router.* Technical Report. Naval Research Lab Washington DC.
[14] Sayon Duttagupta, Dave Singelee, and Bart Preneel. 2022. T-HIBE: A Novel Key Establishment Solution for Decentralized, Multi-Tenant IoT Systems. In *IEEE CCNC.*
[15] Rob Jansen and Robert Beverly. 2010. Toward Anonymity in Delay Tolerant Networks: Threshold Pivot Scheme. In *IEEE MILCOM.*
[16] Aniket Kate, Greg Zaverucha, and Ian Goldberg. 2007. Pairing-Based Onion Routing. In *PETS.*
[17] Jonathan Katz and Yehuda Lindell. 2007. *Introduction to Modern Cryptography.* CRC Press.
[18] Kenneth C. Knowlton. 1965. A Fast Storage Allocator. In *Commun. ACM.*
[19] Chelsea H. Komlo, Nick Mathewson, and Ian Goldberg. 2020. Walking Onions: Scaling Anonymity Networks While Protecting Users. In *USENIX Security.*
[20] Christiane Kuhn, Martin Beck, and Thorsten Strufe. 2020. Breaking and (Partially) Fixing Provably Secure Onion Routing. In *IEEE S&P.*
[21] Christiane Kuhn, Dennis Hofheinz, Andy Rupp, and Thorsten Strufe. 2021. Onion Routing with Replies. In *ASIACRYPT.*
[22] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. 2017. The Loopix Anonymity System. In *USENIX Security.*
[23] Avi Rosenfeld, Sigal Sina, David Sarne, Or Avidov, and Sarit Kraus. 2018. WhatsApp usage patterns and prediction of demographic characteristics without access to message content. In *Demographic Research.*
[24] Philip Scherer, Christiane Weis, and Thorsten Strufe. 2024. A Framework for Provably Secure Onion Routing against a Global Adversary. In *PETS.*
[25] Adi Shamir. 1979. How to Share a Secret. In *Commun. ACM.*
[26] Richard Wang, Dana Butnariu, and Jennifer Rexford. 2011. OpenFlow-based Server Load Balancing Gone Wild. In *USENIX Hot-ICE.*

A Security

In this section, we prove the security of Aimless Onions. We do so in two steps: First, we focus on our use of HIBE and secret sharing to show that the onion key cannot be recovered by the adversary. This step justifies our trust assumption, as we show that corrupting authorities gives the adversary no advantage, as long as no majority of authorities has been corrupted. We can therefore use the *anytrust* model, which is commonly used in mix networks.

Then, we consider Aimless Onions's security in Kuhn et al.'s formal model of onion routing [20] by looking at the two properties they define: *Tail-Indistinguishability* (TI) and *Layer-Unlinkability* (LU).

A.1 HIBE and secret sharing

Before we go on to prove the security of our packet format as a whole, we first consider our combination of HIBE and secret sharing in our threat model to show that the adversary has no advantage if less than the majority of authorities are corrupted.

In general it cannot be assumed that the composition of multiple cryptographic primitives maintains the security of the overall construction. To simplify our security proof we first show that our combination of HIBE and secret sharing is secure via a custom security notion SH-IND.

We define our notion based on an indistinguishability game which asks the adversary to differentiate between two messages. Those messages are secret shared and then HIBE encrypted, just like in Aimless Onions. We then show that we can reduce this new property to the security properties of HIBE and secret sharing.

Definition 5 (SH-IND). *We define the (k, a) shared-HIBE-indistinguishability (SH-IND) game as follows:*

1. *The challenger picks a bit $b \leftarrow^{R} \{0,1\}$ randomly.*
2. *The challenger generates a HIBE instances using* `Setup`.
3. *The adversary submits an identity I and two messages m_0 and m_1 of same length to the challenger.*
4. *The challenger splits m_b into a secret shares $\sigma_1, \ldots, \sigma_a$.*
5. *The challenger encrypts each share σ_i with the HIBE and identity I: $\Sigma_i = \texttt{Encrypt}(I, \sigma_i)$.*
6. *The challenger gives the encrypted shares $\Sigma_1, \ldots, \Sigma_a$ to the adversary.*
7. *The adversary can query the oracle O with an index $i \in \{1, \ldots, a\}$ to retrieve the i^{th} HIBE secret key. The adversary may do this up to $k-1$ times.*
8. *The adversary produces a guess b' and wins if $b' = b$.*

We say that a scheme is SH-IND secure iff no PPT adversary has a non-negligible advantage in κ to win the HS-IND game.

Lemma 1. *Given a IND-ID-CPA secure HIBE and a secure secret sharing algorithm, our combination of HIBE and secret sharing from Section 6 is SH-IND secure.*

Proof. First, we note that the adversary can only obtain $k-1$ HIBE secrets at maximum. Without loss of generality, we assume that the adversary obtains the first $k-1$ secrets: If the adversary does not obtain the first HIBE secrets, we can re-order the encrypted shares accordingly, as each share contains an equal amount of information. This is a result of secret sharing security (Definition 4) when applied to subsets consisting of a single element.

We now define a hybrid game H_1, where in step (6) the challenger does not give all shares to the adversary. Instead, the challenger encrypts a random value $R \leftarrow^{R} \{0,1\}^{|\sigma_i|}$ using the HIBE, and then gives the adversary $a-1$ "real" shares and 1 "random" encryption.

The adversary cannot distinguish this hybrid from the actual game, as distinguishing would break the IND-ID-CPA of the underlying HIBE:

Assume that there is a distinguisher $\mathfrak{D}$ that can differentiate between H_1 and the SH-IND game. We then construct an adversary $\mathfrak{H}$ that wins the IND-ID-CPA game:

1. The IND-ID-CPA challenger C sets up the IND-ID-CPA challenge and gives $\mathfrak{H}$ the system parameters.
2. $\mathfrak{H}$ generates $a-1$ HIBE instances using `Setup`.
3. $\mathfrak{H}$ chooses a message $m \leftarrow^{R} \mathcal{D}$ and an identity $I \leftarrow^{R} \mathcal{I}$.
4. $\mathfrak{H}$ splits m into a secret shares $\sigma_1, \ldots, \sigma_a$, as well as a "fake share" $\tilde{\sigma} \leftarrow^{R} \mathcal{S}$.
5. $\mathfrak{H}$ uses the $a-1$ HIBE instances to produce encryptions of the first $a-1$ shares: $\Sigma_i = \texttt{Encrypt}(I, \sigma_i)$
6. $\mathfrak{H}$ provides C with the ID I and the messages $m_0 = \sigma_a, m_1 = \tilde{\sigma}$ as challenge, and receives the challenge ciphertext Σ_n.
7. $\mathfrak{H}$ now provides $\mathfrak{D}$ with $\Sigma_1, \ldots, \Sigma_a$.
8. On queries to the oracle O from $\mathfrak{D}$, $\mathfrak{H}$ returns the corresponding HIBE secret key. We note that we assume corruption to only appear in the first $k-1$ keys, therefore $\mathfrak{D}$ can not ask for the key of the last instance, which is only known to C.
9. If $\mathfrak{D}$ returns "real game", $\mathfrak{H}$ returns $b = 0$ to C. Otherwise, $\mathfrak{H}$ returns $b = 1$.

We can see that $\mathfrak{H}$ interpolates between the SH-IND game and H_1 for $\mathcal{D}$, and the winning advantage of $\mathfrak{H}$ in the IND-ID-CPA game is equal to the distinguishing advantage of $\mathfrak{D}$.

We can repeat this process $a-k$ times, such that the final $a-k+1$ shares only contain random values. As each hybrid is indistinguishable to its predecessor, the final hybrid is also indistinguishable to the original game.

In this final hybrid however, the adversary has at maximum access to $k-1$ secret shares. Due to the secret sharing security (Definition 4), any value $x \in \mathcal{D}$ is still as likely to be the message, as the distributions of less than k shares are indistinguishable. □

Corollary 1. *Given a random secret $s \leftarrow^{R} \{0,1\}^{\kappa}$ which is first secret shared and then each share HIBE encrypted, no PPT adversary has a non-negligible advantage of recovering a bit of s given at maximum $k-1$ shares.*

A.2 Tail-indistinguishability

The notion of Tail-indistinguishability (TI) ensures that the onion format protects against malicious receivers. The definition is game based and asks an adversary to distinguish between two onions whose paths have different prefixes, but continue along the same suffix after passing through an honest mix node. We recall the original definition of TI by Kuhn et al.:

Definition 6 (Tail-indistinguishability [20]). *TI is defined by the following game:*

1. *The adversary receives the challenge public key and the router identity r_j.*
2. *The adversary may submit any number of onions O_i of her choice to the challenger. The challenger sends the output of* $\text{ProcOnion}(SK, O_i, r_j)$ *to the adversary.*
3. *The adversary submits a message m, a path $P = (r_1, \ldots, r_j, \ldots, r_v)$ with honest node at position j, as well as key pairs for all nodes r_i with $i \neq j$.*
4. *The challenger checks that the path is valid, that the public keys correspond to the secret keys. If the challenge is valid, it sets bit b at random.*
5. *The challenger creates the onion with the adversary's input choice:*

$$(O_1, \ldots, O_{v+1}) \leftarrow \text{FormOnion}(m, P, (PK)_P).$$

It also creates a random onion with a randomly chosen path $\bar{P} = (\bar{r}_1, \ldots, \bar{r}_k = r_j, \ldots, \bar{r}_{\bar{v}+1} = r_{v+1})$ *that includes the subpath from the honest relay to the corrupted receiver:*

$$(\bar{O}_1, \ldots, \bar{O}_{v+1}) \leftarrow \text{FormOnion}(m, \bar{P}, (PK)_{\bar{P}})$$

(6) *If* $b = 0$*, the challenger gives* (O_{j+1}, r_{j+1}) *to the adversary. Otherwise, the challenger gives* $(\bar{O}_{k+1}, \bar{r}_{k+1})$ *to the adversary.*
(7) *The adversary may submit any number of onions* O_i *of her choice to the challenger. The challenger sends the output of* $\text{ProcOnion}(SK, O_i, r_j)$ *to the adversary.*
(8) *The adversary produces guess* b'*.*

TI is achieved if any PPT adversary has only a negligible advantage in guessing $b' = b$ *correctly.*

We adapt the original definition slightly to reflect that the adversary can corrupt up to $k-1$ authorities, and that the authorities are responsible for generating the mix nodes' secret keys. Further, we write $[r_i]$ to mean all identities that map to r_i. We highlight those adaptions in blue in the definition below.

Definition 7 (Aimless-TI). *The Aimless-TI game is defined as follows:*

(1) *The challenger simulates a authorities by setting up a HIBE instances.*
(2) *The adversary receives the challenge router identity* r_j*.*
(3) *The adversary may submit any number of onions* O_i *of her choice to the challenger. The challenger sends the output of* $\text{ProcOnion}(SK, O_i, r_j)$ *to the adversary.*
The adversary may also inquire the secret keys of any relay r_i*,* $[r_i] \neq [r_j]$*, as well as* $k-1$ *secret keys of relay* r_j*.*
(4) *The adversary submits a message* m*, a path* $P = (r_1, \ldots, r_j, \ldots, r_v)$ *with honest node at position* j*.*
(5) *The challenger checks that the path is valid. If the challenge is valid, it sets bit* b *at random.*
(6) *The challenger creates the onion with the adversary's input choice:*

$$(O_1, \ldots, O_{v+1}) \leftarrow \text{FormOnion}(m, P, (PK)_P).$$

It also creates a random onion with a randomly chosen path $\bar{P} = (\bar{r}_1, \ldots, \bar{r}_k = r_j, \ldots, \bar{r}_{\bar{v}+1} = r_{v+1})$ *that includes the subpath from the honest relay to the corrupted receiver:*

$$(\bar{O}_1, \ldots, \bar{O}_{v+1}) \leftarrow \text{FormOnion}(m, \bar{P}, (PK)_{\bar{P}})$$

(7) *If* $b = 0$*, the challenger gives* (O_{j+1}, r_{j+1}) *to the adversary. Otherwise, the challenger gives* $(\bar{O}_{k+1}, \bar{r}_{k+1})$ *to the adversary.*
(8) *The adversary may submit any number of onions* O_i *of her choice to the challenger. The challenger sends the output of* $\text{ProcOnion}(SK, O_i, r_j)$ *to the adversary.*
(9) *The adversary produces guess* b'*.*

Theorem 1. *Aimless Onions fulfills Aimless-TI.*

Intuitively, Aimless-TI requires onions that traverse the same path after an honest node to be indistinguishable, no matter which path they traversed before.

Proof. The challenger provides the adversary with the onion *after* the honest mix node. Therefore, we must assume that all mix nodes following this honest node are malicious.

We now do a hybrid argument: For our first hybrid H_1, we modify the Aimless-TI game such that FormOnion at the honest mix node replaces parts of the header with randomness. In particular, given the honest node r_j, we replace η_j with

$$R \xleftarrow{R} \{0,1\}^{|H|-(v-j)\cdot|H_{\text{hop}}|}$$
$$\eta'_j = (r_{i+1} \,||\, H_{i+1})_{[0..(v-j)\cdot|H_{\text{hop}}|]} \,||\, R$$

Intuitively, we have FormOnion replace the suffix of the header which contains the encrypted padding from the previous hops with randomly chosen bits. We keep the prefix of the header, as that part will be used by the following mix nodes to further process the onion.

For the adversary, this hybrid is indistinguishable to the original game. This is a result of the PRNG security: In the original game, the PRNG is used as a one-time-pad to decrypt the header. In H_1, those values are replaced by truly random bits. Distinguishing those cases implies that the PRNG output can be distinguished from randomness, which contradicts our PRNG definition.

In H_1, we now look at the various elements that the adversary will see, and whether they contain information about b:

- H_v is a random bit string, chosen independently of b.
- k_i are random keys, chosen independently of b.
- σ_i only depend on k_i, and therefore not on b.
- Σ_i only depend on σ_i and the mix node identities which are the same in both scenarios. Σ_i therefore is also independent of b.
- Π_v contains the final recipient and the message, both of which are the same in either scenario and therefore independent of b.
- H_i only contains the information discussed above and random bits, independent of b.

In conclusion, the onion does not contain any information that depends on b and could be used by the adversary to determine the correct scenario. □

A.3 Layer-unlinkability

The notion of LU ensures that an onion cannot be re-identified after passing through an honest mix node. The definition is game based and asks an adversary to identify which of two onions has been processed at an honest mix node. We recall the original definition of LU by Kuhn et al. as well:

Definition 8 (Layer-unlinkability [20]). *LU is defined by the following game:*

(1) – (4) *as in Definition 6*
(5) *The challenger creates the onion with the adversary's input choice:*

$$(O_1, \ldots, O_{v+1}) \leftarrow \text{FormOnion}(m, P, (PK)_P)$$

and a random onion with a a randomly chosen path $\bar{P} = (\bar{r}_1, \ldots, \bar{r}_k = r_1, \ldots \bar{r}_{k+j} = r_j, \bar{r}_{k+j+1}, \ldots, \bar{p}_{\bar{v}+1})$*, that includes the subpath from the honest sender to the honest node of* P *starting at position* k *ending at* $k+j$*, and a random message* m'*:*

$$(\bar{O}_1, \ldots, \bar{O}_{\bar{v}+1}) \leftarrow \text{FormOnion}(m', \bar{P}, (PK)_{\bar{P}})$$

(6) *If $b = 0$, the challenger gives $(O_1, \text{ProcOnion}(O_j))$ to the adversary. Otherwise, the challenger gives $(\bar{O}_1, \text{ProcOnion}(O_j))$ to the adversary.*
(7) *The adversary may submit any number of onions O_i, $O_i \neq O_j$, $O_i \neq \bar{O}_{k+j}$ of her choice to the challenger. The challenger sends the output of $\text{ProcOnion}(SK, O_i, r_j)$ to the adversary.*
(8) *The adversary produces guess b'.*

LU is achieved if any PPT adversary has only a negligible advantage in guessing $b' = b$ correctly.

We again modify the original LU game to accomodate the changes in the system model that Aimless Onions assumes:

Definition 9 (Aimless-LU). *The Aimless-LU game is defined as follows:*

(1) – (5) *as in Definition 7*
(6) – (9) *as (5) – (8) in Definition 8*

Theorem 2. *Aimless Onions fulfills Aimless-LU.*

Proof. We use a hybrid argument to prove Theorem 2.

In the first hybrid H_1, we modify the behavior of FormOnion and ProcOnion at the honest node. In particular, when FormOnion creates the layer for node r_j, it does not embed the master key k_j in the header, but rather draws a random identifier $I_j \leftarrow^{\text{R}} \{0,1\}^{\kappa}$ and saves the tuple (I_j, k_j). When ProcOnion is called to process the onion at the honest mix node, it looks up the identifier I_j in the header to retrieve the saved key k_j.

For the adversary, this hybrid is indistinguishable to the Aimless-LU game as per Corollary 1.

In the second hybrid H_2, we modify ProcOnion such that r_j will reject an onion when I_j is saved, but the onion was not output by FormOnion. This means that the adversary can not modify the challenge onion to provide it to the oracle.

For the adversary, this hybrid H_2 is indistinguishable from H_1: As the onion header and content are protected via a MAC, forging an onion that has the correct I_j would imply a forged MAC tag, which the adversary can only find with negligible probability. From this we conclude that the adversary also cannot modify the challenge onion in H_1, preventing tagging attacks.

In our final hybrid H_3, we change FormOnion to use true randomness instead of ρ when encrypting the header and payload. We have FormOnion save the used randomness indexed by I_j, and modify ProcOnion to use this same randomness to unwrap the onion again.

For the adversary, this hybrid H_3 is indistinguishable from H_2: The only difference between H_3 and H_2 is the use of real randomness instead of pseudorandomness. Per definition of a PRNG, the adversary cannot distinguish the output of a PRNG (with unknown key) and a real random generator.

In H_3 however, the randomness constitutes a one-time-pad encryption of the header and the payload. As such, no information about the contained values can leak to the adversary. From the indistinguishability of H_3 and H_2, H_2 and H_1, and H_1 and the original game, we conclude that Theorem 2 holds. □

A.4 Other attacks

In this section, we have proven the *cryptographic security* of Aimless Onions. We consider other attacks on onion routing and mix networks (such as traffic analysis, drop attacks, denial of service, etc.) as out-of-scope for this paper, as we can employ other state-of-the-art defenses (such as mixing delays and message loops from Loopix [22]). Those attacks and defenses however are independent of the underlying packet format and can be implemented on top of Aimless Onions.

B Supporting replies

Sphinx supports replies by allowing the sender to precompute a Sphinx header (a SURB). A recipient can take this SURB, attach their reply to it, and send it through the mix network, without learning who the final recipient of their reply will be.

We can support SURBs in Aimless Onions by adjusting the format:

- We add a field in the header for the final mix node on the path. For forward messages, this field contains a designated sentinel value. For replies, this field contains the destination.
- We change the MAC in the header to no longer include the payload, but only the header.
- We change the format such that a wide-block cipher like Lioness [1] or AEZ[15] is used for the payload. Such a cipher ensures that a small change in the ciphertext will lead to unpredictable changes in the plaintext.
- When Alice sends her message to Bob, she encrypts the payload $(0^{\kappa} \,||\, \text{Bob} \,||\, m)$.
- When the final mix node receives Alice's message, it finds the sentinel value in the header. It then checks if the prefix of the payload is 0^{κ}. If not, the message has been tagged and must be discarded. Otherwise, it forwards m to Bob.
- When Bob replies, he appends his message to the precomputed header and sends it to the first mix node along the reply path.
- The final mix node on the reply path will find Alice's address in the header field, and can thus forward the message to Alice.

We note that this variant has no hop-by-hop integrity, as a modified payload is only recognized at the final mix node (on the forward path), or the recipient (on the reply path). The security proof that we give in Section A does not hold for this variant, and must be adapted to rely on properties that do not assume hop-by-hop integrity [24].

[15] https://www.cs.ucdavis.edu/~rogaway/aez/