Divide and Funnel: A Scaling Technique for Mix-Networks

Debajyoti Das
KU Leuven, Belgium
debajyoti.das@esat.kuleuven.be

Sebastian Meiser
Universität zu Lübeck, Germany
sebastian.meiser@uni-luebeck.de

Esfandiar Mohammadi
Universität zu Lübeck, Germany
esfandiar.mohammadi@uni-luebeck.de

Aniket Kate
Purdue University / Supra Research, USA
aniket@purdue.edu

***Abstract*—While many anonymous communication (AC) protocols have been proposed to provide anonymity over the internet, scaling to a large number of users while remaining provably secure is challenging. We tackle this challenge by proposing a new scaling technique to improve the scalability/anonymity of AC protocols that distributes the computational load over many nodes without completely disconnecting the paths different messages take through the network. We demonstrate that our scaling technique is useful and practical through a core sample anonymous broadcast protocol, *Streams*, that offers provable security guarantees and scales for a million messages. The scaling technique ensures that each node in the system does the computation-heavy public key operation only for a tiny fraction of the total messages routed through the Streams network while maximizing the mixing/shuffling in every round. Our experimental results show that Streams can scale well even if the system has a load of one million messages at any point in time, with a latency of 16 seconds while offering provable "one-in-a-billion" unlinkability, and can be leveraged for applications such as anonymous microblogging and network-level anonymity for blockchains. We also illustrate by examples that our scaling technique can be useful to other AC protocols to improve their scalability and privacy, and can be interesting to protocol developers.**

1. Introduction

Anonymous communication (AC) protocols [1]–[11] have been on an eternal quest to be scalable in terms of computation and communication overhead while providing strong anonymity properties. AC protocols need to ensure that messages cannot be traced from source to sink, i.e., from sender to recipient. A mixing network or mixnet confuses curious observers and compromised protocol parties by routing each message through multiple parties and "mixing" it with other messages along the way. This mixing property naturally occurs in mixnets with network topologies where messages meet in an honest protocol node or mixnode.

However, a mixnode has to perform public-key cryptographic processing associated with layered encryption, and the scalability of those mixnets is bottlenecked by the computation capacity of the weakest node. Mixnets that attempt to distribute the processing load over several parties in order to (horizontally) scale for a large number of messages, naturally cause a network topology where not all messages meet each other easily. Such protocols compensate for the shortcoming in the network topology in two ways: either they route a message through more nodes, leading to high latency; or they send noise messages between nodes to create the appearance that two messages could have met, leading to high communication overhead. All existing mixnets in the literature that attempt to achieve scalability by distributing computation over multiple parties, either fail to provide provably strong mixing guarantees (such as sender anonymity or relationship anonymity) [1], [9] or unnecessarily amplify the latency or bandwidth overhead [4], [10], [11].

We close that gap by introducing a simple and efficient scaling technique with provably strong mixing guarantees, without introducing additional noise messages or amplifying the latency too much. Our method separates duties: heavy cryptographic public-key operations are performed by a large number of parties (in practice hundreds to thousands), ensuring some kind of unlinkability of the packets, while the mixing is performed in a computation-light manner by an ever changing selection of very few nodes (in practice 1 to 5), which we call the "funnels" for that round. With this construction, a large number of protocol parties perform the required (public-key) cryptographic computations for them before sending the processed messages to their next destination over secure channels (TLS). This way, our scaling technique distributes computation over an arbitrarily large set of parties while preserving almost the same degree of link saturation and the chance for messages to meet. This kind of funnel topology combined with a secure channel is a minor change to an AC protocol, but it seems to have eluded prior work and offers significant advantages for scalability. The simplicity of the construction enables us to prove strong anonymity properties with less noise messages and latency overhead than comparable provable AC protocols [10], [11].

We demonstrate the applicability of our scaling method

on a simple anonymous broadcast protocol, *Streams*, for which we prove a strong mixing property. *Streams* might be of independent interest for applications such as blockchain access privacy and anonymous microblogging, as it scales to a million messages and achieves practical end-to-end latency with strong mixing against honest-but-curious global attackers; these attackers can have strong background knowledge and access to honest-but-curious nodes and clients.

With a prototype implementation with one funnel node we demonstrate that *Streams* can scale to a million messages with an end-to-end latency of 16 seconds for each message for a message size of 512 bytes. Our proofs formally show that mixing occurs in the presence of a global passive adversary that (passively) compromises a fraction of the protocol parties. For a fraction of 10% compromised nodes in the system, *Streams* offers provable security of $\delta \leq 2^{-30}$.[1]

We find our scaling method for distributing computation over many parties to be not only relevant to *Streams* but also encouraging for protocol designers interested in scaling up other AC protocols. As three representative examples, we consider the prominent Loopix [9], Karaoke [4], and Vuvuzela [12] protocols, and describe how our method of divide and funnel can enhance their mixing properties.

2. Existing Mixnet Protocols and (Lack of) Provable Guarantees

Most mixnet-based anonymous communication (AC) protocols, that can scale to a large number of users, do so by splitting the traffic over multiple paths [1], [4], [9], [10], [13]–[17]. However, that also decreases the chance of two messages mixing with each other on a given hop — either those protocol need to increase the latency overhead almost linearly to maintain the same level of security [10], [17], or compensate with additional bandwidth overhead (by adding cover traffic) [4], [9], [13]–[15] or settle for a weaker level of anonymity [1], [9], [16]. With our scaling strategy, we aim to achieve a balance among those worlds: maximize the chance of mixing for two messages while not increasing the latency overhead linearly or introduce additional bandwidth overhead, and scale for a million messages in the system. To further motivate the requirement of a scaling technique with strong security guarantees, we discuss the tradeoffs provided by existing mixnet-based protocols below.

The recently proposed AC protocol Loopix [9] combines a stratified mix network architecture with exponentially distributed delays to achieve a flexible mixing protocol that does not require fixed rounds; and scales for many users by employing multiple paths. Loopix offers tuneable parameters that balances latency overhead and the required traffic volume to offer protection against traffic analyses, without providing a formal proof on the degree of anonymity/mixing for any given parameter set. It is not clear whether Loopix offers strong pairwise unlinkability for interesting latency numbers. In Section 9, we discuss how Loopix could improve their degree of mixing while scaling up by utilizing our scaling approach.

In Vuvuzela [12] each node processes all messages in the system, and hence, is limited by the processing power of each node, leading to a large latency (around 37 seconds with one million messages, with a chain of only 3 nodes). Our technique specifically circumvents such bottlenecks.

Karaoke [4] and Stadium [13] can scale to millions of users by allowing multiple paths. To achieve link saturation, they leverage a number of noise messages in $\Theta(|\text{servers}|^2)$, which yields a property similar to relationship anonymity. Additionally, by requiring the clients to send dummy messages whenever they don't have a real message to send, they allow the clients to hide when they are sending messages. However, these protocols only satisfy differential privacy (DP) guarantees. Differentially private AC protocols allow an attacker to develop a strong suspicion about who sent a message, which we strive to avoid in *Streams*. In Appendix A.2, we discuss how group privacy artifacts and millions of users render (ε, δ)-DP guarantees unsuited for providing strong anonymity guarantees, unless ε and δ are very small. In Section 9, we discuss how to scale up Karaoke with our approach without this explosion of noise messages.

Against passive attackers, Karaoke does not only claim to provide DP guarantees but statistical indistinguishability (negligible δ for $\varepsilon = 0$). The proof outline of Karaoke is, however, inaccurate; we show a counter-example in Appendix A.1. As Karaoke uses the same flawed argument to prove their DP guarantees, the soundness of their DP guarantees is not clear either.

Atom's [10] horizontal scaling technique also can handle a million short messages (up to 160 bytes) with strong mixing guarantees within a reasonable amount of time (around 28 minutes). However, our scaling technique allows our example protocol *Streams* to offer a significantly shorter end to end latency in the order of a few seconds. To demonstrate the effectiveness of our technique, we compare *Streams* with Atom and other protocols in Table 1, in terms of the number of communication hops required to achieve the security guarantees they aim for.

Ando et al. [17] propose a butterfly topology to achieve scalability and prove a mixing property. Butterfly networks are extremely effective in shuffling messages in the absence of compromised parties, but are not resistant in their presence. Even individual compromised nodes can jeopardize the mixing properties of the network. To overcome this challenge, Ando et al. propose to use $\Omega(\log^2(\eta))$ many rounds for the security parameter η in order to provide mixing in the presence of passively compromised nodes. As *Streams* funnels messages through single nodes it can resist passively compromised nodes with just $O(\log(\eta))$ rounds.

Kwon et al. [15] propose XRD that can achieve unobservability with $O(\log \mathsf{K})$ rounds of latency, where K is the total number of nodes in the system. For $\mathsf{K} \in poly(\eta)$, their latency is polylogarithmic in the security parameter η.

1. The $\delta \leq 2^{-30}$ characterizes the traffic leakage; the cryptographic primitives offer the same guarantees as in the literature. To put the order of magnitude of traffic leakage into perspective, attackers have to wait for users to send messages to make novel traffic-observations. Hence, the traffic leakage amplification is far less than for cryptographic primitives.

However, each user is required to send $\sqrt{2\mathsf{K}}$ messages per real message.

Recently, Langowski et al. [16] proposed Trellis, an anonymous broadcast protocol that follows the footsteps of cMix [18] to decouple expensive system setup from the broadcast stage. However, similar to cMix, unless they run the expensive system setup after every broadcast round, messages from the same user are linkable with each other on the last mixnode layer.

Most protocols require all clients to actively participate in the protocol to avoid leakage from user behaviour (e.g., if Bob is not active when a message is sent, Bob could not have been the sender); and *Streams* is no exception to that. However, many protocols (e.g., Vuvuzela, Stadium, Karaoke, XRD, Trellis) require all the clients to send messages in batches to help their mixing process, and/or to add strategic noise messages to the batches. This kind of synchronization among millions of clients is very difficult to achieve in a real world scenario. Our scaling technique does not require the help of noise messages from the servers to achieve mixing. Moreover, we disentangle the mixing property from user behavior. Hence, we only require the servers to have synchronized clocks but not the clients to use a synchronized usage pattern. We explain in Section 4.1 how our mixing property combined with such restricted client behavior directly implies the traditional anonymity notions achieved by the above protocols. Not requiring the messages to be processed in batches is a design advantage in itself.

3. Problem Overview And Design Roadmap

System Model. We consider a typical mix network based architecture [9], [10] allowing users to send messages anonymously using an overlay network of mix nodes. A set $\mathcal{S}$ of users communicate to a set $\mathcal{R}$ of recipients through a set I of intermediate nodes (or just 'nodes'). In real life, the same user can act as sender as well as recipient, however, we consider the sender role and recipient role as two separate logical entities. Each sender is denoted by u_i where $i \in \{1, \ldots, \mathsf{N}\}$ and $|\mathcal{S}| = \mathsf{N}$. Similarly, each recipient is denoted by R_i where $i \in \{1, \ldots, \mathsf{N}'\}$ and $|\mathcal{R}| = \mathsf{N}'$. We summarize the system parameters in Figure 1. Our main objective is to demonstrate our scaling technique by designing an end-to-end provably secure and scalable AC protocol. Similar to provably-secure mix-net systems such as [4], [10], [12], [13], [15], [16], [18], our protocol uses a round-based communication model and synchronized clocks. In Section 7.1 we extend our results to loosely synchronized clocks.

ℓ	Maximum latency allowed for a message
L	Minimum required latency of a message
I	Set of all nodes
I_h	Set of all honest nodes
K	Total number of nodes $\lvert I\rvert$
c	Number of compromised nodes $\lvert I - I_h\rvert$
O	An onion packet
η	The security parameter
δ	the adversarial advantage
$\xleftarrow{\$} [b, c]$	Draw uniformly at random from $[b, c]$

Figure 1: Protocol and system parameters

3.1. Attacker Model and Security Goals

We consider global network level adversaries that can also statically compromise all except two clients and up to c out of $\mathsf{K} = |I|$ nodes. We formally prove mixing properties against passive attackers and consider passive corruptions: the compromised protocol parties still follow the protocol specifications, however the adversary has access to all the internal states of a compromised party. In Appendix B we discuss the necessary adaptations for the protocol against active adversaries based on existing techniques.

Our scaling technique improves the core building block for anonymous communication. To precisely characterize the security notion that we consider, we utilize a property that we call *pairwise unlinkability*: the attacker shall not be able to figure out which of two messages entering a system at a similar time corresponds to which of the two same messages leaving the system at a later point. However, an adversary can additionally leverage differences in user behavior (e.g, if Alice is active only at a specific time of the day) to guess who might have sent a message. Even a trusted-third-party anonymizer can not defend against such leakage. Some AC protocols [4], [9], [10], [13] defend against that by restricting/enforcing how the protocol clients can behave. We consider that problem to be orthogonal and focus on the "mixing" problem. Our notion of pairwise unlinkability suitably captures that notion of mixing. It is also closely related to other prominent anonymity notions — in Section 4.1 we relate it to notions like sender anonymity [19] (which of two potential senders has sent a specific message?) and relationship anonymity [19] (who is in communication with who?).

As with other works on anonymous communication [3], [4], [6], [10], [13], [18], [20]–[22], our formal security analysis does not consider an adversary whose sole purpose is to launch denial-of-service (DoS) attacks. Any technique that can be deployed for other protocols against attacks like targeted flooding to degrade the performance of a node can be deployed in our system as well. However, we incorporate integrity protections in our sample protocol *Streams* using the standard cryptographic methods [23], and deploy countermeasures against DoS-anonymity attacks (loop messages) from the literature [9]. Additionally, in Sections 5.2 and 7.2 we discuss techniques to defend against some DoS attacks relevant to our scaling technique. We also leave the detailed analyses of fingerprinting of web-browsing and other side-channels that might arise in specific application scenarios for future work.

TABLE 1: Different tradeoffs offered by *Streams* compared to other protocols that can handle a total system load of one million messages. In our comparisons we consider the *trap variant* of Atom [10], since the NIZK (non-interactive zero knowledge) variant is significantly slower. Atom* denotes the estimated values for Atom to achieve $\delta < 2^{-30}$ with 10% corrupted nodes. For XRD [15], K denotes the total number of servers in their system. GPA refers to global passive adversaries.

Protocol	Anonymity property against GPA	Security	#hops	Latency (seconds)	Additional noise	Defense against active attacks
Atom [10]	sender anonymity	$\delta \leq 2^{-64}$	320	1680	none	trap messages + anytrust server group
Atom*	sender anonymity	$\delta \leq 2^{-30}$	120	630	none	trap messages + anytrust server group
XRD [15]	sender anonymity	$\delta \leq 2^{-64}$	22	128	$\sqrt{2\mathsf{K}}$ per message	NIZK
Karaoke [4]	relationship anonymity	DP	14	6	25000 by each server	server noise + bloom filter
Loopix [9]	sender anonymity	unknown	≥ 3	tunable	tunable	loop messages
Streams (ours)	pairwise unlinkability	$\delta \leq 2^{-30}$	32	16	none	loop messages

3.2. A Layman's Protocol With Provable Security

The ideal functionality presented in anonymity trilemma work [24] provides an important insight: *mixnets can achieve strongest degree of mixing by keeping all the messages together in every round.* Following that insight, let us first consider a round-based mixnet protocol that routes all the messages in the system via a common path, which achieves full node saturation and hence a strong degree of mixing: every message in a given round, except the messages that need to be delivered in that round, goes to the same node. The clients agree on the common path based on a randomness beacon; [2] for a given round r if the randomness beacon returns the string $\{x_r, x_{r+1}, \ldots, x_{r+\ell}\}$, any packet constructed at round r will be onion encrypted for the path of nodes $\{x_r, \ldots, x_{r+d}, R\}$ for a delay $d \leq \ell$ and recipient R intended by the client. If ℓ is polylogarithmic in the security parameter η, this protocol achieves provable mixing guarantees, however, the scalability is limited by the number of packets a node can process in a round.

3.3. Our Scaling Technique

Setup assumptions. We assume that each pair of nodes in the system maintain a persistent SSL/TLS session between them. Additionally, we leverage the *Sphinx* packet format [23] that provides end-to-end encryption for all messages and ensures that a node does not learn the path length and its own relay position on the path of a packet.

Scaling Technique (Divide and Funnel). We propose to separate duties such that, instead of one node processing all the onion packets in a round, many nodes come together to share the processing load, while the messages can "mix" in one node. Each round is separated into a *compute phase*, where the task of onion decryption to prohibit linking is distributed over all nodes, and a *funnel phase*, where a randomly chosen node collects and mixes all the messages.[3] The compute phase does not have a fixed time span. Immediately after processing, each compute node forwards the packet to the next funnel node. The funnel node uses the full time of each round. At the end of the round, the funnel node forwards the shuffled packets to the respective subsequent compute nodes based on the next node information in the Sphinx packet header. Figure 2 illustrates these compute and funnel phases.

We assume an authenticated and encrypted channel (using persistent TLS connections) between each pair of nodes. Therefore, even a global network level adversary cannot see the content of a packet passed between two nodes unless one of them is compromised; however the adversary can observe that a packet is passed between them.

The separation into funnel or compute node is conceptual: the same node can act as a funnel node or a compute node in different rounds. The funnel node in a round is picked using the value emitted by the randomness beacon. The client picks the compute nodes $\{x_1, \ldots, x_d\}$ uniformly at random (with replacement) from all available nodes for each hop of an onion packet independent of any other packet or any other hop of the same packet, and constructs the onion packet for the path $\{x_1, \ldots, x_d\}$. The client decides the delay d by picking a number from $[\frac{\ell}{2}, \ell - 1]$ following a distribution D. In general D can be any discrete probability distribution; however, in a typical setting we assume D to be uniform in $[\frac{\ell}{2}, \ell - 1]$. [4]

Funnel nodes act as mix nodes. All messages meet in the same funnel nodes as their paths are coordinated by the randomness beacon: A node acts as a funnel node in a round if the output of the beacon in that round matches its own id. The funnel node is a bottleneck of the system in terms of

2. The system could use a round-robin scheduling for funnel nodes; we choose to use randomness beacons instead so that an attacker cannot strategically compromise funnel nodes.

3. To simplify the explanation, we describe the protocol with one funnel per round. In Section 5.2, we discuss how the protocol can be extended for more than one funnel per round, and the corresponding tradeoffs.

4. Such a variable delay is not required to prove the pairwise unlinkability property. However, with a fixed delay for all messages, the actual anonymity set of a message would be limited to the messages sent in the same round. We avoid that by using such a randomized delay, for the purpose of designing an end-to-end protocol.

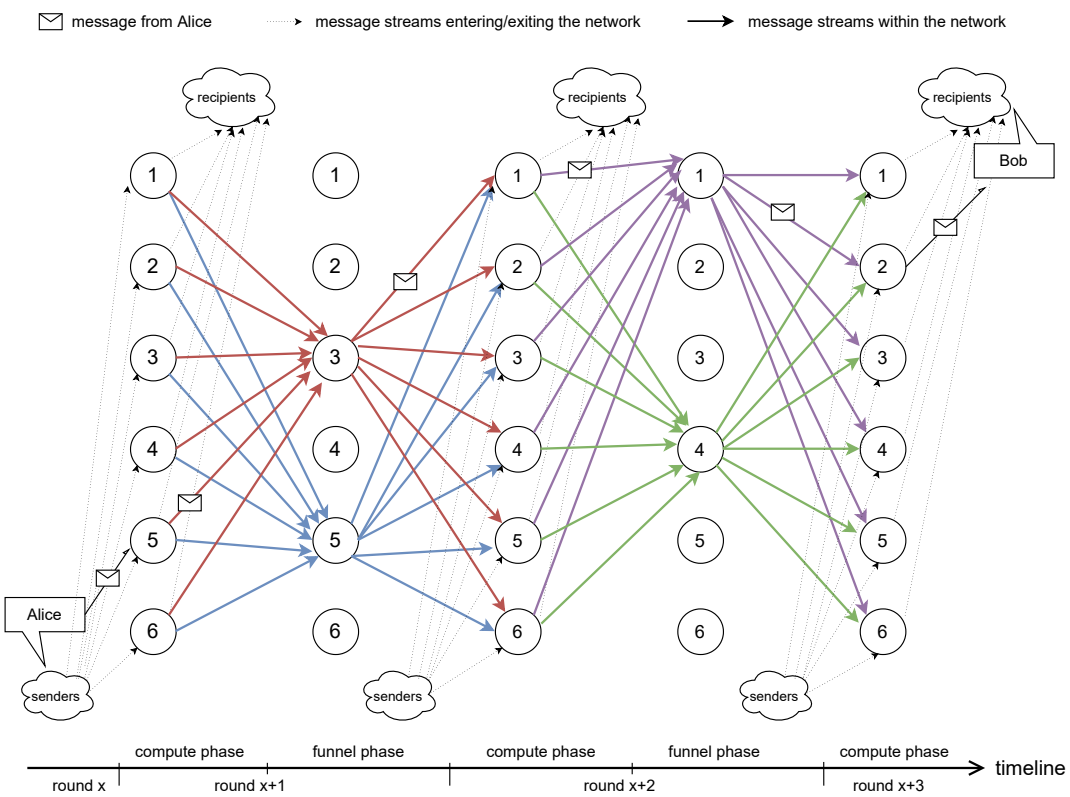


Figure 2: Divide and funnel routing strategy in *Streams*: in each round messages fan out in the compute phase to all available nodes to distribute the computation-heavy task, then they are funneled through a small number of nodes in the funnel phase to achieve better mixing. When Alice sends an onion packet in round x with onion layers for nodes $\{5, 1, 2\}$, the packet also goes through the funnel nodes $\{3, 1\}$ respectively, however onion decryption happens only on the compute nodes.

network bandwidth; however, it does not perform any heavy cryptographic computations, allowing the system to scale up to the full bandwidth capabilities.

Compute nodes act similar to onion routers [1], [25]: in every round a compute node peels an onion-layer for each packet and immediately forwards them to the designated funnel node. The pseudocode representations of the protocols run by honest clients and nodes are presented in Fig. 3.

If two messages are processed by some honest compute nodes (not necessarily same) in round r, and then both of them go through the same honest funnel node in round $r + 1$, the two messages achieve "mixing" even if the whole network before and after is compromised. In Figure 4, we pictorially show the possible cases when two messages can mix (or not).

Clients in *Streams* do *not* need to be synchronized: clients choose the path of compute nodes for their messages and then send them to the first such compute node. Clients need not be aware of the succession of funnel nodes or any round times.

4. Security Definition And Background

Anonymity properties, such as sender anonymity or relationship anonymity, depend heavily on the behavior of clients and their choices for the overall message latency: Even if the protocol in question implements a trusted third party it cannot hide which clients are sending messages at which time; moreover, if Alice and Bob send messages at different times, but the overall latency of each of those messages is drawn from the same (independent) distribution, then the recipients of said messages as well as a passive observer can learn information about the potential sender simply by analyzing the timing.

To avoid dealing with these client-dependent and distribution dependent aspects of anonymity we here focus on the degree of mixing provided by AC protocols. We formalize the question if the adversary could distinguish whether or not two messages, that spent at least a given amount of shared time in the protocol, could have been swapped along the way. This property is close to message swapping properties from the literature, such as tail indistinguishability by Kuhn et al. [26] and unlinkability by Kate et al. [27]. We assume an honest-but-curious global network level attacker that can eavesdrop on a fraction of the nodes (statically chosen), and has strong background knowledge about the behavior of the clients, formally the attacker controls all but two users.

The Pairwise Unlinkability game $\mathcal{G}_{\mathsf{PU}}^{\Pi,\mathcal{A},\mathsf{c},t}(1^\eta)$ for pro-

Client Protocol:

SendMessage(msg):

$d \leftarrow$ *DelayDistribution* $(\ell/2, \ell - 1)$
$p := \{x_1, \ldots, x_d\} \xleftarrow{\$} I^d$;
Construct onion packet O with path p and content (R, msg) /* R is the intended recipient for msg. */
Send $(\perp, \mathsf{O})$ to x_1.

Node Protocol:

Q_f, Q_c := queues to store incoming messages.
nodeID := the unique ID in $[0, \mathsf{K} - 1]$

IncomingMessage (**next hop** x**, onion packet** O**):**

if $x = \perp$ **then** ADD O to Q_c
else ADD $\{x, O\}$ to Q_f
end if

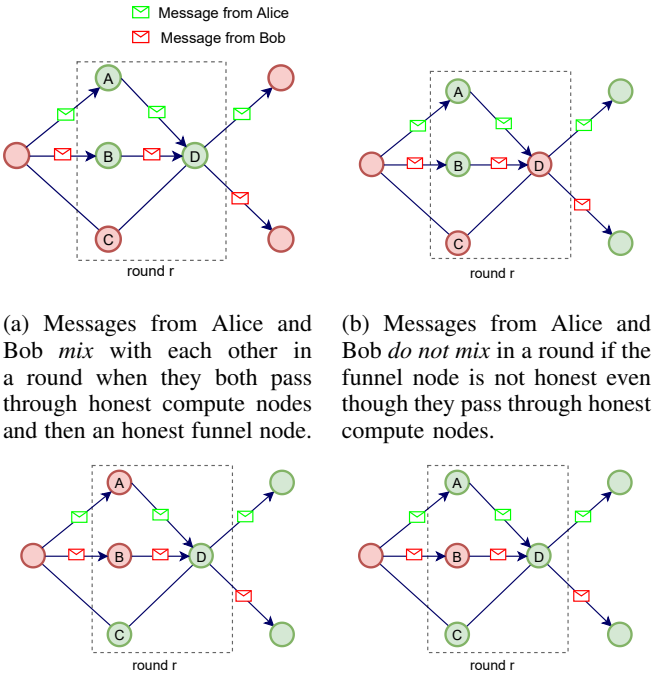
NewRound (**round number** r**):**

/* Compute phase in round r. */
/* All nodes act as compute nodes in compute phases.*/
funnel := Query the randomness beacon
tempQ := Dequeue all the elements from Q_c
while tempQ is not empty **do**
  O := dequeue an element from tempQ
  $\{x, O'\}$:= Remove one onion layer from O
  if $x \in \mathcal{R}$ **then** deliver O' to x
  else Forward the pair $\{x, O'\}$ to funnel over TLS
  end if
if nodeID = funnel **then**
  /* funnel phase in round r. */
  Wait for all the messages for round r from all nodes
  tempQ := Dequeue all elements from Q_f and shuffle
  while tempQ is not empty **do**
    $\{x, O\}$:= dequeue the first element from tempQ
    Forward $(\perp, O)$ to node x over TLS

Figure 3: Protocol design of *Streams* with *divide-and-funnel* scaling technique, where each round has *compute* and *funnel* phases — in the compute phase the task of onion decryption to prohibit linking is distributed over all nodes, and in the funnel phase all the messages are collected and mixed by a single randomly chosen funnel node.

tocol Π against adversary $\mathcal{A}$ can be described as follows:

- The challenger Ch provides the adversary $\mathcal{A}$ with the description of Π (that includes the description of the sets $\mathcal{S}$, $\mathcal{R}$, and I).
- $\mathcal{A}$ statically corrupts all recipients in $\mathcal{R}$, all senders in $\mathcal{S}$ except from a pair u_0, u_1, and a subset of I denoted by I_{corr}, such that $|I_{\mathsf{corr}}| \leq \mathsf{c}$ (i.e., no more than c nodes are corrupted).
- Ch and $\mathcal{A}$ engage in an execution of Π where Ch acts on behalf of u_0, u_1 and the honest nodes, while $\mathcal{A}$ controls the corrupted parties and monitors the network traffic as a global passive adversary.
- At any time t', $\mathcal{A}$ sends a pair of challenge messages $(u_0, m_0, t_{s,0}, t_{f,0}, R_0)$ and $(u_1, m_1, t_{s,1}, t_{f,1}, R_1)$ to Ch where u is the sender of the message, m the content, t_s the time the message enters the system, t_f the time the message leaves the system, and R the receiver of the message, with $\min(t_{f,0}, t_{f,1}) - \max(t_{s,0}, t_{s,1}) \geq t$ and $t' < \min(t_{s,0}, t_{s,1})$.
- In turn, Ch chooses a random bit $b \in \{0, 1\}$ and initiates the challenge transmissions according to the following cases:
  - If $b = 0$, Π transmits the messages $(u_0, m_0, t_{s,0}, t_{f,0}, R_0)$ and $(u_1, m_1, t_{s,1}, t_{f,1}, R_1)$.
  - If $b = 1$, Π transmits the messages $(u_1, m_0, t_{s,1}, t_{f,0}, R_0)$ and $(u_0, m_1, t_{s,0}, t_{f,1}, R_1)$.
- $\mathcal{A}$ can terminate the game any time by outputting a bit b^*, as a guess for the challenge bit b.
- The game returns 1 if and only if $b^* = b$ (i.e., $\mathcal{A}$ guesses correctly), otherwise the game returns 0.



(a) Messages from Alice and Bob *mix* with each other in a round when they both pass through honest compute nodes and then an honest funnel node.

(b) Messages from Alice and Bob *do not mix* in a round if the funnel node is not honest even though they pass through honest compute nodes.

(c) Messages from Alice and Bob *do not mix* with each other in a round if none of them passes through honest compute nodes in the same round.

(d) Messages from Alice and Bob *do not mix* with each other in a round if both of them do not pass through honest compute nodes in the same round.

Figure 4: Cases where two messages mix (or not).

Definition 1 (Pairwise unlinkability)**.** *A protocol* Π *provides* pairwise unlinkability *of messages over time* t *and* c *compromise up to probability* δ *for* $0 \leq \delta < 1$ *if, for all probabilistic polynomial time (PPT) adversaries* $\mathcal{A}$ *passively and statically compromising at most* c *nodes, the following holds:*

$$\Pr\left[\mathcal{G}_{\mathsf{PU}}^{\Pi,\mathcal{A},\mathsf{c},t}(1^\eta) = 1\right] - \Pr\left[\mathcal{G}_{\mathsf{PU}}^{\Pi,\mathcal{A},\mathsf{c},t}(1^\eta) = 0\right] \leq \delta(\eta).$$

Informally, we say that the two messages *are shuffled* from the adversary's point of view if a protocol achieves pairwise

unlinkability, which means that the messages are shuffled almost as good as a trusted-third-party anonymizer could shuffle them. We say that a protocol achieves *strong* pairwise unlinkability if δ is negligible in a security parameter η. Ideally, we want our protocol to achieve strong pairwise unlinkability. We discuss below how pairwise unlinkability relates to sender anonymity and relationship anonymity.

4.1. Pairwise Unlinkability and Anonymity

Pairwise unlinkability is closely related to well-known anonymity notions like sender anonymity and relationship anonymity, while avoids the leakage from client behaviour. We informally explain their relationship below, and refer to Appendix D for a more formal argument.

Sender anonymity. The common anonymity notion *sender anonymity* states that the recipient of a message cannot distinguish whether the message originated in one sender over another sender, even for a pair of potential senders of the adversary's choice. This closely resembles pairwise unlinkability with one key difference: sender anonymity typically talks about a single challenge message, not about a pair of messages. To bridge the gap between sender anonymity and pairwise unlinkability we can require a degree of bandwidth overhead, such as ensuring that all senders communicate regularly or send dummy messages. However, even if dummy messages are sent, an adversary might still deduce the challenge sender from timings alone.

If, say, the adversary observes Alice sending a message in round t and Bob sending a message in round $t+2$, the arrival time of the challenge message together with the distribution of the latency might tell the adversary who of them is more likely to have sent the challenge message. In the simplest example, for a constant latency, the adversary could immediately exclude one of them from being the challenge sender. This apparent attack is independent of how a protocol achieves anonymity and even applies if the messages are kept in a trusted third party for the same amount of time. However, if a protocol follows batch processing (i.e., all the messages are sent in the same round as in Karaoke and Atom), pairwise unlinkability immediately implies sender anonymity.

Relationship anonymity. A similar notion states that if two senders send one message each to two receivers, a third party is unable to determine which sender talks to which receiver significantly better than purely guessing. Loopix [9] calls this property *Sender-Receiver Third-party Unlinkability*. Given that the two messages in question are sent in the same round and that both senders choose the latencies from the same distribution, pairwise unlinkability immediately implies this anonymity property.

5. Formal Protocol Description

We use a hybrid world UC model [28] to present our scaling technique with our core protocol *Streams*– where the protocol has access to some additional ideal (hybrid) functionalities that is available to the protocol as well as the adversary. A protocol party (an honest user or node) or the adversary can access such a functionality through an incorruptible ITI $\mathcal{F}$ that provides certain ideal guarantees, e.g., clock time, key registration etc. More specifically, our formalization uses four such functionalities: a round-based communication functionality $\mathcal{F}_{round}$, a globally available randomness beacon $\mathcal{F}_{CRF}$, a key registration functionality $\mathcal{F}_{RKR}$, and a secure channel functionality $\mathcal{F}_{SCS}$. The environment $\mathcal{E}$ can access those ideal functionalities either through the protocol parties or through the adversary.

Rounds := {**false**, . . . , **false**} // Array length $\mathsf{K}+\mathsf{N}+\mathsf{N}'$
Round := 0; PartiesIncremented := 0
QUEUE = a queue where the incoming messages are stored

QueryRound() from $\mathcal{A}$ or $\mathcal{F}_{CRF}$ or party i:
return {Round, Rounds}

RequestRound() from party i:
return Rounds[i]

NextRound() from party i:
if Rounds[i] = **true** **then**
return "invalid action"
else
Rounds[i] ← **true**; PartiesIncremented += 1
if PartiesIncremented = $\mathsf{K}+\mathsf{N}+\mathsf{N}'$ **then**
Round += 1; PartiesIncremented ← 0
Reset Rounds[j] := **false** $\forall j : 0 \leq j < \mathsf{K}+\mathsf{N}+\mathsf{N}'$
Forward all elements of QUEUE to $\mathcal{A}$; empty QUEUE

Upon receiving msg $(P, P_{next}, O, \mathsf{round})$ **from** *Streams*
if round = Round **then**
ADD $(P, P_{next}, O, \mathsf{Round})$ to QUEUE

Figure 5: Round Functionality $\mathcal{F}_{round}$

Round Functionality $\mathcal{F}_{round}$. We introduce a hybrid functionality $\mathcal{F}_{round}$ (see Figure 5) to enforce rounds on the protocol parties. We ensure that the environment $\mathcal{E}$ activates the honest parties in every round. $\mathcal{F}_{round}$ ensure, though, that the environment $\mathcal{E}$ cannot activate a protocol party multiple times in the same round by keeping track of the Rounds[i] flag for each party i (including both clients and nodes). Additionally, it ensures that all the network packets intended to send for a given round is not send before or after that round to an honest protocol party. As a consequence, the environment can stop the entire protocol at anytime. As then no messages would be delivered anymore, stopping the entire execution does not leak any information to the environment.

Randomness Beacon Functionality $\mathcal{F}_{CRF}$. We assume that each protocol party (including the adversary) has access to an incorruptible randomness beacon. In particular, future values of this beacon are not known to the adversary. We model this beacon with the ideal functionality $\mathcal{F}_{CRF}$ (see Fig. 6) that outputs each time a ℓ-long substring of an infinite

crf = an infinitely long random string.

GetFunnels(ℓ):

rnd, _ $\leftarrow$ QueryRound()
return $\{\mathsf{crf}[\mathtt{rnd}] \mod \mathsf{K}, \ldots, \mathsf{crf}[\mathtt{rnd}+\ell-1] \mod \mathsf{K}\}$

Figure 6: Randomness Beacon Functionality $\mathcal{F}_{CRF}$

random string beacon. Using that ℓ-length string a protocol party can derive the next ℓ funnel nodes.

Secure Channel Functionality $\mathcal{F}_{SCS}$. We also use the secure communications sessions functionality $\mathcal{F}_{SCS}$ from the work of Gajek et al. [29, Figure 4]. They show that $\mathcal{F}_{SCS}$ abstracts the TLS [30] protocol. It is crucial to note here that all the protocol parties in our model work in rounds, and therefore, $\mathcal{F}_{SCS}$ as well forwards all the messages to the $\mathcal{F}_{round}$ functionality instead of the environment; the $\mathcal{F}_{round}$ functionality in turn forwards those messages to the environment when the round ends.

Packet format. We use an improved version [31] of the Sphinx packet design [23] to ensure that all messages are end-to-end encrypted; we call them "onion packets". The Sphinx packet design guarantees that an intermediate node, just by looking at a packet, does not learn anything other than the next node on the path — it hides the path length and the position of the node on the path. The security properties of the packet design is already incorporated in the onion routing subprotocol Π_{sub} that we use from the work of Kuhn et al. [26].

Key registration functionality. Practical realizations of onion encryption functionality and secure channel functionality in turn assume the availability of a public key infrastructure (PKI) to all the users and nodes — which is no different from any other mixnet-based design. We consider the key registration functionality $\mathcal{F}_{RKR}$ that realizes such PKI setups, and handles all cryptographic operations. $\mathcal{F}_{RKR}$ is solely used by the subprotocols Π_{sub} and $\mathcal{F}_{SCS}$, which are treated in a black-box manner throughout this section. For completeness, we provide a description of Π_{sub}, $\mathcal{F}_{SCS}$ and $\mathcal{F}_{RKR}$ in Appendix E.

5.1. The Core Protocol

Our protocol has two kinds of parties — clients and nodes. So we define our protocol in two parts as well — clients and nodes. Additionally, the protocol parties as well as the adversary have access to the hybrid functionalities as described above.

Each round is then separated into a *compute phase*, where the task of onion decryption to prohibit linking is distributed over all nodes, and a *funnel phase*, where a randomly chosen node collects and mixes all the messages. In UC-realization of our protocol, we split those two phases into two separate rounds, to avoid having two sequential communications within a single round. Therefore, one single

QUEUE = a FIFO queue.

SendMessage(msg, R) **from party** i

$r \leftarrow QueryRound()$
if round $\neq r$ **then**
  reject packet and exit
ADD (msg, R) to QUEUE

Upon new round from $\mathcal{E}$**:**

boolean flag := **RequestRound**() //defined in $\mathcal{F}_{round}$
if flag $\neq$ **true then**
  return "invalid action"
if round $\mod 2 = 0$ // compute round **then**
  while QUEUE is not empty **do**
    $m := (msg, R) \leftarrow$ dequeue QUEUE;
    $d \leftarrow DelayDistribution\left(\frac{\ell}{2}, \ell - 1\right)$
    $p := \{x_1, \ldots, x_d\} \xleftarrow{\$} I^d$;
    call `Process_new_onion`$(self, m, d, p)$ from Π_T
NextRound()

Upon receiving a message m **from** Π_{sub}**:**

Output "Message m received" to $\mathcal{E}$

Figure 7: Client Protocol Design Π_{client} as described in Section 5.1.

Π_T:`Process_new_onion`$(self, msg, d+1, p)$

Call `Process_new_onion`$(self, msg, d+1, p)$ from Π_{sub}.
Intercept the network packet $packet$ and send it to Π_{rer}.

Π_T: `Forward_Onion`(O)

Call `Forward_Onion`(O) in the subprotocol Π_{sub}.
Intercept the network packet $packet$ and send it to Π_{rer}.

Π_{rer}: **Upon a packet** $packet$

_, $funnel \leftarrow$ GetFunnels(2) // Select next funnel node
Send $packet$ over $\mathcal{F}_{SCS}$ to $funnel$.

Figure 8: $\Pi_T$and Π_{rer} for the core *Streams* protocol (with single funnel every round).

round of our original protocol maps to two rounds in the UC-version. In the compute round, each compute node processes the packets with them and forwards them to the designated funnel node, then in the funnel round, the funnel node shuffles all the messages received in the last round, and forwards them to the respective subsequent compute nodes.

The funnel nodes are picked using the string emitted by the randomness beacon $\mathcal{F}_{CRF}$. Each client picks the compute nodes uniformly at random (with replacement) from all available nodes for each hop of an onion packet independent of any other packet or any other hop of the same packet. All the packets in every even round go to the next designated funnel node. Then in the next (odd) round, the funnel node shuffles all the received packets and forwards them (without any cryptographic operation) to the compute

INPUT_QUEUE = queue to store incoming messages
OUTPUT_QUEUE = queue to store outgoing messages
nodeID := a unique ID in $[0, \mathsf{K}-1]$

Upon input message (onion packet O):

ADD O to INPUT_QUEUE

Upon new round from $\mathcal{E}$:

boolean flag := **RequestRound**() //defined in $\mathcal{F}_{round}$
if flag $\neq$ **true then**
return "invalid action"
funnels := GetFunnels(1)
if round $\mod 2 = 1$ AND nodeID $\in$ funnels **then**
Π_{funnel}
else if round $\mod 2 = 0$ **then**
Π_{worker}
swap INPUT_QUEUE and OUTPUT_QUEUE.
NextRound()

Π_{funnel}**:**

Shuffle OUTPUT_QUEUE
while OUTPUT_QUEUE is not empty **do**
$O \leftarrow$ dequeue first element from OUTPUT_QUEUE
Forward O to $\mathcal{F}_{SCS}$

Π_{worker}**:**

while OUTPUT_QUEUE is not empty **do**
$O \leftarrow$ dequeue first element from OUTPUT_QUEUE
call Π_T : **Forward_Onion**(O)

Figure 9: Node Protocol Design as described in Section 5.1. The design remains the same with multiple funnels per round (as described in Section 5.2), except the set *funnels* contains multiple elements. The corresponding statements are colored with teal.

$\rho :=$ number of funnels per round.

GetFunnels(ℓ):

round, $\leftarrow$ QueryRound()
return $\big(\{\mathsf{crf}[\mathtt{round}\cdot\rho] \mod \mathsf{K}, \ldots, \mathsf{crf}[\mathtt{round}\cdot\rho+\rho-1] \mod \mathsf{K}\}, \ldots, \{\mathsf{crf}[(\mathtt{round}+\ell-1)\rho] \mod \mathsf{K}, \ldots, \mathsf{crf}[(\mathtt{round}+\ell-1)\rho+\rho-1] \mod \mathsf{K}\}\big)$

Π_{rer}**: Upon a packet** *packet*

_, funnels $\leftarrow$ GetFunnels(2)
nextFunnel $= \mathsf{H}(packet, \mathtt{funnels}) \mod \rho$ // H is a cryptographic hash function
Send *packet* over $\mathcal{F}_{SCS}$ to funnels[nextFunnel].

Figure 10: Modifications for *Streams* with multiple funnel nodes per round.

nodes based on the next node information in the Sphinx packet header. Then, again in the even round, the compute node removes one layer of the onion packet, and forwards the packet immediately to the next designated funnel node.

The packets are onion encrypted only for the compute nodes, not for the funnel nodes. Additionally, we assume authenticated and encrypted channel between each pair of nodes which is realized by the $\mathcal{F}_{SCS}$ functionality. The pseudocode representations of the protocols run by each honest client and each honest node are presented in Fig. 7 and Fig. 9 respectively.

We aim for anonymous broadcast to the network. Therefore, in our protocol *Streams*, messages from our last node are sent to the environment instead of delivering them to an explicit receiver. The delivery of messages occurs through the environment which controls the network functionality.

5.2. Multiple Funnel Nodes Per Round: Performance vs. Resiliency Tradeoff

Having a single funnel in *Streams* inherently creates single points of failure: if a malicious node is chosen as funnel it can perform denial-of-service (DoS) attacks by dropping packets. Arguably this is not a new concern for anonymous communication protocols — corrupted mixnodes in existing mixnet designs can drop all packets passing through them, and those protocols use additional integrity measure to defend agianst them. However, funneling every packet through the same node gives this node the power to disrupt the entire stream of packets and makes the node a prime target for external DoS attackers. Our scaling methodology can be extended to several funnel nodes at a time, which presents a trade-off between performance and robustness against such DoS attacks. We additionally refer to Appendix B for the heuristic mechanisms that would allow detecting and defending against such active attacks; and would deter a funnel from launching such a disruption.

Instead of a single funnel node per round, the packets are distributed among ρ funnel nodes, each of which is still chosen based on the randomness beacon. We suggest that ρ remains a small number (between 2 and 5) chosen based on the desired level of robustness. Compute nodes can distribute packets among the h funnel nodes based on a deterministic function applied to the packet digest and the randomness beacon, e.g., a hash function with the (encrypted) packet content and the beacon's current randomness as inputs.

The pseudocode representation of the client (c.f. Fig. 7) and the node (c.f. Fig. 9) remains mostly same as the core protocol (with single funnel) with minor modifications which we present in Fig. 10.

6. Security Analysis

Here we analyze the security of the core protocol against a global passive adversary that can passively compromise (the compromised parties still follow the protocol) a portion of the nodes. We use existing techniques from the literature to provide integrity measures against active adversaries and discuss them in Appendix B.

6.1. Key Proof Ideas

The key idea is that two messages get shuffled if they go through an honest funnel node right after going through honest compute nodes (c.f. Figure 4). It is not necessary that they pass through the same honest compute node, however, they need to pass through some honest compute nodes in the same round just before passing through an honest funnel node. If the messages stay in the system long enough (L is sufficiently high), with high probability such a sequence will occur at least once.

Single Funnel Per Round. Let a be the probability of a randomly picked node being honest; $a = \frac{\mathsf{K}-\mathsf{c}}{\mathsf{K}}$. Since both the funnel node and each compute node are picked uniformly at random (with replacement, and independent of all other nodes), the probability that the funnel node or compute node chosen in any round is compromised is $\frac{\mathsf{c}}{\mathsf{K}} = (1-a)$, and the probability that it is honest is a. Therefore, the probability that the two messages mix in for a given pair of compute nodes in round r and funnel node in round $(r+1)$ is a^3, and the probability that they don't mix in this pair of rounds is $(1-a^3)$. If two messages both stay in the system for L rounds, the probability that they don't mix is at most $\delta < \left(1-a^3\right)^L$.

Multiple Funnels Per Round. When there are ρ funnel nodes per round, we need to consider the probability that the two messages might not go through the same funnel node in a round. For any given round, the probability that the pair of messages meet in the same funnel node is $\frac{1}{\rho}$. Given that they have met on a funnel node in a given round, the probability of that specific funnel node being compromised is still $\frac{\mathsf{c}}{\mathsf{K}} = (1-a)$; and the probability that it is honest is a. Overall, the probability that the two packets mix in a given round is bounded by $Y = \frac{1}{\rho} \times a^3$. The probability that they do not mix is at most $\delta \leq (1 - \frac{a^3}{\rho})^L$.

6.2. Proof Roadmap

For proving security, we first prove that an intermediary representation, a UC ideal functionality $\mathcal{F}_{\textit{Streams}}$ (defined in Figure 11), of *Streams* that does not rely on cryptographic operations but on shared memory. This ideal functionality $\mathcal{F}_{\textit{Streams}}$ is carefully crafted such that all attacks on *Streams* can be mounted on $\mathcal{F}_{\textit{Streams}}$ as well, against a wide range of attacker capabilities. In a second step, we prove pairwise unlinkability for the faithfully abstracted ideal functionality, which in turn implies pairwise unlinkability for *Streams*. Here we present the main security theorems and their implications, and refer an interested reader to Appendix C for the detailed proofs.

We can say that, at the expense of latency, our ideal functionality provides pairwise unlinkability — if two messages stay together in $\mathcal{F}_{\textit{Streams}}$ for a sufficiently long time (poly-logarithmic in the security parameter), they get shuffled.

Theorem 1 (Pairwise unlinkability of $\mathcal{F}_{\textit{Streams}}$)**.** *Given a constant integer ρ, if the amount of compromised nodes*

$\textit{inputBuffer}[]$ array of queues to store messages for nodes
crf = an infinitely long random string
queue = a hashmap
$\texttt{round} := 0$; $\textit{newRound}[] := \{\texttt{false}, \texttt{false}, \ldots\}$;
$\textit{partyCount} := 0$

Upon new round from $\mathcal{E}$ for party P:

if $\textit{newRound}(P) =$ **true** **then**
  return "invalid action"
set $\textit{newRound}(P) :=$ **true** ; $\textit{partyCount} += 1$
if $\texttt{round}$ is odd (funnel round) AND P is a client **then**
  $(m, t) \leftarrow$ dequeue $\textit{inputBuffer}[P]$
  $d \leftarrow \textit{DelayDistribution}(\frac{\ell}{2}, \ell - 1)$; $\{x_1, \ldots, x_d\} \xleftarrow{\$} I^d$
  if $!\exists x \in \{x_1, \ldots, x_d\}$ such that $x \in \mathtt{I_h}$ **then**
    Send $(m, x_1, \ldots, x_d)$ to $\mathcal{S}$
  else
    let $x_a :=$ the **first** honest party on the path $\{P, x_1, \ldots, x_d\}$
    Generate a random message q
    Send $(q, x_1, \ldots, x_a)$ to $\mathcal{S}$
    store $(q, x_a, m, x_{a+1}, \ldots, x_d)$ in $\textit{queue}(\texttt{round} + a)$
if $\texttt{round}$ is even (compute round) AND $\textit{partyCount} = \mathsf{N} + \mathsf{K}$ **then**
  SendInformation()
NextRound(P)

Upon input message (m, t) from $\mathcal{E}$ for party P:

if $\texttt{round} \neq t$ **then**
  reject packet and exit
Add (m, t) in $\textit{inputBuffer}[P]$

Upon receiving a message m for party P:

Output m to $\mathcal{E}$

Figure 11: Ideal functionality $\mathcal{F}_{\textit{Streams}}$

SendInformation()

$\_, \texttt{funnels} \leftarrow$ GetFunnels(2)
$y = \mathsf{H}(\textit{packet}, \texttt{funnels}) \mod \rho$
for each $(q, x_a, m, x_{a+1}, \ldots, x_d) \in \textit{queue}(\texttt{round})$ **do**
  Remove $(q, x_a, m, x_{a+1}, \ldots, x_d)$ from $\textit{queue}(\texttt{round})$
  $\textit{link} := (q, y)$
  if $y \in I_h$ **then**
    $\textit{link} := (\perp, y)$
  $x_\phi :=$ **next** honest node on the path $\{x_{a+1}, \ldots, x_d\}$
  if there is **no** such x_ϕ **then**
    Add $(\textit{link}, m, x_{a+1}, \ldots, x_d)$ in a temporary queue Q
  else
    generate a random message q'
    Add $(\textit{link}, q', x_{a+1}, \ldots, x_\phi)$ in Q
    Add $(q', x_\phi, m, x_{\phi+1}, \ldots, x_d)$ to $\textit{queue}(\texttt{round} + \phi - a)$
Shuffle the elements of Q and send them to $\mathcal{S}$

Figure 12: Leakage from the ideal functionality $\mathcal{F}_{\textit{Streams}}$

is a constant fraction $\frac{c}{K} < 1$, $\mathcal{F}_{Streams}$ *provides pairwise unlinkability of messages over* $\mathfrak{L}$ *rounds up to probability* δ *as in Definition 1, where* $\delta < \gamma^{\mathfrak{L}/2}$ *with* $\gamma = 1 - \frac{\left(\frac{K-c}{K}\right)^3}{\rho}$.

an ideal functionality $\mathcal{F}$ *is realized by* a protocol Π if all attacks (within the execution model) that can be mounted on Π can be translated to attacks on Π, for a wide range of attacker capabilities. An ideal functionality, like $\mathcal{F}_{Streams}$, can abstract away from cryptographic details while faithfully modeling all weaknesses of the protocol. Our protocol *Streams* UC-realizes the ideal functionality $\mathcal{F}_{Streams}$.

Theorem 2. *For any subprotocol* Π_{sub} *in the* $\mathcal{F}_{RKR}$*-hybrid model that UC realizes* $\mathcal{F}_{sub}$*, the anonymity protocol Streams from Section 5.2 using the subprotocol* Π_{sub} *in the* $\mathcal{F}_{CRF}$*,* $\mathcal{F}_{RKR}$*,* $\mathcal{F}_{SCS}$*,* $\mathcal{F}_{round}$*-hybrid model UC-realizes* $\mathcal{F}_{Streams}$ *in the* $\mathcal{F}_{round}$*-hybrid model.*

From the work by Kuhn et al. [26] we know that the Sphinx version [31] we use in our protocol UC realizes $\mathcal{F}_{sub}$ under a PKI setup (realized by the hybrid functionality $\mathcal{F}_{RKR}$), and we can state the following lemma.

Lemma 1. *Sphinx packet format from [31] UC realizes* $\mathcal{F}_{sub}$ *in the* $\mathcal{F}_{RKR}$ *hybrid model.*

As a corollary to Theorem 2, Theorem 1, and Lemma 1 we can state the following for our protocol *Streams*.

Theorem 3 (Security of *Streams*)**.** *In the* $\mathcal{F}_{RKR}$*-hybrid model, given a constant integer* ρ *and a constant fraction* $\frac{c}{K} < 1$*, Streams provides pairwise unlinkability of messages over* $\mathfrak{L}$ *rounds up to probability* δ *as in Definition 1, where* $\delta < \gamma^{\mathfrak{L}/2}$ *and* $\gamma = 1 - \frac{\left(\frac{K-c}{K}\right)^3}{\rho}$.

As a corollary to the above theorem, we can state the following about the core protocol from Section 5.1 for $\rho = 1$ (single funnel per round).

Theorem 4 (Security of *Streams* core)**.** *In the* $\mathcal{F}_{RKR}$*-hybrid model, given a constant fraction* $\frac{c}{K} < 1$*, Streams from Section 5.1 provides pairwise unlinkability of messages over* $\mathfrak{L}$ *rounds up to probability* δ *as in Definition 1, where* $\delta < \gamma^{\mathfrak{L}/2}$ *where* $\gamma = 1 - \left(\frac{K-c}{K}\right)^3$.

In Theorems 3 and 4, for all $\mathfrak{L} \in \omega(\log \eta)$ for a security parameter η the δ is negligible, and all pair of messages that stays together in the protocol for at least $\mathfrak{L}$ rounds, get shuffled with overwhelming probability. Recall that $\mathfrak{L}$ rounds of the protocol in the UC-framework translates to $L = \mathfrak{L}/2$ rounds (considering the compute phase and the funnel phase as a single round) in the original protocol.

6.3. Analysis

Here we first analyze Theorem 4 to evaluate the simple case of introducing single funnel node per round, and then we evaluate the impact of introducing multiple funnel by analyzing Theorem 3.

In Theorem 4, $\gamma = 1 - (\frac{K-c}{K})^3$ is conceptually the proportion of compute node and funnel node pairs in a round where the two messages cannot mix. Figure 13a plots the relationship between γ and the fraction $\frac{c}{K}$ of compromised parties. If we want to have the same level of concrete security as without compute and funnel nodes, we need to increase latency, or with similar latency the protocol can only be resilient against lesser fraction of compromised nodes. However, one advantage of this construction is that the cost or overhead does not increase linearly with the number of clients, or more importantly, does not even depend on the number of clients. In Fig. 13b, we compare the latency overhead needed for our protocol (with compute and funnel separation) to achieve a given level of security with the scenario where the compute and funnel separation is not required (refer to Theorem 8 for security of our layman's protocol). For $\frac{c}{K} \leq 0.2$ the number of rounds only doubles with the separation of duties to achieve the same level of security. Even though the compute and funnel method provides scalability at the cost of security, δ still decreases exponentially with latency. We show the relationship between them in Fig. 13c.

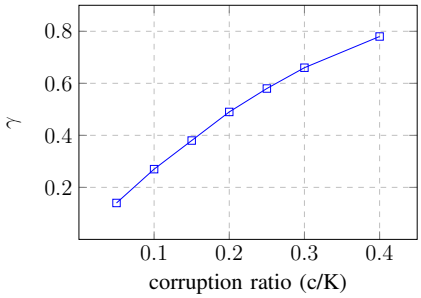
Impact of Multiple Funnels. We plot the trade-off between performance and the choice of ρ (up to $\rho = 4$) in Fig. 14, with a security goal of $\delta < 2^{-30}$ and the system handles one million packets per round. As we show in Fig. 14, the latency needs to increase with the number of funnel nodes per round, to process the same total number of messages (1 million) and to achieve the same level of security ($\delta \leq 2^{-30}$). Note that the funnel nodes are only restricted by bandwidth availability, but not processing power in our system: it takes less than 120 milliseconds to process one million messages by a single funnel node, c.f. Figure 15c. Therefore, we can decrease the duration of each round with the number of funnel nodes ρ (to process one million messages per round) For example: for $\rho = 4$, the average processing time for each funnel is around 30 milliseconds; an overall round duration of 250 milliseconds still leaves more than 200 milliseconds for communication delay and loose synchronization.

7. Resiliency Improvements

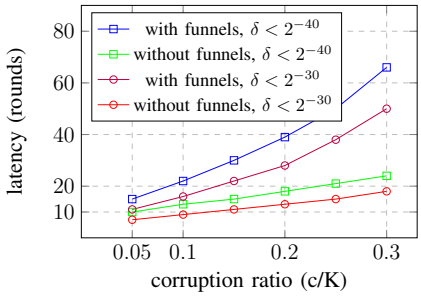
7.1. Resiliency for Loose Synchronization

So far we assumed that all protocol parties are perfectly synchronized (c.f. Section 5). As maintaining such a synchronization continuously can be challenging, we here discuss relaxing that assumption by allowing each protocol party to follow their own local clock.

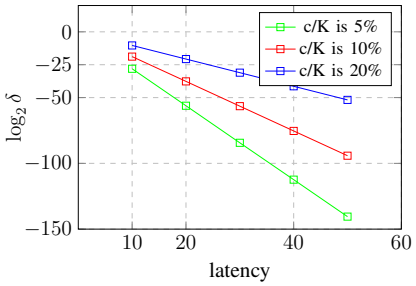
We assume that the maximum difference between the local clocks of any two nodes is bounded by μ milliseconds. The clients do not need to keep track of rounds and can send messages to the system whenever they want. A client sends an onion packet to the first compute node on the onion path. The compute node, based on its local clock, decides which funnel node to forward the packets to. As long as



(a) corruption ratio $\frac{c}{K}$ vs. the probability γ of two messages not mixing in a randomly chosen funnel node and its predecessors (compute nodes) in a pair of paths.

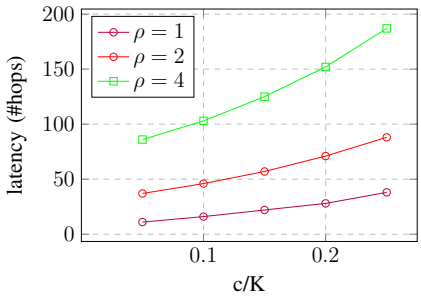


(b) corruption ratio $\frac{c}{K}$ vs. the required latency (in rounds) to achieve different levels of security, viz, one in a billion ($\delta < 2^{-30}$) and one in a trillion ($\delta < 2^{-40}$) security.

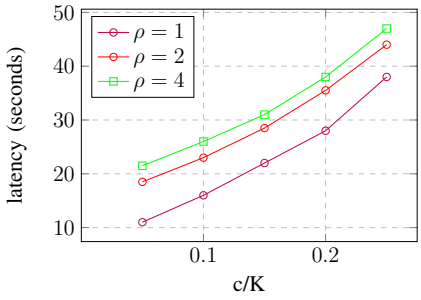


(c) latency (in rounds) vs. $\log \delta$ for different values of corruption ratio $\frac{c}{K}$ (e.g. 5%, 10%, 20%). The linear decrease in $\log \delta$ means exponential decrease in the δ value.

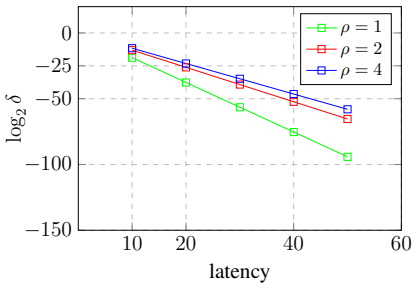
Figure 13: Effective security after introducing compute and funnel phases



(a) $\frac{c}{K}$ vs. the required latency ℓ (number of hops) for different values of ρ to achieve one in a billion ($\delta < 2^{-30}$) security.



(b) $\frac{c}{K}$ vs. latency (seconds) for different values of ρ to achieve $\delta < 2^{-30}$ security, to process one million packets per round.



(c) latency (in seconds) vs. $\log \delta$ for $\frac{c}{K} = 0.1$ and $\rho = 1, 2, 4$. The linear decrease in $\log \delta$ means exponential decrease in the δ.

Figure 14: Performance vs. Resiliency against single point of failure (ρ = #funnels per round) for *Streams*. Note that the round size decreases with ρ for the same number of messages in the system, since the funnels are only restricted by available bandwidth, not by their processing power.

μ is lower than a few hundred milliseconds, we can add μ to duration of our rounds to handle the synchronization gap among the nodes. We can still have a reference global clock which the nodes can synchronize their local clocks with from time-to-time. We only need equivocation protection from that global clock — *Streams* does not depend on that clock for anonymity.

However, suppose some nodes (at most 10% for example) differ by more than few hundred milliseconds from the reference global clock. Such compute nodes can send packets to wrong funnel nodes. If a node receives an onion packet that it is not supposed to receive, the node just forwards the packet to the correct compute node (according to the onion packet header) at the end of the round. Thus, the protocol still functions properly, although the latency needs to be increased based on the amount of such unsynchronized (and compromised) nodes to guarantee the same degree of mixing. With this modified approach, a node does not have to derive in which round to act as a funnel node. For all the onion packets (according to the onion headers) if it is the intended compute node, it acts as a compute node; for all the rest of the packets it acts as a funnel node and forwards them to the next corresponding compute nodes at the end of the round.

Note that the messages that are transmitted by out-of-sync nodes in a round might not mix with other messages. However, it does not impact the mixing of the rest of the messages. In the $\mathcal{F}_{RKR}$-hybrid model, we can model the out-of-sync nodes as compromised parties. If there are at most ψ out-of-sync nodes, and c additional compromised nodes, as a corollary to Theorem 3 we can state:

Theorem 5. *In the $\mathcal{F}_{RKR}$-hybrid model, given a constant integer ρ, a constant fraction $\frac{c}{K}$, and another constant fraction $\frac{\psi}{K}$ such that $\frac{c}{K} + \frac{\psi}{K} < 1$ Streams provides pairwise unlinkability of messages over $\mathfrak{L}$ rounds up to probability δ as in Definition 1, where $\delta < \gamma^{\mathfrak{L}/2}$ and $\gamma = 1 - \frac{\left(\frac{K-c-\psi}{K}\right)^3}{\rho}$.*

For simplicity, the above theorem overapproximates the states for synchronization: either the nodes and in-sync, or they are out-of-sync. A thorough security proof with variable network delays and variable level of synchronization is left for future work.

Note that other protocol with round based communica-

tion models [4], [10], [12], [13], [15], [16], [18] also face similar synchronization challenges. However, because of our loose synchronization, *Streams* is slightly more resilient to such challenges compared to other protocols.

7.2. DoS Attacks Against Funnel Nodes

Although our formal security analysis does not consider DoS attacks by external parties, our design of having few funnel nodes per round introduces such possibilities. If a powerful adversary is able to redirect the DoS attack to the next funnel node within a span of one round, and can keep doing that, the adversary will be able to block the system.

To defend against such attacks, we utilize the fact that each pair of nodes have a persistent TLS connection between them. If a funnel node is under attack, when the compute nodes send packets to the funnel they will not receive any TCP/IP acknowledgments for the dropped packets. In our defense strategy, whenever the compute nodes detect such an attack against the funnel nodes, the compute nodes will shuffle the packets they locally have and directly forward them to the next compute nodes at the end of the round. This will compromise anonymity but provide availability for the system when the funnel nodes are under attack.

If we assume that the adversary has a limited capacity to perform DoS attacks on the funnel nodes, we can consider that only up to ι funnel nodes will be under attack; and the compute nodes will forward to the next compute nodes the messages that are supposed to go to those funnel nodes. Those messages will not have a chance to mix with other messages. If a strong adversary can attack all funnel nodes in succession, *Streams* will effectively be the same Karaoke, but without the noise messages.

Even though, any protocol can be susceptible to external DoS attacks, this attack against funnel nodes is amplified in *Streams* compared to other protocol when an adaptive adversary can quickly redirect their attack to new nodes in every round, because of the small number of funnel nodes. However, if the adversary cannot adapt quickly to redirect the attack, the use of randomness beacons alleviates this problem, since the funnels nodes for the next round are not predictable in the current round. Against a static adversary or slowly adaptive adversary, the funnel nodes in *Streams* are as vulnerable as the compute nodes.

8. Implementation and Evaluation

8.1. Implementation Details

We evaluate our scaling technique by individually evaluating the performance of compute and funnel nodes and then evaluating *Streams*, using a proof-of-concept implementation in Go language (v1.15).[5] The implementation uses the standard *crypto* library in Go, and elliptic curve P25519 from NaCl library to implement cryptographic and sphinx packet operations. We take all measurements by repeating an experiment 10 times and taking an average of those. We make the following system considerations.

Synchronization. For the prototype implementation we consider a global clock that every protocol party (clients and nodes) follows. For real deployments, the global clock can be replaced with local clocks in combination with the idea of loose synchronization technique described in Section 7.1.

About Rounds. In our implementation we choose one second as the duration for each round. This decision is influenced by our scalability goal of processing one million messages every round. We show in Section 8.3 that the round duration determines how many messages can be processed in a round.

Random Shuffle. We implement the Fisher–Yates shuffle [32] to achieve in-memory shuffle of n elements with $\Theta(n)$ computational complexity. This algorithm requires a continual source of randomness — each funnel node uses a locally stored random number table for that purpose.

8.2. Processing Capacity of Compute Nodes

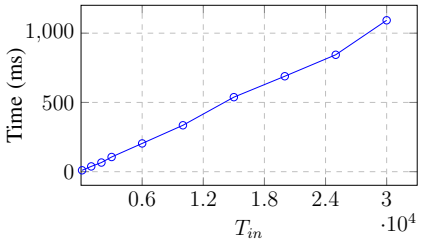
We first evaluate how many onion packets can be processed by a single compute node in a given amount of time, which also reflects how a naive protocol can scale without the divide and funnel technique. To that end, we run a standalone compute node on an AWS t3.2xlarge instance (8 virtual processors), and measure the time spent to process different number of onion packets. Each sphinx packet is constructed with 512 bytes of plaintext and 16 layers of onion encryption. In Figure 15a, we plot a graph between number of onion packets given to a compute node vs. time taken to process those packets. This demonstrates that a compute node (or nodes in a protocol without funnel compute separation) cannot process more than 30000 packets in one second.

8.3. Processing Capacity of Funnel Nodes

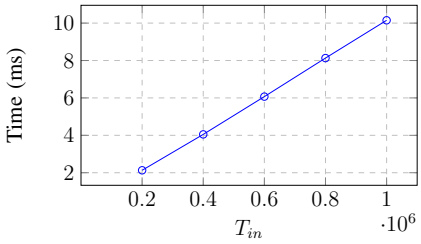
We evaluate how our funnel nodes can scale for different numbers (200K, 400K, ..., 1M) of onion packets in the system even with slow compute nodes. As the number of compute nodes is expandable, funnel nodes dictate the round duration. We measure for funnels how much time is taken by the TLS layer to process different number of packets and how much time is taken to run the shuffle algorithm.

Experimental Setup. To evaluate the performance/scalability of funnel node, we run a standalone funnel node on an AWS t3.2xlarge instance (8 virtual processors), and send varying number of onion packets to that node to measure the time taken for the following two operations: (i) computation time on the TLS layers to process different number of packets; (ii) run the Fisher–Yates shuffle for those packets. Each sphinx packet is constructed with 512 bytes of plaintext and 16 layers of onion encryption, however we do not decrypt any packets during this experiment since we run a standalone funnel.

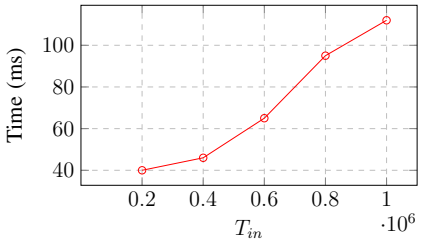
5. The source code is available at: https://github.com/dedas111/protocolX



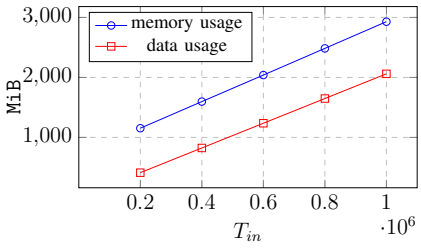
(a) number of packets (T_{in}) sent to a compute node per round (x-axis) vs. time taken in milliseconds (y-axis) for the compute node to process those packets.



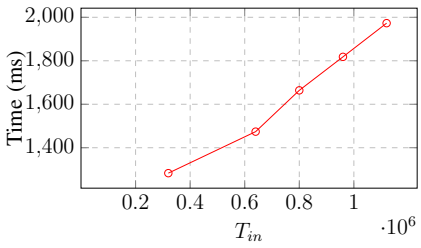
(b) Number of packets (T_{in}) sent to a funnel node (x-axis) vs. the time in milliseconds (y-axis) to run Fisher Yates shuffle for the given number of packets.



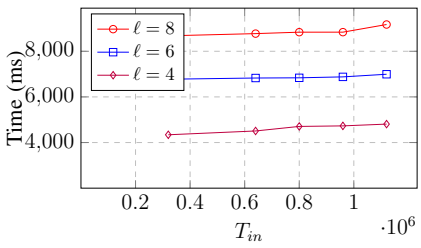
(c) number of packets (T_{in}) sent to a funnel node via TLS connections (x-axis) vs. the time in milliseconds (y-axis) to process those packets (by the TLS layer).



(d) number of packets (T_{in}) sent to a funnel node per round vs. the resources consumed in terms of the maximum memory usage and the amount of data received.



(e) number of packets (T_{in}) sent to a funnel node via TLS connections (x-axis) vs. the time in milliseconds (y-axis) to receive them through network connection.



(f) end-to-end latency plotting number of packets (T_{in}) sent to the protocol (x-axis) vs. the time in milliseconds (y-axis) to deliver the slowest packet for a limit of ℓ hops.

Figure 15: Scalability of *Streams* and performance of subcomponents

Results. We plot our findings in Figure 15b and Fig. 15c. All the measurements are average of 10 runs approximated to the nearest integer. Since we achieve in-memory shuffle using Fisher–Yates algorithm, the observed shuffle time is only around 10 ms even for a million packets (cf. Fig. 15b), even though shuffle algorithm runs in a single thread.

The TLS layer processing of 1M packets takes around 112 milliseconds (compared to 1 second round duration, cf. Fig. 15b). The overhead mainly involves AES encryption/decryption for TLS. These experiments show that the funnel is not restricted by computation.

Memory and Network Overhead. We also measure the memory usage of the process and the throughput requirements (amount of data received by the funnel node over the network) for varying number of packets — we plot them in Fig. 15d. We observe that for 1M packets the memory utilization by the server process remains below 3GB; however, the network throughput requirement can become a bottleneck (receives a total of 2.1GB in messages). If we choose round duration to be 1 second, to handle 1M packets the responsible funnel node requires a burst network capacity of around 17 Gbps.

We additionally demonstrate the practicality of our system by running the following experiment: We run a funnel node on an AWS EC2 c5.18xlarge instance that supports 25 Gbps [33]. We use such a powerful instance only to fulfil the network requirements, we have already shown that funnels are not restricted by computation overhead. Now we send varying number of sphinx packets from another AWS instance such that the next hop is also the sender machine. Then we measure the round-trip time for those packets. If the round-trip-times for all packets are between 1 second to 2 seconds, that means all the packets were received and processed by the funnel within a single round. We plot our results in Fig. 15e, which shows that 1 second round duration is adequate for even 1M total messages in the system — which is more than 30x higher than what a compute node can process.

Many service providers [34], [35] support up to 40 Gbps network speed. Streams nodes do not continuously need a high network capacity, a high burst capacity and a moderate average capacity suffices. If the system needs to support more than 1M messages per round, the servers can set up Multipath-TCP [36], [37] to fulfill the GBps requirement.

8.4. Scalability for *Streams*

To demonstrate that *Streams* can scale to 1M messages, we deploy our prototype of the core protocol of *Streams* (single funnel) using AWS instances distributed over us-east and eu-central regions, we choose EC2 c5.18xlarge instances to satisfy the bandwidth requirement. Then we send varying number of messages in batches and measure how much time it takes for the whole batch to be delivered to the recipients. We simulate sender and recipient by a single AWS EC2 c5.18xlarge instance. In the same batch, we choose same latency (rounds) for every message in the batch for each of execution. We choose the round duration to be 1 second.

If the delay (from the time the first message is sent by the sender to the time the last message is received by the recipient) for a batch is chosen to be X rounds, we expect all the messages to be delivered within X+1 seconds[6], otherwise the funnel nodes are not able to process all the messages within the given round duration.[7]

Results. In Fig. 15f we plot the end-to-end latency of batches of different sizes for chosen delays (number of rounds), the measurements are average of 10 individual instances. In the figure we observe that, when the delay is chosen to be 8 rounds (for example), even for a total number of messages of around 1.1M in a batch, the the total delay does not exceed 9 seconds. It shows that our scaling technique and our protocol can easily scale for 1M messages.

End-to-end Latency. From Fig. 13b we know the number of rounds required for *Streams* to achieve $\delta \leq 2^{-30}$ is 16 — that makes the end-to-end latency 16 seconds for the round duration of 1 second (to scale for 1M messages). Atom is closely related to *Streams* in terms of security guarantees provided and its scalability. However, Atom requires around 28 minutes for their recommended parameters. On the other hand, Atom provides stronger security guarantees ($\delta < 2^{-64}$) than *Streams*. We estimate the end-to-end latency of Atom (in their trap message scenario with no churn) from their measurements [10], for $\delta \leq 2^{-30}$ and $\frac{c}{K} = 10\%$; the estimated end-to-end latency is around 630 seconds — still an order of magnitude higher than that of *Streams*. The setup requirements of Atom are slightly different from *Streams*: Atom requires computationally powerful nodes for a large number of computation-heavy cryptographic operations, but can tolerate some nodes with low bandwidth availability. The computation becomes the bottleneck for Atom with larger messages (they use 160 byte messages in their evaluation), whereas our funnel nodes are mainly restricted by bandwidth requirement.

9. Application Considerations

Other protocols can make use of our scaling technique of splitting the mixing and computing responsibilities to improve their scalability/privacy properties.Below we describe how our scaling technique can be used to improve some example protocols other than *Streams*:

Loopix [9]. Loopix employs multiple paths to scale for many users, which in turn reduces the chance of two messages mixing with each other. Instead Loopix can split the responsibilities in the following way: the randomized delay and mixing of messages happens at a funnel node, while the onion decryption happens at a compute node. This separation of duties would not introduce fixed-length rounds to Loopix, thus allowing Loopix to keep the desired asynchronous model. To keep a comparable level of security as well as the overall structure of the protocol, the paths chosen by clients now include both funnel nodes and compute nodes. In exchange, the separation of duties could drastically reduce the requirement to expand the number parallel paths for Loopix, and hence, could guarantee better mixing.

Karaoke [4]. Since Karaoke already works in rounds, it is easier to adopt our scaling technique. Depending on the number of messages that have to be processed per round, the nodes would choose one or more funnel nodes after the compute phase (e.g., one funnel node per million of messages). If the number of users in the system exceeds several million, the messages can be randomly distributed among a few funnel nodes, e.g., by using a hash function with the message and the random number from the randomness beacon as input. In that case, each honest funnel node will achieve shuffling for the subset of messages it receives. As this subset would be randomly chosen, over log-many rounds pairwise shuffling will occur. As a result, this separation of duties increases the chance of mixing, and it reduces the number of parallel paths. Reducing the number of in parallel paths in turn further improves the required number of round until messages mix, i.e., until mixing can be proven. Additionally, our scaling method will reduce the required number of messages drastically to achieve the same level of link saturation — the noise requirement will reduce from $\Theta(|\text{servers}|^2)$ to $\Theta(|\text{funnels}|^2)$.

Vuvuzela [12]. Vuvuzela employs a single chain of nodes and can directly enjoy the benefits of efficient scaling using our technique. The extension is very similar to how *Streams* employs the scaling technique in this paper: each Vuvuzela node is replaced with a funnel node and many compute nodes. By employing many compute nodes, Vuvuzela can significantly reduce the time required to process packets in a round, and thus reduce the end-to-end latency by a significant factor, or scale for more number of users with similar end-to-end latency.

9.1. Application Scenarios of *Streams*

Our performance analysis clearly demonstrates that *Streams* can scale well with a large number of users. Beyond the traditional mixnet applications such as anonymous e-mailing, we find *Streams* useful for applications such as network-level anonymity for publishing blockchain transactions [38], and anonymous microblogging.

10. Conclusion

In this paper we introduced a scaling technique for AC protocols that horizontally scale (public-key) cryptographic computations while still allowing messages to meet and mix. Our experiments demonstrate that the funnel node is 30x more efficient than a compute node, which clearly demonstrate usefulness of the proposed divide-and-funnel strategy across applications. We demonstrated the applicability of the technique with the protocol *Streams* by scaling it for

6. The total end-to-end latency includes some delay for the packet to be delivered from the last node to the client.

7. In our experiments we send messages in a batch and wait for the batch to complete to easily account for the messages, the design of *Streams* does not require that.

a million messages while keeping the end-to-end latency as low as 16 seconds, and guaranteeing good pairwise unlinkability of messages with 10% compromised nodes in the system. An added advantage over other round-based protocols like Karaoke, XRD, or Atom is that *Streams* (and our scaling technique in general) does not require the users to be synchronized with rounds. Our scaling technique can be leveraged by other protocol designers (demonstrated through the examples of Loopix, Vuvuzela, and Karaoke) to improve scalability and mixing guarantees through network link saturation.

Acknowledgement

We thank Olaf Bernhardt for their help with the implementation. This work is partially supported by the Research Council KU Leuven under the grant C24/18/049, CyberSecurity Research Flanders with reference number VR20192203, Defense Advanced Research Projects Agency (DARPA) under contract number FA8750-19-C-0502, the National Science Foundation under grant CNS-1719196, and the Purdue Research Foundation. Esfandiar Mohammadi was partly supported by the BMBF Project MLens. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any of the funders.

References

[1] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, ser. SSYM'04. USA: USENIX Association, 2004, p. 21.

[2] P. Golle and A. Juels, "Dining cryptographers revisited," in *Proc. of Eurocrypt 2004*, 2004.

[3] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "P2P Mixing and Unlinkable Bitcoin Transactions," in *Proc. 25th Annual Network & Distributed System Security Symposium (NDSS)*, 2017.

[4] D. Lazar, Y. Gilad, and N. Zeldovich, "Karaoke: Distributed private messaging immune to passive traffic analysis," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, Carlsbad, CA, Oct. 2018, pp. 711–725.

[5] S. Eskandarian, H. Corrigan-Gibbs, M. Zaharia, and D. Boneh, "Express: Lowering the cost of metadata-hiding communication with cryptographic privacy," *ArXiv*, vol. abs/1911.09215, 2019.

[6] D. Lu, T. Yurek, S. Kulshreshtha, R. Govind, A. Kate, and A. Miller, "Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 887–903.

[7] I. Abraham, B. Pinkas, and A. Yanai, "Blinder–scalable, robust anonymous committed broadcast," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1233–1252.

[8] N. Alexopoulos, A. Kiayias, R. Talviste, and T. Zacharias, "MCMix: Anonymous Messaging via Secure Multiparty Computation," in *Proceedings of the 26th USENIX Security Symposium*. USENIX Association, 2017, pp. 1217–1234.

[9] A. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis, "The loopix anonymity system," in *Proc. 26th USENIX Security Symposium*, 2017.

[10] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford, "Atom: Horizontally scaling strong anonymity," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17, 2017, p. 406–422.

[11] M. Ando, A. Lysyanskaya, and E. Upfal, "On the complexity of anonymous communication through public networks," *CoRR*, vol. abs/1902.06306, 2019. [Online]. Available: http://arxiv.org/abs/1902.06306

[12] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich, "Vuvuzela: Scalable private messaging resistant to traffic analysis," in *Proc. 25th ACM Symposium on Operating Systems Principles (SOSP 2015)*, 2015.

[13] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich, "Stadium: A distributed metadata-private messaging system," 10 2017, pp. 423–440.

[14] D. Lazar, Y. Gilad, and N. Zeldovich, "Yodel: Strong metadata security for voice calls," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP '19, 2019, p. 211–224.

[15] A. Kwon, D. Lu, and S. Devadas, "Xrd: Scalable messaging system with cryptographic privacy," in *Symposium on Networked Systems Design and Implementation*, 2020.

[16] S. Langowski, S. Servan-Schreiber, and S. Devadas, "Trellis: Robust and scalable metadata-private anonymous broadcast," Cryptology ePrint Archive, Paper 2022/1548, 2022.

[17] M. Ando, A. Lysyanskaya, and E. Upfal, "Practical and Provably Secure Onion Routing," in *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP)*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, pp. 144:1–144:14.

[18] D. Chaum, D. Das, F. Javani, A. Kate, A. Krasnova, J. de Ruiter, and A. T. Sherman, "cmix: Mixing with minimal real-time asymmetric cryptographic operations," in *15th International Conference on Applied Cryptography and Network Security 2017*, 2017.

[19] M. Backes, A. Kate, P. Manoharan, S. Meiser, and E. Mohammadi, "AnoA: A Framework For Analyzing Anonymous Communication Protocols," in *Proc. 26th IEEE Computer Security Foundations Symposium (CSF 2013)*, 2013, pp. 163–178.

[20] L. Barman, I. Dacosta, M. Zamani, E. Zhai, A. Pyrgelis, B. Ford, J. Feigenbaum, and J.-P. Hubaux, "Prifi: Low-latency anonymity for organizational networks," *Proceedings on Privacy Enhancing Technologies*, vol. 2020, pp. 24–47, 10 2020.

[21] H. Corrigan-Gibbs, D. Boneh, and D. Mazières, "Riposte: An anonymous messaging system handling millions of users," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 321–338.

[22] H. Corrigan-Gibbs and B. Ford, "Dissent: Accountable Anonymous Group Messaging," 2010, pp. 340–350.

[23] G. Danezis and I. Goldberg, "Sphinx: A Compact and Provably Secure Mix Format," 2009, pp. 269–282.

[24] D. Das, S. Meiser, E. Mohammadi, and A. Kate, "Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency - choose two," in *2018 IEEE Symposium on Security and Privacy (SP)*, May 2018, pp. 108–126, extended version under https://eprint.iacr.org/2017/954.

[25] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Onion Routing," *Commun. ACM*, vol. 42, no. 2, pp. 39–41, 1999.

[26] C. Kuhn, M. Beck, and T. Strufe, "Breaking and (Partially) Fixing Provably Secure Onion Routing," in *Proceedings of the 41st IEEE Symposium on Security and Privacy*. IEEE, 2020, pp. 168–185.

[27] A. Kate, G. M. Zaverucha, and I. Goldberg, "Pairing-based onion routing with improved forward secrecy," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 4, pp. 1–32, 2010.

[28] R. Canetti, "Universally Composable Security: A New Paradigm for Cryptographic Protocols," 2001, pp. 136–145.

[29] S. Gajek, M. Manulis, O. Pereira, A.-R. Sadeghi, and J. Schwenk, "Universally composable security analysis of tls," in *Provable Security*, J. Baek, F. Bao, K. Chen, and X. Lai, Eds. Springer Berlin Heidelberg, 2008, pp. 313–327.

[30] R. 8446, "The transport layer security (tls) protocol version 1.3," https://tools.ietf.org/html/rfc8446, accessed April 2021.

[31] F. Beato, K. Halunen, and B. Mennink, "Improving the sphinx mix network," in *Cryptology and Network Security*, 2016, pp. 681–691.

[32] M. Eberl, "Fisher-yates shuffle," *Arch. Formal Proofs*, vol. 2016, 2016.

[33] A. AWS, "Amazon EC2 instance network bandwidth," https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-network-bandwidth.html, Accessed January 2021.

[34] H. Electric, "Hurricane Electric internet services," http://he.net/ip_transit.html, Accessed July 2021.

[35] OVHcloud, "Customizable public bandwidth to reach your full potential," https://www.ovhcloud.com/de/bare-metal/bandwidth/, Accessed July 2021.

[36] R. 8684, "Tcp extensions for multipath operation with multiple addresses," https://datatracker.ietf.org/doc/html/rfc8684, accessed April 2021.

[37] B. Hesmans and O. Bonaventure, "Tracing multipath tcp connections," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14, 2014, p. 361–362.

[38] R. Henry, A. Herzberg, and A. Kate, "Blockchain access privacy: Challenges and directions," *IEEE Security Privacy*, vol. 16, no. 4, pp. 38–45, 2018.

[39] F. Shirazi, M. Simeonovski, M. R. Asghar, M. Backes, and C. Díaz, "A survey on routing in anonymous communication protocols." *ACM Comput. Surv.*, vol. 51, no. 3, pp. 51:1–51:39, 2018. [Online]. Available: http://dblp.uni-trier.de/db/journals/csur/csur51.html#ShiraziSABD18

[40] M. Alsabah and I. Goldberg, "Performance and security improvements for tor: A survey," *ACM Comput. Surv.*, vol. 49, no. 2, Sep. 2016. [Online]. Available: https://doi.org/10.1145/2946802

[41] A. Serjantov, R. Dingledine, and P. Syverson, "From a trickle to a flood: Active attacks on several mix types," vol. 2578, 02 2003.

[42] M. Backes, A. Kate, P. Manoharan, S. Meiser, and E. Mohammadi, "AnoA: A Framework For Analyzing Anonymous Communication Protocols," *Journal of Privacy and Confidentiality (JPC)*, vol. 7(2), no. 5, 2016.

[43] A. Kate and I. Goldberg, "Using Sphinx to Improve Onion Routing Circuit Construction," 2010, pp. 359–366.

[44] M. O. Rabin, "Randomized byzantine generals," in *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*. IEEE, 1983, pp. 403–409.

[45] I. T. L. Computer Security Division, "Interoperable randomness beacons: Csrc," accessed April 2021. [Online]. Available: https://csrc.nist.gov/projects/interoperable-randomness-beacons

[46] Drand, "Drand - a distributed randomness beacon daemon," Accessed April 2021. [Online]. Available: https://github.com/drand/drand

[47] A. Bhat, N. Shrestha, A. Kate, and K. Nayak, "Randpiper - reconfiguration-friendly random beacons with quadratic communication," *IACR Cryptol. ePrint Arch.*, no. 1590, 2020. [Online]. Available: https://eprint.iacr.org/2020/1590

[48] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *2017 IEEE Symposium on Security and Privacy (SP)*. Ieee, 2017, pp. 444–460.

[49] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "Hydrand: Practical continuous distributed randomness," in *2020 IEEE Symposium on Security and Privacy (SP). IEEE*, 2020.

[50] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography," *Journal of Cryptology*, vol. 18, no. 3, pp. 219–246, 2005.

[51] T. Hanke, M. Movahedi, and D. Williams, "Dfinity technology overview series, consensus system," *arXiv preprint arXiv:1805.04548*, 2018.

[52] M. Haahr, "True random number service," Accessed April 2021. [Online]. Available: https://www.random.org/

[53] "blockchain oracle service, enabling data-rich smart contracts," https://provable.xyz/, accessed 2021.

[54] "Generate random numbers for smart contracts using chainlink vrf," accessed 2021. [Online]. Available: https://docs.chain.link/docs/chainlink-vrf

Appendix A. Additional Details

A.1. Incompleteness in the Security Analysis of Karaoke

Against passive attackers, Karaoke claims statistical indistinguishability (i.e., negligible δ for $\varepsilon = 0$). [8] Unfortunately, the proof outline of Karaoke is inaccurate. The proof outline identifies bad events and bounds the probability of these events to happen (e.g., with probability of at most 10^{-14}). Then it claims that the views of the adversary conditioned on this bad event not happening are indistinguishable. We found a counterexample that satisfies the condition of those bad events not happening but shows two such conditional views are not identically distributed. Therefore, it is not clear how indistinguishability can hold for Karaoke.

More specifically, we found that the argument of Theorem 3 in their paper [4] is inaccurate; we can observe that from the following counterexample:

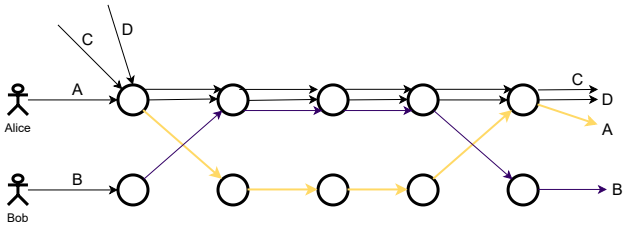


Figure 16: Alice sends message A which meets with noise messages C and D at layer 1, and then diverge from them to the yellow path. Message B meets with noise messages C and D at layer 2, and diverge from them at layer 5.

Counter-example. Consider a scenario (depicted in Fig. 16) where Alice and Bob send messages A and B respectively. Here, C and D are noise messages. One run in which this observation can occur is if Alice sent A and her message went on the yellow path before meeting C and D again at layer 5, while Bob sent B and his message went on

8. Against active adversaries, Karaoke claims to still provide differential privacy guarantees.

the **purple** path. As per theorem 3 in the security analysis (Section 5) of Karaoke: no bad event occurred, so the proof sketch of Karaoke would indicate that the adversary cannot make an educated guess as to whether Alice or Bob might be the sender of A.[9]

This observation, however, can also occur for different choices of paths for Alice's, Bob's, and the noise messages C and D. If we compute the probability for this observation to occur if Alice sends A and Bob sends B and compute the probability of this observation to occur if Alice sends B and Bob sends A, we can see that these probabilities are not equal and not indistinguishable.

Consider that Alice and Bob, as well as the senders of noise messages C and D will choose either the upper or the lower node in each layer with probability 1/2 independent of all other choices they and others make. However, conditioned on the adversary's observation of the traffic pattern, Alice's message could have taken the yellow path with probability $\frac{1}{3}$, and C or D could have taken the yellow path with probability $\frac{2}{3}$. Given that Alice's message has taken the yellow path, the probability that Alice sends A is 1. If we consider the case that C or D has taken the yellow path: given that C or D has taken that path, the probability that Alice sends A is $\frac{1}{2}$, and the probability that Bob sends A is also $\frac{1}{2}$. Overall, conditioned on the observation of the adversary of the specific traffic pattern, the probability that Alice sends A is $\left(\frac{1}{3} \times 1 + \frac{2}{3} \times \frac{1}{2}\right) = \frac{2}{3}$; whereas, the probability that Bob sends A is $\frac{1}{3}$.

From the adversary's point of view (after observing the traffic), this observation is more likely if Alice sent A and Bob B, so an adversary guessing that would have a non-negligible advantage in case this observation occurs. There are many similar cases where an exploitable bias exists and a proof following Karaoke's approach would need to take them all into consideration. The proof of Theorem 3 in the Karaoke paper does not consider this effect from the observation of the traffic. Therefore, their security analysis against global passive adversaries is inaccurate.

A.2. Differential Privacy Unsuited for AC Protocols

Differential Privacy quickly deteriorates in multi-challenge scenarios (also called group privacy), which characterizes that a single person's communication behavior influences more than a single message, as can happen if a person simultaneously uses more than one connection. If a user influences k messages and the protocols satisfies ϵ-DP, the group privacy theorem tells us that for this user $k\epsilon$-DP holds. For a typical epsilon value of 1, a user that influences only 20 messages at once would only achieve 20-DP ($\epsilon = 20$), which results in very weak anonymity guarantees.

Additionally, Approximate Differential Privacy also quantifies the probability δ for the ϵ guarantees to break. Karaoke and Stadium use Approximate Differential Privacy, which is particularly unsuited for massively deployed anonymous communication as it allows for a failure probability δ that is typically around 10^{-5}. If the guarantees can fail with probability 10^{-5} and the protocol is used by 10^6 users, the probability that the guarantees fail for at least one person could be up to $1-(1-10^{-5})^{10^6} = 0.9999546023 \sim 100\%$.

Appendix B. Defense Against Active Attackers

Resiliency of anonymous communication against active attacks is well studied in the literature [39], [40]. We leverage existing techniques to protect packet integrity and to prevent a total loss of anonymity due to packet dropping for *Streams*. Concretely, we use the same strategy as Loopix [9] to defend against active attacks. Our protocol already makes use of the Sphinx [23] packet format, which comes with confidentiality (including padding) and message integrity, and allows for defense against replay attacks and tagging attacks (see below). Additionally, following Loopix, we incorporate messages (called *loop messages*) that users send to themselves to detect and combat packet drops by an active adversary. We consider the following relevant attacks:

(n-1) Attacks [41]. In such attacks, the adversary blocks all but one target message to a node in order to follow the target message. If the adversary decides to drop messages from an honest user Alice, Alice will likely not receive her own loop messages back and she will know that the system is under attack. Alice can then spread the word through some public medium so that other users can stop using the system.

If Alice sends λ loop messages for every real message, the adversary can drop a message from Alice with a probability of $\frac{\lambda}{1+\lambda}$ that Alice will detect the attack (since loop messages are indistinguishable from real messages). Therefore, if the adversary drops k messages from Alice, Alice will detect the attack with probability $1-(\frac{1}{1+\lambda})^k$. As a result, the probability of detection increases drastically with increasing k.

Note that we do not require any specific usage pattern from the users to enable this defense, we only require that they add λ additional messages per real message whenever they are using the system. Additionally, λ can be any constant number, however for our system we conservatively choose $\lambda = 1$. [10]

Replay Attacks. Replay attacks are detected and prevented using the *replay detection tag* implemented in the Sphinx packet header. This tag allows a node to verify if a packet has already been seen or not; if the packet is a replay it is dropped. In our system, *compute* nodes verify the replay detection tag. Since the traffic from a compute node to a funnel is protected by TLS, the funnel is protected from replay attacks if the compute node is honest. If the compute

9. In our example in Figure 16, the noise messages do not originate in Layer 1, while in Karaoke the routers produce such noise messages. The same counterexample would hold, if the top router would generate the noise messages C and D.

10. This value corresponds to roughly four loop messages per other message when compared to Loopix; arguably Loopix' randomized and strictly controlled message sending patterns might confuse some adversaries – we leave these considerations and the exact choice of λ to the system designers.

node is adversarial, the funnel anyway cannot ensure *mixing* for the packets coming from that compute node. The next honest compute node could then figure out whether a replay attack occurs, using the replay detection tags.

Tagging Attacks. The Sphinx packet structure also defends against tagging attacks — if the adversary tries to tag a message, the Sphinx packet verification will fail and the packet will be dropped. If a loop message is dropped, the corresponding user will detect the attack.

Appendix C. Security Analysis (extended)

C.1. More About The Ideal Functionality $\mathcal{F}_{Streams}$

The ideal functionality $\mathcal{F}_{Streams}$ basically acts as as trusted third party to whom users tell that they would like to anonymously send a message. This trusted third party leaks as much information as *Streams* would leak. Due to the regular meeting points at funnel nodes and TLS protection, $\mathcal{F}_{Streams}$ does not need to leak which onion is sent from which compute node to which compute node, as long as the party that sends the onion and the next subsequent funnel node are honest. Removing the very leakage of which onion is sent to whom enables us to prove a strong shuffling property for Theorem 1, pairwise unlinkability with overwhelming probability (see Definition 1).

Formally, the ideal functionality $\mathcal{F}_{Streams}$ provides API calls for when clients want to send a message and they react to network messages. Moreover, as we consider a round-based protocol and the UC-framework is a sequential activation framework (to simplify the analysis), we formally need a "new round" API call.

The ideal functionality $\mathcal{F}_{Streams}$ expects input messages of the form (msg, R, t). As the protocol works in rounds, $\mathcal{F}_{Streams}$ stores the input messages in an input queue. Upon the "new round"-command, each element from the input queue is processed. When processing an input, the ideal functionality $\mathcal{F}_{Streams}$ checks which message only has compromised parties $x_i \notin \mathtt{I_h}$ on its path. For those cases, the ideal functionality leaks the message to the simulator $\mathcal{S}$. Otherwise, $\mathcal{F}_{Streams}$ provides a temporary identifier (in the form of a random integer) to $\mathcal{S}$ in place of a message, along with the segment of the path until the next honest compute node. When the round corresponding to that compute node comes, $\mathcal{F}_{Streams}$ again provides a new temporary identifier along with the next segment of path. When $\mathcal{F}_{Streams}$ switches the temporary identifiers, if the funnel node after the honest compute node is not honest, it provides the mapping between the old and the new identifiers to allow $\mathcal{S}$ to link between packets. Here we slightly over-approximate the leakage by not distinguishing between honest and compromised recipients, because in some protocol setting (anonymous broadcast) or anonymity notion (sender anonymity) the adversary can anyway see the message in plaintext once it comes out of the protocol.

C.2. Pairwise Unlinkability Proof for $\mathcal{F}_{Streams}$

Theorem 6 (Pairwise unlinkability of $\mathcal{F}_{Streams}$)**.** *Given a constant integer ρ, if the amount of compromised nodes is a constant fraction $\frac{\mathsf{c}}{\mathsf{K}} < 1$, $\mathcal{F}_{Streams}$ provides pairwise unlinkability of messages over $\mathfrak{L}$ rounds up to probability δ as in Definition 1, where $\delta < \gamma^{\mathfrak{L}/2}$ with $\gamma = 1 - \frac{\left(\frac{\mathsf{K}-\mathsf{c}}{\mathsf{K}}\right)^3}{\rho}$.*

Proof. Recall that we assume that each node in a round is chosen uniformly at random (funnel nodes by the randomness beacon and compute nodes by the clients) with replacement and independent of choices made in other rounds. Conceptually, a careful strategy where nodes are chosen by avoiding repetition as much as possible can provide better resilience against compromisation. For the ease of analysis, we opt for independence.

If two messages remain in $\mathcal{F}_{Streams}$ for $\mathfrak{L}$ rounds, they are shuffled if both of those two messages have honest compute nodes on their path in some round r, and then an honest node is picked as the common funnel node for both of them in round $r+1$. If that happens, a shuffled list of newly generated temporary identifiers are given to $\mathcal{S}$ on behalf of those messages. We depict this case in Figure 4 (a); the other parts of the figure depict cases in which shuffling does not occur.

Let a be the probability of a randomly picked node being honest; $a = \frac{\mathsf{K}-\mathsf{c}}{\mathsf{K}}$. Since both the funnel node and each compute node are picked uniformly at random (with replacement, and independent of all other nodes), the probability that the funnel node or compute node on the path of a message in any round is compromised is $\frac{\mathsf{c}}{\mathsf{K}} = (1-a)$, and the probability that it is honest is a.

Therefore, the probability that both the messages have honest compute nodes in round r is a^2. The probability that they have the same funnel node in round $(r+1)$ is $\frac{1}{\rho}$. Given that they have the same funnel node in round $(r+1)$, the probability of that funnel node being compromised honest is a. Therefore, the probability that they don't mix in this pair of rounds is $\gamma = \left(1 - \frac{a^3}{\rho}\right)$. If two messages both stay in the system for $\mathfrak{L}$ rounds, the probability that they don't mix is at most $\delta < \left(1 - \frac{a^3}{\rho}\right)^{\frac{\mathfrak{L}}{2}}$.

□

Note that, $\mathfrak{L}$ rounds in the UC-framework version of our protocol translates to $L = \mathfrak{L}/2$ rounds in the original protocol. If $\mathfrak{L}$ in the above theorem is polylogarithmic, δ becomes negligible, which gives us the following corollary.

Corollary 1. *Given a constant fraction $\frac{\mathsf{c}}{\mathsf{K}}$, in the presence of any adversary $\mathcal{S}$, if two arbitrary messages stay together in the protocol $\mathcal{F}_{Streams}$ for $\mathfrak{L} \in \omega(\log \eta)$ rounds they are shuffled with an overwhelming probability.*

C.3. Abstraction Proof for *Streams*

Theorem 7. *For any subprotocol Π_{sub} in the $\mathcal{F}_{RKR}$-hybrid model that UC realizes $\mathcal{F}_{sub}$, the anonymity protocol*

Streams from Section 5.2 using the subprotocol Π_{sub} in the $\mathcal{F}_{CRF}$, $\mathcal{F}_{RKR}$, $\mathcal{F}_{SCS}$, $\mathcal{F}_{round}$-hybrid model UC-realizes $\mathcal{F}_{Streams}$ in the $\mathcal{F}_{round}$-hybrid model.

Recall that formally *Streams* runs in the $\mathcal{F}_{CRF}, \mathcal{F}_{RKR}, \mathcal{F}_{SCS}, \mathcal{F}_{round}$ hybrid model. Our ideal functionality $\mathcal{F}_{Streams}$ absorbs the hybrid functionalities $\mathcal{F}_{CRF}$, $\mathcal{F}_{RKR}$ and $\mathcal{F}_{SCS}$ completely. However, we keep the $\mathcal{F}_{round}$ functionality untouched. We stress that it makes our result stronger that we cast our ideal functionality in the $\mathcal{F}_{round}$-hybrid model. It is straightforward to let $\mathcal{F}_{Streams}$ additionally absorb $\mathcal{F}_{round}$.

Note that, the realization of the ideal functionalities $\mathcal{F}_{RKR}, \mathcal{F}_{SCS}, \mathcal{F}_{CRF}$ in the UC-world translates to secure public-key encryption scheme, secure TLS/SSL, and incorruptible randomness beacon respectively in the real world.

Kuhn et al. [26] show that under standard cryptographic assumptions there is a protocol Π_{sub} in the $\mathcal{F}_{RKR}$-hybrid model that UC realizes $\mathcal{F}_{sub}$. The key idea is to utilize (in a black-box reduction) the UC-realization proof of Π_{sub} such that in the proof the subprotocol's ideal functionality $\mathcal{F}_{sub}$ can be considered. This ideal functionality $\mathcal{F}_{sub}$ is used to abstract away from any cryptographic operations. The second key insight is that the attacker (and the simulator) can perfectly predict how many onions are in the protocol and when each party sends a message. So, only if the recipient is compromised or a message is sent to a client (or the input buffer is full) information is leaked from the protocol. In those cases, the ideal functionality $\mathcal{F}_{Streams}$ indeed leaks information such that the simulator can faithfully (and indistinguishably) simulate the network traffic. We present the full proof below.

Proof. We show the theorem via a series of game hops, starting with the protocol Π and an arbitrary network adversary $\mathcal{A}$. With delta changes in each game, in the final game we end up with the ideal functionality $\mathcal{F}_{Streams}$ and a simulator $\mathcal{S}$. As $\mathcal{F}_{CRF}$ does not send messages to the environment and is not accessible to the environment, it can be easily absorbed by the ideal functionalities. For brevity, we hence neglect it in the subsequent argumentation.

Game 1. *In this game, we consider the original protocol Streams execution with the network attacker $\mathcal{A}$ and the environment $\mathcal{E}$. The protocol follows the code in Figure 7 and Fig. 9.*

Now, we design a game and a protocol where the subprotocols associated with onion processing are replaced with ideal functionality from [26].

Game 2. *Instead of calling the protocol subroutines from Π_{sub}, our protocol Streams now calls the ideal functionality $\mathcal{F}_{sub}$ from Kuhn et al. [26] (for completeness also in the appendix Figure 17). Moreover, the attacker is replaced by a variant of the simulator S_{sub} from Kuhn et al.*

- ***The simulator*** *S^*_{sub} behaves like the simulator S_{sub} in the paper of Kuhn et al. [26] except that acts on one kind of message differently to S_{sub}: if an $\mathcal{F}_{SCS}$ instance sends a message $p := (\texttt{"sent"}, P_i, Map, \mathsf{size})$, where "sent" is a string, P_i is the sender of a packet, Map is the next funnel in the protocol, size is the size of a packet. In that case, S^*_{sub} sends the message p directly to A_d, which is running inside S_{sub}. As $\mathcal{A}_d$ is stateless, these extra messages do not change $\mathcal{A}_d$'s behaviour.*

The protocol Streams still follows the code in Figure 7 and Fig. 9, except that it called $\mathcal{F}_{sub}$ instead of Π_{sub}.

Claim 1. *There is a simulator S_{sub} such that Game 1 is indistinguishable from Game 2.*

Proof of Claim . Analogously to UC's completeness theorem, it suffices to consider the dummy attacker $\mathcal{A}_d$ that forwards all messages to the environment and only acts on the environment's orders.[11] We replace $\mathcal{A}_d$ with a simulator S^*_{sub} that almost behaves like the simulator S_{sub} in the paper of Kuhn et al. [26], which internally runs $\mathcal{A}_d$. S^*_{sub}, however, has to also present an indistinguishable view for $\mathcal{E}$; hence, it has to forward all $\mathcal{F}_{SCS}$ notifications to the environment, just as $\mathcal{A}_d$ would do.

Next, we show that Game 1 (running Π_{sub}, $\mathcal{F}_{RKR}$, and $\mathcal{A}_d$) and Game 2 (running $\mathcal{F}_{sub}$ and S^*_{sub}) are indistinguishable. We show that, if Game 1 is distinguishable from Game 2, then Π_{sub} does not UC realize $\mathcal{F}_{sub}$, which contradicts [26]. $\mathcal{F}_{RKR}$ is faithfully simulated within S_{sub}; hence, it behaves exactly the same in these two interactions.

Towards contradiction, assume that Game 1 is distinguishable from Game 2. Given an environment $\mathcal{E}$ that can distinguish Game 1 from Game 2, we construct an environment $\mathcal{E}_{sub}$ that can distinguish Π_{sub} and $\mathcal{F}_{RKR}$ interacting with the dummy attacker $\mathcal{A}_d$ from $\mathcal{F}_{sub}$ interacting with S_{sub}. $\mathcal{E}_{sub}$ internally runs $\mathcal{E}$. $\mathcal{E}_{sub}$ has to ensure that $\mathcal{E}$ believes that it is in Game 1 or Game 2, respectively. Hence, $\mathcal{E}_{sub}$ has to ensure that $\mathcal{E}$ gets the same messages as in Game 1 and Game 2, respectively. So, we have to make sure that $\mathcal{E}$ sees the same the funnel-protocol communication and the notification messages from $\mathcal{F}_{SCS}$ for each packet that are handed through from $\mathcal{A}_d$ in Game 1. The funnel-protocol communication can be achieved by $\mathcal{E}_{sub}$ running the funnel-protocol instances. In Game 1 and Game 2, the $\mathcal{F}_{SCS}$ notification messages are sent whenever a funnel or a compute node instance communicates with $\mathcal{F}_{round}$. Hence, $\mathcal{E}_{sub}$ has to ensure that $\mathcal{E}$ gets these notification messages at the correct time, which can do as it internally runs $\mathcal{F}_{SCS}$.

- **The environment** $\mathcal{E}_{sub}$ internally runs $\mathcal{F}_{SCS}$, $\mathcal{F}_{round}$, and $\mathcal{E}$. Let $Int_{1,sub}$ be the interaction between Π_{sub} and $\mathcal{F}_{RKR}$ from [26] and the dummy attacker $\mathcal{A}_d$ with an environment (in our case $\mathcal{E}_{sub}$), and let $Int_{2,sub}$ be the interaction between $\mathcal{F}_{sub}$ and the simulator S_{sub} from [26]. As $\mathcal{E}_{sub}$ does not know whether it is interacting with $Int_{1,sub}$ or $Int_{2,sub}$, we describe its behavior agnostic to $b = 1$ or $b = 2$ with $Int_{b,sub}$.

– Upon receiving a message a party from $Int_{b,sub}$ (i.e., from a Π_{sub} instance or $\mathcal{F}_{sub}$), run Π_{client} and forward the response to $\mathcal{E}$.

11. For any other attacker $\mathcal{A}$ and each environment $\mathcal{E}$, there is an environment $\mathcal{E}'$ that internally emulates the interaction between $\mathcal{E}$ and $\mathcal{A}$. $\mathcal{E}'$ interacts with the dummy attacker $\mathcal{A}_d$ and produces the same view.

– Upon receiving a message over the network from $Int_{b,sub}$ (i.e., from $\mathcal{A}_d$ or S_{sub}), forward the message to $\mathcal{F}_{SCS}$ and faithfully (as in Game 1) compute the interaction between $\mathcal{F}_{round}$, the funnel instances, and $\mathcal{E}$.
– Upon receiving a message from $\mathcal{E}$ for the network attacker, directly forward this message to the network attacker in $Int_{b,sub}$ (i.e., to $\mathcal{A}_d$ or $\mathtt{S}_{sub}$).
  * Upon receiving a notification message from the internally emulated $\mathcal{F}_{SCS}$, forward it to $\mathcal{E}$. (We stress that S^*_{sub} is split into this interaction and the part that is run in Int_{sub}.)

For each $b \in \{1,2\}$, we have to show that for $\mathcal{E}$ the interaction within $\mathcal{E}_{sub}$, which in turn interacts with $Int_{b,sub}$, is indistinguishable from the interaction with Game b. For $b = 1$, the interaction within $\mathcal{E}_{sub}$ solely differs in the order in which the notification message from $\mathcal{F}_{SCS}$ arrives. As these messages first reach $\mathcal{E}_{sub}$ before reaching $\mathcal{E}$, $\mathcal{E}_{sub}$ can successfully reverse the order again (see above) and constructs a perfect view for $\mathcal{E}$.

For $b = 2$, $\mathcal{E}_{sub}$ internally emulates Game 1 (except for Π_{sub}). We show that for $b = 2$ nevertheless the view of $\mathcal{E}$ when being emulated within $\mathcal{E}_{sub}$ is indistinguishable from the view when interacting in Game 2. Recall that the only difference between Game 1 and Game 2 is that Π_{sub} is replaced by $\mathcal{F}_{sub}$, and $\mathcal{A}_d$ is replaced by $\mathtt{S}_{sub}$. As $\mathcal{F}_{sub}$ is changed by $\mathcal{E}_{sub}$, it suffices to analyze whether the message transcript to $\mathtt{S}_{sub}$ is indistinguishable for $\mathtt{S}_{sub}$ and whether the transcript from $\mathtt{S}_{sub}$ (through $\mathcal{E}_{sub}$) is indistinguishable for $\mathcal{E}$.

Whenever by $Int_{2,sub}$ a message is sent by $\mathtt{S}_{sub}$ to $\mathcal{E}_{sub}$, this messages first goes through the internally emulated instances of the $\mathcal{F}_{SCS}, \mathcal{F}_{round}$, and $\Pi_{\textit{funnel}}$ protocols. These protocols solely forward messages, and of these only $\mathcal{F}_{SCS}$ sends a notification to the network attacker $\mathcal{A}_d$. In this case, as defined above, $\mathcal{E}_{sub}$ directly forwards the notification to $\mathcal{E}$; this is exactly what would happen in Game 2. All other messages are forwarded and, as in Game 2, potentially sent to $\mathcal{E}$. Hence, whenever in Game 2 a message is sent to $\mathcal{E}$ also in $\mathcal{E}_{sub}$'s internal emulation (if $b = 2$) a message is sent to $\mathcal{E}$.

Next, we consider the case where for $b = 2$ a message is sent by $\mathcal{E}$ (while being internally emulated by $\mathcal{E}_{sub}$) to the network attacker, which would in Game 2 be $\mathtt{S}^*_{sub}$. As defined above, in this case, $\mathcal{E}_{sub}$ sends the message directly to the network attacker in $Int_{b,sub}$. As $b = 2$, the network attacker is $\mathtt{S}_{sub}$. Hence, the message transcript (from $\mathtt{S}_{sub}$'s point of view) is exactly the same as in Game 2.

If Int_{sub} is the interaction with Π_{sub}, $\mathcal{F}_{RKR}$, and $\mathcal{A}_d$, $\mathcal{E}_{sub}$ ensures that $\mathcal{E}$ has exactly the same view as in Game 1. If Int_{sub} is the interaction with $\mathcal{F}_{sub}$ and S_{sub}, $\mathcal{E}_{sub}$ ensures that $\mathcal{E}$ has exactly the same view as in Game 2. Hence, by assumption, with the translation of $\mathcal{E}_{sub}$ the submachine $\mathcal{E}$ can distinguish the interaction with Π_{sub}, $\mathcal{F}_{RKR}$, and $\mathcal{A}_d$ from the interaction with $\mathcal{F}_{sub}$ and S_{sub}.

For any poly-bounded $\mathcal{E}$, $\mathcal{E}_{sub}$ acts as a poly-bounded environment in the UC game. Yet, Kuhn et al. [26] proved that there is no poly-bounded environment that can distinguish these two interactions, which is a contradiction. Hence, Game 1 and Game 2 are indistinguishable. ◇

Game 3. *We replace $\Pi_{\textit{worker}}$, $\Pi_{\textit{client}}$, $\mathcal{F}_{SCS}$, $\mathcal{F}_{sub}$ with the ideal functionality $\mathcal{F}_{\textit{Streams}}$. The simulator S^*_{sub} is replaced by a simulator S_f. The simulator S_f internally runs S^*_{sub} but translates the format of the output of $\mathcal{F}_{\textit{Streams}}$ to the format output by $\mathcal{F}_{sub}$. We stress that as we are in the hybrid $\mathcal{F}_{round}$-model, $\mathcal{F}_{round}$ remains in the ideal world as it was in the previous games.*

Claim 2. *With the simulator S_f, Game 3 is indistinguishable from Game 2.*

Proof of Claim . For the analysis, we divide the execution in overlapping sub-sequences of the form compute node, funnel, compute node (overlapping at the last funnel). For those sub-sequences where the funnel is malicious or the first compute node is malicious, $\mathcal{F}_{\textit{Streams}}$ has exactly the same leakage as $\mathcal{F}_{sub}$, except that the format of the leakage is translated. If one of the funnels is honest and the first compute nodes is honest, though, $\mathcal{F}_{\textit{Streams}}$, in contrast to $\mathcal{F}_{sub}$, does not leak which compute node sends (the ideal abstraction of) an onion to which other compute node. Next, we argue that this leakage is also hidden in Game 2, as $\mathcal{F}_{SCS}$ and the funnels hide this information.

As the first compute node is honest, it does not leak to the network attacker to whom the onion is sent. If the first funnel is honest, it does not leak the link between the two compute nodes to the network attacker. As $\mathcal{F}_{SCS}$ only notifies the network attacker that some messages was sent and as the funnels shuffle the messages received by them in a round, the network attacker does not learn by whom an onion was sent. ◇

Therefore, for simulator $\mathcal{S}$ our protocol *Streams* UC-realizes the ideal functionality $\mathcal{F}_{\textit{Streams}}$. □

C.4. Pairwise Unlinkability for Layman's Protocol

We also want to analyze the scenario where we do not need to scale horizontally, i.e, protocol described in Section 3.2 is sufficient (e.g., the nodes are as powerful as network routers or the total number of users is less than few thousands).

Theorem 8 (Pairwise unlinkability of the layman's protocol)**.** *For any subprotocol Π_{sub} in the $\mathcal{F}_{RKR}$-hybrid model that UC realizes $\mathcal{F}_{sub}$, given a constant fraction $\frac{\mathsf{c}}{\mathsf{K}} < 1$, the layman's protocol (using Π_{sub}) described in Section 3.2 provides pairwise unlinkability of messages over $\mathfrak{L}$ rounds up to probability δ as in Definition 1, where $\delta < \left(\frac{\mathsf{c}}{\mathsf{K}}\right)^{\mathfrak{L}}$.*

Proof Sketch. We skip the detailed proof as the proof methodology is very similar to the proof with compute and funnel nodes. The UC proof becomes much easier if we do not have to distinguish between compute and funnel nodes. There is one key difference for the combinatorial argument:

instead of the funnel node in the funnel round and the two compute nodes in the immediate next compute round, there is only one node and just one round.

Therefore, instead of representing each pair of rounds with three coin tosses in the compute and funnel node scenario, we have exactly one round with one coin toss for the layman's protocol with success probability $a = \frac{\mathsf{c}}{\mathsf{K}}$. Hence, we can define an ideal functionality $\mathcal{F}_{core}$ as described in **??**, which shuffles the messages in the system whenever they encounter an honest node on the path. To provide a simulator for $\mathcal{F}_{core}$ we can use the exact same simulator $\mathcal{S}_{sub}$ as the one we use in the proof of Theorem 2, with only one minor modification: $\mathcal{S}_{sub}$ directly forwards all the network messages to the round functionality.

The probability of **not** finding an honest node in a path of length $\mathfrak{L}$ is upper bounded by $\delta \leq \left(\frac{\mathsf{c}}{\mathsf{K}}\right)^{\mathfrak{L}}$. Therefore, if two arbitrary messages stay in the protocol for at least $\mathfrak{L}$ rounds, they are shuffled with probability at least $1-\delta$. □

The above theorem also gives us an important insight about how much security is degraded to achieve scalability through our funnel and compute nodes. Note that the number of rounds $\mathfrak{L}$ in the theorem translates to $\mathfrak{L}$ rounds in original layman's protocol described in Section 3.2 as well, (unlike our Streams protocol) since there is no separate funnel and compute phase.

Appendix D. Pairwise Unlinkability and And Related Anonymity Notions

Our notion of pairwise unlinkability is conceptually closely related to *tail indistinguishability* by Kuhn et al. [26]. The main difference is that in their definition packets are required to meet in a node that processes them cryptographically. Since our nodes in a path are split into compute nodes and funnel nodes, our notion of pairwise unlinkability is not tied to packets that are assumed to meet, but covers all packets. In this sense pairwise unlinkability follows a previous notion of unlinkability of Kate et al. [27], but extends it with the explicit time when a message enters and leaves the system.

D.1. Sender anonymity

The common anonymity notion *sender anonymity* states that the recipient of a message cannot distinguish whether the message originated in one sender over another sender, even for a pair of potential senders of the adversary's choice. This closely resembles pairwise unlinkability with one key difference: sender anonymity typically talks about a single challenge message, not about a pair of messages. Below we provide the game description and definition of sender anonymity (which is an adaptation of the definition from AnoA [42]).

The Sender Anonymity game $\mathcal{G}_{\mathsf{SA}}^{\Pi,\mathcal{A},\mathsf{c}}(1^\eta)$ for protocol Π against adversary $\mathcal{A}$ can be described as follows:

- The challenger `Ch` provides the adversary $\mathcal{A}$ with the description of Π (that includes the description of the sets $\mathcal{S}$, $\mathcal{R}$, and I).
- $\mathcal{A}$ statically corrupts all recipients in $\mathcal{R}$, all senders in $\mathcal{S}$ except from a pair u_0, u_1, and a subset of I denoted by I_{corr}, such that $|I_{\mathsf{corr}}| \leq \mathsf{c}$ (i.e., no more than c nodes are corrupted).
- `Ch` and $\mathcal{A}$ engage in an execution of Π where `Ch` acts on behalf of u_0, u_1 and the honest nodes, while $\mathcal{A}$ controls the corrupted parties and monitors the network traffic as a global passive adversary.
- At any time t', $\mathcal{A}$ sends the challenge pair $(u_0, m_0, t_{s,0}, R_0)$ and $(u_1, \_, \_, \_)$ to `Ch` where u is the sender of the message, m the content, t_s the time the message enters the system, and R the receiver of the message.
- In turn, `Ch` chooses a random bit $b \in \{0,1\}$ and initiates the challenge transmissions according to the following cases:
  - If $b = 0$, Π transmits the message $(u_0, m_0, t_{s,0}, R_0)$.
  - If $b = 1$, Π transmits the messages $(u_1, m_0, t_{s,0}, R_0)$.
- $\mathcal{A}$ can terminate the game any time by outputting a bit b^*, as a guess for the challenge bit b.
- The game returns 1 if and only if $b^* = b$ (i.e., $\mathcal{A}$ guesses correctly), otherwise the game returns 0.

Definition 2 (Sender Anonymity)**.** *A protocol Π provides* sender anonymity *of messages for c compromise up to probability δ for $0 \leq \delta < 1$ if, for all probabilistic polynomial time (PPT) adversaries $\mathcal{A}$ passively and statically compromising at most c nodes, the following holds:*

$$\Pr\left[\mathcal{G}_{\mathsf{SA}}^{\Pi,\mathcal{A},\mathsf{c},t}(1^\eta) = 1\right] - \Pr\left[\mathcal{G}_{\mathsf{SA}}^{\Pi,\mathcal{A},\mathsf{c},t}(1^\eta) = 0\right] \leq \delta(\eta).$$

If different clients can send messages at different times, there is an inherent leakage that can break sender anonymity. If, say, the adversary observes Alice sending a message in round t and Bob sending a message in round $t+2$, the arrival time of the challenge message together with the distribution of the latency might tell the adversary who of them is more likely to have sent the challenge message. In the simplest example, for a constant latency, the adversary could immediately exclude one of them from being the challenge sender. Note that this apparent attack is independent of how a protocol achieves anonymity and even applies if the messages are kept in a trusted third party for the same amount of time.

The definition of pairwise unlinkability aims to avoid dealing with such client-dependent aspects of anonymity and measures the core mixing property of the protocol. However, if a protocol follows batch processing (as in Karaoke and Atom), pairwise unlinkability immediately translates to sender anonymity — that relationship can be captured by the following lemma.

Lemma 2. *Let us assume for a protocol Π that all the senders (including u_{1-b}) send their messages at time t_s, and the messages can stay exactly t rounds in the protocol, and are delivered to the designated recipients at time $t_f =$*

$t_s + t$. *If the protocol* Π *provides pairwise unlinkability of messages over* t *rounds up to probability* δ *as in Definition 1, it also provides sender anonymity up to probability* δ *as in Definition 2.*

Proof Sketch. We prove the above lemma by showing that, if there exist an adversary $\mathcal{A}_{\mathsf{SA}}$ that wins the sender anonymity game $\mathcal{G}_{\mathsf{SA}}^{\Pi,\mathcal{A},\mathsf{c}}(1^\eta)$ against protocol Π, we can construct an adversary $\mathcal{A}_{\mathsf{PU}}$ that wins the pairwise unlinkability game $\mathcal{G}_{\mathsf{PU}}^{\Pi,\mathcal{A},\mathsf{c},t}(1^\eta)$.

$\mathcal{A}_{\mathsf{PU}}$ acts as the challenger for $\mathcal{G}_{\mathsf{SA}}^{\Pi,\mathcal{A},\mathsf{c}}$ against $\mathcal{A}_{\mathsf{SA}}$. In the game $\mathcal{G}_{\mathsf{PU}}^{\Pi,\mathcal{A},\mathsf{c},t}$, $\mathcal{A}_{\mathsf{PU}}$ follows the exact same steps as $\mathcal{A}_{\mathsf{SA}}$ in $\mathcal{G}_{\mathsf{SA}}^{\Pi,\mathcal{A},\mathsf{c}}$, until $\mathcal{A}_{\mathsf{SA}}$ sends the challenge pairs. $\mathcal{A}_{\mathsf{PU}}$ forwards all the transcripts from $\mathcal{G}_{\mathsf{PU}}^{\Pi,\mathcal{A},\mathsf{c},t}$ to $\mathcal{A}_{\mathsf{SA}}$.

When $\mathcal{A}_{\mathsf{SA}}$ sends the challenge pair (u_0, m_0, t_s, R_0) and $(u_1, \_, \_, \_)$, $\mathcal{A}_{\mathsf{PU}}$ uses the challenge pair $(u_0, m_0, t_s, t_f, R_0)$ and $(u_1, m_1, t_s, t_f, R_1)$, where m_1 is a randomly generated message and $m_1 \neq m_0$. Note that t_s and t_f are restricted by the assumption of the lemma.

When $\mathcal{A}_{\mathsf{SA}}$ terminates $\mathcal{G}_{\mathsf{SA}}^{\Pi,\mathcal{A},\mathsf{c}}$, $\mathcal{A}_{\mathsf{PU}}$ also terminates $\mathcal{G}_{\mathsf{PU}}^{\Pi,\mathcal{A},\mathsf{c},t}$. $\mathcal{A}_{\mathsf{PU}}$ returns the same guess for b as $\mathcal{A}_{\mathsf{SA}}$.

The view of $\mathcal{A}_{\mathsf{RA}}$ is translated as is from the view of $\mathcal{A}_{\mathsf{PU}}$. Therefore, if $\mathcal{A}_{\mathsf{SA}}$ has guessed correctly, $\mathcal{A}_{\mathsf{PU}}$ also guesses correctly. □

Note that the reverse direction is also true (sender anonymity implies pairwise unlinkability when t_s and t_f are same for all messages), and can be proven analogously.

D.2. Relationship anonymity

Relationship anonymity states that if two senders send one message each to two receivers, a third party is unable to determine which sender talks to which receiver significantly better than purely guessing. Loopix [9] calls this property *Sender-Receiver Third-party Unlinkability*. Given that the two messages in question are sent in the same round and that both senders choose a sufficiently large latency from the same distribution, pairwise unlinkability immediately implies this anonymity property. Below we provide the game description and definition for relationship anonymity adapted from AnoA [42].

The Relationship Anonymity game $\mathcal{G}_{\mathsf{RA}}^{\Pi,\mathcal{A},\mathsf{c}}(1^\eta)$ for protocol Π against adversary $\mathcal{A}$ can be described as follows:

- The challenger `Ch` provides the adversary $\mathcal{A}$ with the description of Π (that includes the description of the sets $\mathcal{S}$, $\mathcal{R}$, and I).
- $\mathcal{A}$ statically corrupts all recipients in $\mathcal{R}$ except from a pair R_0, R_1, all senders in $\mathcal{S}$ except from a pair u_0, u_1, and a subset of I denoted by I_{corr}, such that $|I_{\mathsf{corr}}| \leq \mathsf{c}$ (i.e., no more than c nodes are corrupted).
- `Ch` and $\mathcal{A}$ engage in an execution of Π where `Ch` acts on behalf of u_0, u_1 and the honest nodes, while $\mathcal{A}$ controls the corrupted parties and monitors the network traffic as a global passive adversary.
- At any time t', $\mathcal{A}$ sends a pair of challenge messages $(u_0, m_0, t_{s,0}, R_0)$ and $(u_1, m_1, \_, R_1)$ to `Ch` where u is the sender of the message, m the content, t_s the time the message enters the system, and R the receiver of the message.
- In turn, `Ch` chooses a random bit $b \in \{0, 1\}$ and initiates the challenge transmissions according to the following cases:
  - If $b = 0$, Π transmits the messages $(u_0, m_0, t_{s,0}, R_0)$ and $(u_1, m_1, t_{s,0}, R_1)$.
  - If $b = 1$, Π transmits the message $(u_0, m_0, t_{s,0}, R_1)$ and $(u_1, m_1, t_{s,0}, R_0)$
- $\mathcal{A}$ can terminate the game any time by outputting a bit b^*, as a guess for the challenge bit b.
- The game returns 1 if and only if $b^* = b$ (i.e., $\mathcal{A}$ guesses correctly), otherwise the game returns 0.

Definition 3 (Relationship Anonymity)**.** *A protocol* Π *provides* relationship anonymity *of messages for* c *compromise up to probability* δ *for* $0 \leq \delta < 1$ *if, for all probabilistic polynomial time (PPT) adversaries* $\mathcal{A}$ *passively and statically compromising at most* c *nodes, the following holds:*

$$\Pr\left[\mathcal{G}_{\mathsf{RA}}^{\Pi,\mathcal{A},\mathsf{c},t}(1^\eta) = 1\right] - \Pr\left[\mathcal{G}_{\mathsf{RA}}^{\Pi,\mathcal{A},\mathsf{c},t}(1^\eta) = 0\right] \leq \delta(\eta).$$

Lemma 3. *Let us assume for a protocol* Π *that all the senders (including* u_{1-b}*) send their messages at time* t_s*, and the messages can stay exactly* t *rounds in the protocol, and are delivered to the designated recipients at time* $t_f = t_s + t$*. If the protocol* Π *provides pairwise unlinkability of messages over* t *rounds up to probability* δ *as in Definition 1, it also provides relationship anonymity up to probability* δ *as in Definition 3.*

Proof Sketch. We prove the above lemma by showing that, if there exist an adversary $\mathcal{A}_{\mathsf{RA}}$ that wins the relationship anonymity game $\mathcal{G}_{\mathsf{RA}}^{\Pi,\mathcal{A},\mathsf{c}}(1^\eta)$ against protocol Π, we can construct an adversary $\mathcal{A}_{\mathsf{PU}}$ that wins the pairwise unlinkability game $\mathcal{G}_{\mathsf{PU}}^{\Pi,\mathcal{A},\mathsf{c},t}(1^\eta)$.

$\mathcal{A}_{\mathsf{PU}}$ acts as the challenger for $\mathcal{G}_{\mathsf{RA}}^{\Pi,\mathcal{A},\mathsf{c}}$ against $\mathcal{A}_{\mathsf{RA}}$. In the game $\mathcal{G}_{\mathsf{PU}}^{\Pi,\mathcal{A},\mathsf{c},t}$, $\mathcal{A}_{\mathsf{PU}}$ follows the exact same steps as $\mathcal{A}_{\mathsf{RA}}$ in $\mathcal{G}_{\mathsf{SA}}^{\Pi,\mathcal{A},\mathsf{c}}$, until $\mathcal{A}_{\mathsf{SA}}$ sends the challenge pairs.

When $\mathcal{A}_{\mathsf{RA}}$ sends the challenge pair (u_0, m_0, t_s, R_0) and $(u_1, m_1, \_, R_1)$, $\mathcal{A}_{\mathsf{PU}}$ uses the challenge pair $(u_0, m_0, t_s, t_f, R_0)$ and $(u_1, m_1, t_s, t_f, R_1)$ for $\mathcal{G}_{\mathsf{PU}}^{\Pi,\mathcal{A},\mathsf{c},t}$. Note that t_s and t_f are restricted by the assumption of the lemma.

In $\mathcal{G}_{\mathsf{RA}}^{\Pi,\mathcal{A},\mathsf{c}}(1^\eta)$, R_0 and R_1 are honest, and therefore, do not reveal the contents of the messages they receive. Accordingly, $\mathcal{A}_{\mathsf{PU}}$ forwards all the transcripts from $\mathcal{G}_{\mathsf{PU}}^{\Pi,\mathcal{A},\mathsf{c},t}$ to $\mathcal{A}_{\mathsf{SA}}$, except the contents of the messages received by R_0 and R_1. When $\mathcal{A}_{\mathsf{RA}}$ terminates $\mathcal{G}_{\mathsf{RA}}^{\Pi,\mathcal{A},\mathsf{c}}$, $\mathcal{A}_{\mathsf{PU}}$ also terminates $\mathcal{G}_{\mathsf{PU}}^{\Pi,\mathcal{A},\mathsf{c},t}$. $\mathcal{A}_{\mathsf{PU}}$ returns the same guess for b as $\mathcal{A}_{\mathsf{SA}}$.

The view of $\mathcal{A}_{\mathsf{RA}}$ is directly translated from the view of $\mathcal{A}_{\mathsf{PU}}$, except $\mathcal{A}_{\mathsf{RA}}$ cannot see the contents of the messages received by R_0 and R_1. Therefore, if $\mathcal{A}_{\mathsf{RA}}$ has guessed correctly, $\mathcal{A}_{\mathsf{PU}}$ also guesses correctly. □

Note that the reverse direction is not true — relationship anonymity does not imply pairwise unlinkability when t_s and t_f are same for all messages. Intuitively, the adversary in the relationship anonymity game cannot observe

the message contents received by the honest recipients, and therefore, has less insights compared to the pairwise unlinkability and sender anonymity games.

D.3. Pairwise properties and more than two parties

Pairwise unlinkability for all pairs of messages is a strong property that holds for all pairs of messages at the same time. The property also naturally extends to more than two messages. Practically, distinguishing between two cases where more than two messages are different/swapped is often easier than just distinguishing a single pair. Formally, we can bound the advantage by increasing δ. Consider a case where there are three messages $(u_0, m_0, t_{s,0}, t_{f,0}, R_0)$, $(u_1, m_1, t_{s,1}, t_{f,1}, R_1)$, and $(u_2, m_2, t_{s,2}, t_{f,2}, R_2)$ with $\min(t_{f_0}, t_{f_1}, t_{f_2}) - \max(t_{s_0}, t_{s_1}, t_{s_2}) \geq t$. We know that the adversary cannot distinguish this case from $(u_1, m_0, t_{s,0}, t_{f,0}, R_0)$, $(u_0, m_1, t_{s,1}, t_{f,1}, R_1)$, and $(u_2, m_2, t_{s,2}, t_{f,2}, R_2)$ with an advantage greater than δ. Moreover, we know that the adversary cannot distinguish that case from $(u_2, m_0, t_{s,0}, t_{f,0}, R_0)$, $(u_0, m_1, t_{s,1}, t_{f,1}, R_1)$, and $(u_1, m_2, t_{s,2}, t_{f,2}, R_2)$ with an advantage greater than δ. Thus, the adversary cannot be able to distinguish the first and last of those cases with advantage greater than $2 \cdot \delta$. This argument extends naturally to any larger set of messages that are being swapped around simultaneously.

Appendix E. Existing functionalities

E.1. Ideal Functionality for Onion Routing

We borrow the ideal functionality $\mathcal{F}_{sub}$ for onion routing from the work of Kuhn et al. [26, Algorithm 1]. We present the ideal functionality in Figure 17 for completeness. Kuhn et al. [26, Appendix E] also presents a modified version of Sphinx [43] that realizes the ideal functionality $\mathcal{F}_{sub}$. We use the same modified version of Sphinx as our Π_{sub} in the current work. We aim for anonymous broadcast to the network. In our ideal functionality, messages from our last node are sent to the environment instead of delivering them to an explicit receiver. The delivery of messages occurs through the environment which controls the network functionality.

E.2. Secure Communications Sessions

We use the ideal functionality $\mathcal{F}_{scs}$ from the work of Gajek et al. [29, Figure 4] to realize secure communications sessions. Their work shows that the TLS protocol [29, Figure 5] UC-realizes the ideal functionality $\mathcal{F}_{scs}$.

E.3. Randomness Beacons

We assume that each protocol party (including the adversary) has access to an incorruptible randomness beacon. In particular, future values of this beacon are not known to the adversary.

A randomness beacon [44] emits a new *random* value at intermittent intervals such that the emitted values are bias-resistant, i.e., no entity can influence a future beacon value, and unpredictable, i.e., no entity can predict future beacon value. NIST's Randomness Beacons project [45] and the emerging Drand Organization [46] are two prominent ready to use Internet-based instantiations of randomness beacon, while several other protocols [47]–[51] and implementations [52]–[54] are also available.

To focus on the building blocks that we provide, we abstract away from the cryptographic details of those constructions and assume such an ideal randomness beacon. It outputs each time a ℓ-long substring of an infinite random string beacon; using that ℓ-length string a protocol party can derive the funnel nodes for the next ℓ rounds. Formally, we only require the randomness beacon to be unpredictable before the protocol starts, as our adversary can only statically compromise parties; the beacon, however, can also be leveraged for resistance against dynamic corruption.

Data structure:
Bad: Set of Corrupted Nodes
L: List of Onions processed by adversarial nodes
B_i: List of Onions held by node P_i
// Notation:
// $\mathcal{S}$: Adversary (resp. Simulator)
// $\mathcal{Z}$: Environment
// $\mathcal{P} = (P_{o_1}, \ldots, P_{o_n})$: Onion path
// $O = (sid, P_s, P_r, m, n, \mathcal{P}, i)$: Onion = (session ID, sender, receiver, message, path length, path, traveled distance)
// N: Maximal onion path length

On message Process_New_Onion$(P_r, m, n, \mathcal{P})$ *from* P_s
- // P_s creates and sends a new onion (either instructed by $\mathcal{Z}$ if honest or $\mathcal{S}$ if corrupted)
- **if** $|\mathcal{P}| > N$; // selected path too long
- **then**
  - Reject
- **else**
  - $sid \leftarrow^R$ session ID ; // pick random session ID
  - $O \leftarrow (sid, P_s, P_r, m, n, \mathcal{P}, 0)$; // create new onion
  - Output_Corrupt_Sender$(P_s, sid, P_r, m, n, \mathcal{P}, \text{start})$
  - Process_Next_Step(O)

Procedure Output_Corrupt_Sender$(P_s, sid, P_r, m, n, \mathcal{P}, temp)$
- // Give all information about onion to adversary if sender is corrupt
- **if** $P_s \in$ *Bad* **then**
  - Send "$temp$ belongs to onion from P_s with $sid, P_r, m, n, \mathcal{P}$" to $\mathcal{S}$

Procedure Process_Next_Step$(O = (sid, P_s, P_r, m, n, \mathcal{P}, i))$
- // Router P_{o_i} just processed O that is now passed to router $P_{o_{i+1}}$
- **if** $P_{o_j} \in$ *Bad for all* $j > i$ **then**
  - Send "Onion from P_{o_i} with message m for P_r routed through $(P_{o_{i+1}}, \ldots, P_{o_n})$" to $\mathcal{S}$
  - Output_Corrupt_Sender$(P_s, sid, P_r, m, n, \mathcal{P}, \text{end})$
- **else**
  - // there exists an honest successor P_{o_j}
  - $P_{o_j} \leftarrow P_{o_k}$ with smallest k such that $P_{o_k} \notin$ Bad $temp \leftarrow^R$ temporary ID
  - Send "Onion $temp$ from P_{o_i} routed through $(P_{o_{i+1}}, \ldots, P_{o_{j-1}})$ to P_{o_j}" to $\mathcal{S}$
  - Output_Corrupt_Sender$(P_s, sid, P_r, m, n, \mathcal{P}, temp)$
  - Add $(temp, O, j)$ to L

On message Deliver_Message$(temp)$ *from* $\mathcal{S}$
- // Adversary $\mathcal{S}$ (controlling all links) delivers onion belonging to $temp$ to next node
- **if** $(temp, \_, \_) \in L$ **then**
  - Retrieve $(temp, O = (sid, P_s, P_r, m, n, \mathcal{P}, i), j)$ from L $O \leftarrow (sid, P_s, P_r, m, n, \mathcal{P}, j)$
  - **if** $j < n + 1$ **then**
    - $temp' \leftarrow^R$ temporary ID;
    - Send "$temp'$ received" to P_{o_j}
    - Store $(temp', O)$ in B_{o_j}
  - **else**
    - **if** $m \neq \perp$ **then**
      - Send "Message m received" to P_r

On message Forward_Onion$(temp')$ *from* P_i
- // P_i is done processing onion with $temp'$ (either decided by $\mathcal{Z}$ if honest or $\mathcal{S}$ if corrupted)
- **if** $(temp', \_) \in B_i$ **then**
  - Retrieve $(temp', O)$ from B_i
  - Remove $(temp', O)$ from B_i
  - Process_Next_Step(O)

Figure 17: Ideal functionality $\mathcal{F}_{sub}$ for onion routing [26].