OrgAn: Organizational Anonymity with Low Latency

Debajyoti Das*
Purdue University
U.S.A.
das48@purdue.edu

Easwar Vivek Mangipudi*
Purdue University
U.S.A.
emangipu@purdue.edu

Aniket Kate
Purdue University
U.S.A.
aniket@purdue.edu

Abstract

There is a growing demand for network-level anonymity for delegates at global organizations such as the UN and Red-Cross. Numerous anonymous communication (AC) systems have been proposed over the last few decades to provide anonymity over the internet; however, they either introduce high latency overhead, provide weaker anonymity guarantees, or are difficult to be deployed at the organizational networks. Recently, the PriFi system introduced a client/relay/server model that suitably utilizes the organizational network topology and proposes a low-latency, strong-anonymity AC protocol. Using an efficient (almost) key-homomorphic pseudorandom function in lattice-based cryptography and Netwon's power sums, we present a novel AC protocol OrgAn in this client/relay/server model that provides strong anonymity against a global adversary controlling the majority of the network. OrgAn's cryptographic design allows it overcomes several major problems with any realistic PriFi instantiation: (a) unlike PriFi, OrgAn avoids frequent, interactive, slots and key agreement protocol among the servers; (b) a PriFi relay has to receive frequent communication from the servers which can not only become a latency bottleneck but also reveal the access pattern to the servers and increases the chance of server collusion/coercion, while OrgAn servers are absent from any real-time process; (c) finally, unlike PriFi, OrgAn can handle absentees and churn without re-running the setup.

We formally prove the strong anonymity of the OrgAn protocol and show that it defends against relevant active attacks. As another key contribution, we demonstrate how to make this public-key cryptographic solution scale equally well as the symmetric-cryptographic PriFi using practical pre-computation and storage requirements. We demonstrate the performance of OrgAn through a prototype implementation and up to four hundred-client experiments. We find that OrgAn provides similar end-to-end latency guarantees as PriFi, while still ignoring the PriFi's setup challenges. Our evaluation shows that network anonymity is feasible for latency-sensitive applications like VoIP, Skype video calls for organizations with a few hundred clients.

1 Introduction

In an influential work, Le Blond et al. [37] recognize an urgent need for traffic-analysis-resistant meta-data hiding (anonymous) communication at multinational organizations such as International Committee of the Red Cross (ICRC). The study finds that sensitive projects such as humanitarian activities at these organizations can be highly susceptible to subpoenas as well as powerful state-sponsored network eavesdroppers, and that there is a clear demand for anonymity for intra-organization communication as well as their interactions to the global services. As a recent US national Intelligence Council global trends' report [40] indicates we are moving towards a more contested world post pandemic, and anonymity needs at the global organizations [47] are bound to grow.

With its low latency and low bandwidth overheads, the Tor network [24, 25] is the most popular tool for anonymity over the Internet; however, the current Tor design is demonstrated to be significantly vulnerable to traffic analysis empirically [9, 27, 31] as well as conceptually [21]. Even otherwise, Tor is not suitable for organizational networks as (a) it carries some notoriety, and (b) it will route all the traffic over the Internet outside the network. The latter not only increases the attack surface and makes the communication traffic more susceptible to traffic analysis but also introduces latency overhead.

Mixing networks (mixnets) [14] improve the protection against traffic-analysis through increased latency overhead in the form of communication over several hops (i.e., indirection) and mixing messages at one or more honest hops. Over the last four decades, numerous mix-net inspired protocols [16, 20, 33–36, 49, 50] has been proposed that can deter traffic analysis attacks; however, their high latency overheads of several seconds (at least) is unacceptable for most organizational communication activities such as browsing, messaging, or video calls.

For high traffic-analysis resistance while maintaining low latency overhead, dining-cryptographers network (DC-net) [15] and its successors [12, 18, 29, 32, 51, 53, 54] are much better suited. Using a cryptographic setup/coordination among the users, these schemes offer provably strong anonymity in constant number of rounds [22]. Nevertheless, most standard DC-net designs have three key problems: (i) First, all the users need to run a key agreement protocol among themselves to agree on shared secrets keys; such an agreement protocol is not scalable as it comes with high communication overhead and has to be repeated often towards stopping linkability/co-relations across multiple rounds. (ii)Second, it requires all the users to participate in a slot agreement protocol before every round; otherwise two or more messages may collide as only one user is supposed to send a message in any given round. (iii) Finally, the DC-net designs draws their efficiency gains

*Both authors contributed equally to this research.

over mixnets through the fixed user setup and co-ordination between them [23]: Unlike for mixnets, any user arrival, absentees and departure mandates re-running the setup with the new group.

Solutions such as Dissent in Numbers [55], Verdict [19], MCMix [6] mitigate the first scalability problem by shifting the key agreement to the second tier: here, a set of servers perform DC-net like protocol on users' behalf and privacy is maintained as long as any one of the servers remains honest. (This is known as the *anytrust* assumption.) DiceMix [45] avoids the second slot agreement problem employing Newton's power sums formulation [30] by allowing all the n users to send one message each in a single round of the protocol. PowerMix [38] combines both ideas with the multi-party computation (MPC) as a service towards overcoming the above three problems, and recently, Blinder [4] protocol made the server-enabled design more efficient for a reduced adversarial threshold. To minimize the risks of coercion and collusion, these servers should be typically spread across different geopolitical jurisdictions, and these designs introduce significant latency overhead as the underlying MPC still require significant interaction among the servers.

With the above organizational anonymity problems in mind, PriFi [8] introduces a novel client/relay/server architecture, where a set of few servers (in an interactive manner) help the clients in establishing shared secrets among themselves, but actual anonymous communication happen through another relay node and the server only need to push messages to this relay nodes. As its key features, PriFi removes all server-to-server communications from the latency-critical path and it ensure that the intra-organization communications does not get routed over the Internet.

As demonstrated by Barman et al. [8], the client/relay/server model is very well suited for organizational anonymity; however, we observe four key problems with the current cryptographic design: (i) PriFi continues with the one user per slot model from Dissent and its successors, and a PriFi instantiation has to choose between a regular expensive slot selection or message linkability across different rounds. (ii) While the servers do not talk to each other in the online phase, they still need to be continuously online and communicate with the relay node for every slot. (iii) Third, every server as well as the relay has to know and interact with each other, which may facilitate coercion and collusion. (iv) Finally, dealing with any churn among the clients as well as the servers will be expensive in PriFi as they will require re-running the complete setup again.

1.1 Our Contribution

We present OrgAn, a new AC protocol using almost key-homomorphic pseudorandom functions (PRF) and Newton's power sums in the client/relay/server model. Similar to other recent DC-net based protocols, our protocol provides strong anonymity guarantees with resistance against intersection attacks and active attacks under Discrete Log (DL) and Ring Learning-with-rounding (R-LWR) assumptions.

Importantly, OrgAn resolves the above four problems with the PriFi design. The use of additive key-homomorphic PRF allows OrgAn to avoid the overheads of server-computed slot selection as messages from all clients are processed together using Newton's power sums. As the generated output is a random permutation of the input messages, OrgAn also ensures that a client's messages are not linkable across two different protocol runs even *without* rerunning the setup. OrgAn servers do not talk to each other or the relay node during the setup as well as the online phase. Moreover, the servers and the relay nodes do not even need to be aware of each others' presences, which can be particularly useful against coercion and collusion.

While providing above features using a public key cryptographic primitive (i.e., key-homomorphic PRF), OrgAn also maintains latency overhead in milliseconds using a storage vs. computation tradeoff associated with the R-LWR setting. Using a prototype implementation we show that OrgAn achieves reasonable round-trip-time (RTT) of 61 milliseconds for a system of 100 clients when they are communicating to the outside world compared to typical RTT of 20 milliseconds. OrgAn can easily scale for latency-sensitive and high throughput demanding applications like VoIP, VPN, Skype video calls etc. — it can support 1.9 Mbps throughput for every client in an organization with 100 clients.[1] We also compare the performance of OrgAn with the existing protocol (PriFi) and show that performance wise OrgAn is not much different from PriFi, despite solving the issues mentioned above.

We are the first to consider the churn in the DC-net setting and offer practical solution to the problem.

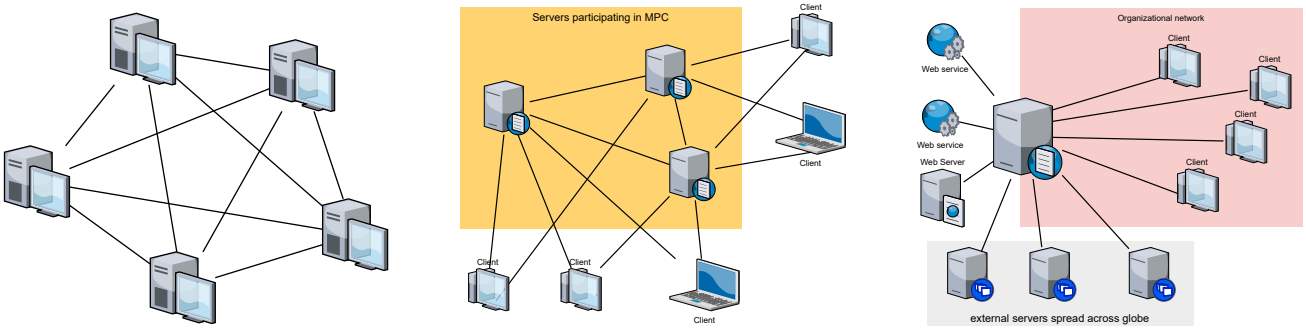
1.2 Comparison with Relevant DC-net inspired Protocols

As discussed earlier, most other mix-net inspired designs are not suitable for the organizational network or the LAN setting under consideration here. Moreover, Das et al. [22] shows that protocols inspired from dining cryptographers' network (DC-net) can achieve better anonymity compared to other techniques when low latency is required. With low latency and resistance to traffic-correlation attacks in mind, this work focuses on the DC-net based AC protocols.

As illustrated in Fig. 1, we divide the DC-net based systems into three protocol architectures: point-to-point (P2P) DC-nets, client-server MPC, and client-relay-server aggregation. In the following, we compare OrgAn with prominent DC-net based solutions across these architectures.

P2P DC-nets. In P2P DC-net protocols [18, 29, 45], the clients perform the anonymous broadcast themselves without involving any external computational server. These designs

[1] Skype recommends [48] 1.5 Mbps for HD video calls, and 4 Mbps for group video calls with 5 people.



(a) In the original DC-net [15, 29], Dissent [18], Dicemix [45] and similar protocols, clients interact with each others over point-to-point links to achieve anonymous broadcast among themselves.

(b) In server-aided model of [4, 6, 19, 38, 55] and others, each client only communicates with a set of servers, which interact with each other towards realizing anonymous broadcast for the clients.

(c) In the client/relay/server architecture of PriFi [8] and this work, the online phase does not involve any communication between the clients and the servers; the servers also do not need to communicate among themselves.

Figure 1. Illustrative Examples for Different Architectures for DC-net Inspired Protocols.

incur significant overhead (computation, as well as bandwidth) for key agreement and slot agreement. For example, DiceMix [45] take several seconds to complete a protocol run. OrgAn can be considered as an overhaul of DiceMix [45], where we remove the communication among clients by introducing a set of any-trust, setup servers. Each of these servers only sends a PRF key to every client during setup; it need not communicate with any other server or even be aware of them.

Client-server DC-nets. P2P DC-nets do not scale well as the numbers of clients increase. Client-server DC-net protocols [4, 6, 17, 19, 26, 38, 55] aim at making these protocol scale by shifting the DC-net (or similar computations) to the servers as MPC. The assumptions on the servers range from two-servers only, three-servers only, the any-trust assumption to a $3/4$ honest majority assumption, and correspondingly these protocols offer different guarantees in terms of the robustness, fairness and censorship-resistance. Nevertheless, all these protocols still require quadratic computation (in the number of clients) and may not be able manage overall latency overhead of less than a second. Moreover, from the organizational network perspective, the regular interaction between the servers makes them vulnerable to collusion in a geo-politically diverse setting.

Client/Relay/Server Model. PriFi [8] introduces what we call client/relay/server model that utilizes a relay server to avoid major latency overheads, while achieving anonymity in an organizational network. PriFi reliably achieves key agreement and slot agreement using a setup phase. However, the packets from the same user are linkable in between two setup runs in PriFi — if that needs to be avoided, the expensive setup needs to be run after every packet. Thus, PriFi achieves high throughput with compromised anonymity guarantees, or strong anonymity with less throughput. Our protocol OrgAn provides stronger anonymity guarantees by achieving unlinkability among the packets from the same user without compromising the throughput.

Although both PriFi and OrgAn depend on a group of external servers to generate shared secrets among the users — OrgAn does not require any involvement from those servers during the protocol run; thus, reducing the chance of coercion and collusion among them or with the relay.

2 Overview

2.1 Setup and Communication Model

Consider an organizational network with N clients $u_1, \ldots, u_{\mathsf{N}}$. They wish to access intra-organizational services as well as connect to services outside the organizational network without revealing which client is actually communicating. For this, they use our protocol OrgAn. Other than the clients, the infrastructure for OrgAn consists of a set of K setup servers denoted as $G_1, \ldots, G_{\mathsf{K}}$, and one relay node R. Using the relay node and the setup servers, the clients want to achieve anonymity while communicating with different services.

The relay R is a gateway server between the organizational network and the outside world. It helps the clients of the OrgAn protocol to transmit messages outside as well as to intra-organization services, but does not act as a trusted third party in the anonymization process. All the messages from the clients of the system (outbound traffic) are transmitted through the relay to the services (intra-oraganization or outside world). All the response messages from the outside world (inbound traffic) are received by the relay and forwarded to the clients. We assume that the relay has high availability and high computation power.

Additionally there is a small set of K servers $G_1, \ldots, G_{\mathsf{K}}$ outside the organization, we call them setup servers. These setup servers help the clients in the setup or key agreement process, but do not take part in the anonymization process.

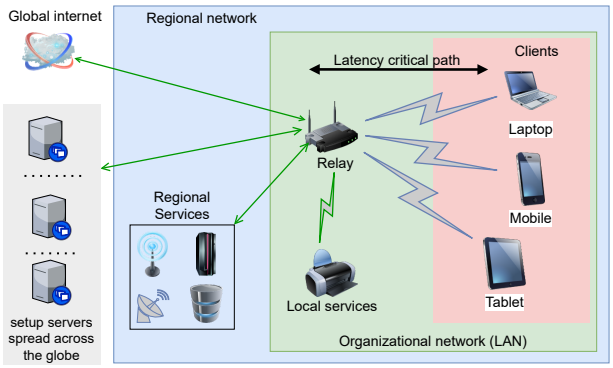


Figure 2. System overview of OrgAn

These servers could be maintained by independent third parties outside the organization and are assumed to be scattered across the globe.

We do not require the setup servers and the relay to communicate with each other during the setup phase or protocol run — they do not even need to know each other. The clients can mutually agree on the set of setup servers, ensuring that for each client there is at least one setup server that they trust. We do not require the setup servers to be online except for the setup phase.

We first describe the protocol in Section 4 assuming all the clients are online during the protocol run. In Section 8 we present the required modifications to handle client churn.

In this paper we only focus on the anonymity of outbound traffic. We consider the solution provided by PriFi for inbound traffic adequate and can be easily fit into our protocol; hence, we do not consider inbound traffic for the rest of the paper.

2.2 Threat Model

We consider a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ who can observe all network traffic. Additionally, the adversary can compromise some clients. However, we assume that at least two clients are honest. We assume that at least one of the setup servers is honest.

Moreover, the relay can be under adversarial control, but we do not consider denial-of-service (DoS) attacks from the relay, because the clients can easily identify if the availability of the relay is compromised — and the organization does not want to openly show that they are against privacy of their members/employees. However, the relay may launch other active attacks as long as the relay can not detected/blamed by the honest clients. We also do not consider an adversary whose sole purpose is to launch DoS attacks.

We allow the adversary to launch active attacks from inside the organizational network. However, for simplicity of description, we do not consider any active attacks like DoS from outside of the local network. Such attacks can be easily handled by proper ingress filtering by the relay, otherwise the relay itself will compromise its availability.

2.3 Goals

Anonymity. We want our protocol to achieve *sender anonymity* for the outbound traffic, i.e., the (in-)ability of any third party or the relay to figure out which user has sent a specific message, even if all but two clients and all but one server (setup node) are compromised. We want *strong* sender anonymity, which means, the adversarial advantage in guessing the sender correctly is at most negligibly better than random guessing.

Accountability. In our protocol, we want the honest clients to be able the detect the scenario where some malicious user(s) or the relay tries to disrupt the protocol by sending incorrect/malformed messages. For each such incident, the honest clients should be able to identify at least one malicious party with significant probability, and no honest user should be misidentified as a malicious party.

Low Latency. The system should operate with low latency overhead to support applications like voice and video calls.

Reduced Bandwidth Overhead. We want to minimize (preferably eliminate completely) any communication overhead because of the setup servers during the protocol run. We also want to minimize the communication overhead among the clients and the relay node.

Scalability. We want OrgAn to support small to medium organizations (i.e., up to several hundred clients).

Non-goals. Our design does not consider the followings:

DoS attacks. Similar to other DC-net based protocols [8, 8, 18, 19, 45, 55], our protocol does not consider an adversary whose sole purpose is to launch DoS attacks. This remains an open problem not only for DC-net based protocols, but in general for all anonymous communication systems.

Packet loss. Similar to other existing DC-net protocols [6, 8, 18, 19, 55], even OrgAn does not provide a solution for reliable packet delivery. If there is a packet loss between the user and the relay, the receiving end can request the packet again after a timeout.

2.4 Protocol Idea

Our protocol design is based on the idea that each client u_i has an individual secret $\mathbf{r}_i$ — here $\sum_i \mathbf{r}_i$ is known to everyone, but the individual values of $\mathbf{r}_i$ values with the honest clients are not known. The clients achieve that using a setup phase, with the help of a few setup servers. If there is at least one honest setup node, this setup can be achieved by a secret sharing scheme between the setup servers and the clients.

In our protocol design, we use an almost key-homomorphic PRF $\mathcal{F}$ that satisfies $\mathcal{F}(\mathbf{k}, x) = \mathcal{F}(\mathbf{k}_1, x) + \mathcal{F}(\mathbf{k}_2, x) + e$ with bounded elements in the error vector e and $\mathbf{k} = \mathbf{k}_1 + \mathbf{k}_2$. In a round d, each client u_i uses an element of $\mathcal{F}(\mathbf{r}_i, d)_t$ as the mask for their DC-net cipher that they send to the relay for a slot t. Since the relay knows $\sum_i \mathbf{r}_i$ but not the individual $\mathbf{r}_i$ values associated with the honest clients, it can

decrypt the message for slot t only after receiving the DC-net ciphers from all the clients for that slot by computing $\mathcal{F}(\sum_i \mathbf{r}_i, d)_t$. We use group arithmetic to eliminate the error, the relay retrieves the message correctly.

The main rationale behind using an *almost* key-homomorphic PRF instead of a perfectly key-homomorphic PRF is that the latter is computationally really expensive . By using a lattice-based additive *almost* key-homomorphic PRF we improve the performance drastically (cf. Section 7).

Base Protocol. We solve the slot agreement problem generally faced by DC-net based protocols, by constructing power-sums of messages similar to Dicemix [45]. However, we do it (significantly) more efficiently than Dicemix by exploiting the network model and the (almost) key homomorphic property of $\mathcal{F}$. We detail our protocol description in Sections 4 and 5. However, there is one crucial problem with the powersum equation system — with large number of clients, solving the equation system becomes the bottleneck of the whole system. Additionally, to send one IP packet (say, of 512 bytes length), the clients should be able to allow the relay to link all the *fragments*, considering each round allows the clients to send 226 bits (we call them *fragments*, the limitation comes from the computational limitation on group algebraic operations on a large group).

Bulk Protocol. We design a *Bulk* protocol inspired by the design of Dissent [18] to solve the above two problems. Before sending an IP packet, OrgAn uses the powersum equation system in the first round to generate a permutation for the fragments the clients want to send. In the following round the clients use that permutation for the fragments, and run a typical DC-net. The clients still generate the masks for the ciphertext using $\mathcal{F}$. The bulk protocol provides unlinkability for IP packets, but not the fragments of a single IP packet (and that is by design).

2.5 What We Achieve

In our design we mainly solve two latency critical problems inherent in other DC-net designs:

1. we get rid of the slot agreement overhead by using Dicemix[45]-like powersum equation system,
2. we avoid the requirement for regular key-agreement that is required even in Dicemix by using PRFs.

By getting rid of these two overheads, we achieve a round-trip-time (RTT) of 61 milliseconds for a system of 100 clients when they are communicating to the outside world compared to typical RTT of 20 milliseconds. OrgAn can easily scale for latency-sensitive and high throughput demanding applications like voice and video calls.

We show that our protocol provides strong sender anonymity against global passive network-level adversaries (in Section 6.4). Moreover, we prove security of our protocol against relevant active attacks (in Section 6.5). We show that honest clients can detect such active attacks as well as identify a maliciously acting party with overwhelming probability.

Additionally, we are the first to consider client churn for DC-net inspired protocols — in Section 8 we present our solution to handle client churn.

3 Preliminaries

Power-sum equations and solution[30, 45]. Consider the following system of equations:

$$E(1) = x_1 + x_2 + \cdots + x_{\mathsf{N}} \in \mathbb{Z}_{\mathsf{p}}$$
$$E(2) = x_1^2 + x_2^2 + \cdots + x_{\mathsf{N}}^2 \in \mathbb{Z}_{\mathsf{p}}$$
$$\cdots$$
$$E(\mathsf{N}) = x_1^{\mathsf{N}} + x_2^{\mathsf{N}} + \cdots + x_{\mathsf{N}}^{\mathsf{N}} \in \mathbb{Z}_{\mathsf{p}}$$

with each $x_i \in \mathbb{Z}_{\mathsf{p}}$. This equation system can be solved using Newton's identities [30, 45]. Mathematically we denote the function as $\mathsf{SolveEqn}(E(1), \ldots, E(\mathsf{N}))$ that takes such an equation system as input, and outputs an unordered set of N elements $\{x_1, x_2, \ldots, x_{\mathsf{N}}\}$, if the equation system is solvable. In the base protocol phase, each x_i is the input of client u_i; the equation system is computed and solved by the relay R using the $\mathsf{SolveEqn}()$ function to find an unordered set of client messages x_i.

Almost Key-Homomorphic Pseudorandom Function. A key-homomorphic pseudo random function family (PRF) [10] is a PRF which is homomorphic in the key-input of the function. Our protocol uses the lattice based pseudorandom function family [10] $\mathcal{F}(\mathbf{k}, x) = \lfloor \mathbf{k} \cdot \mathcal{H}(x) \rfloor_{\mathsf{q}} \in R_q$, for $\mathbf{k}, \mathcal{H}(x) \in R_{\mathsf{v}}$. R_{q} and R_{v} are polynomial rings; each element of the ring is a polynomial of degree less than u with integer coefficients, represented as a vector of u elements in $\mathbb{Z}_{\mathsf{q}}$ and $\mathbb{Z}_{\mathsf{v}}$. When the elements are polynomials $(\cdot)$ indicates the product of two such polynomial ring elements. $\mathcal{F}$ is computationally indistinguishable from a random function family, and almost key-homomorphic, i.e., $\mathcal{F}(\mathbf{k}, x) = \mathcal{F}(\mathbf{k}_1, x) + \mathcal{F}(\mathbf{k}_2, x) + \epsilon$ where $\epsilon \in R_{\mathsf{v}}$ with each coefficient of $\epsilon \in \{-1, 0, 1\}$. The security of the PRF considered is based on the Ring-LWR assumption provided in Appendix Section 6.1. For an element $x \in \mathbb{Z}_{\mathsf{v}}$, the rounding-down function is defined as $\lfloor x \rfloor_{\mathsf{q}} = \lfloor x \cdot \frac{\mathsf{q}}{\mathsf{v}} \rfloor \in \mathbb{Z}_{\mathsf{q}}$. Rounding down a vector involves rounding down each element of the vector.

Eliminating the error: While the employed PRF introduces an error, we use a suitable scaling of messages to eliminate the error while computing the equation system. For example, let $\mathbf{k} = \mathbf{k}_1 + \mathbf{k}_2$. Consider two clients with messages $x_1, x_2 \in \mathbb{Z}_{\mathsf{p}}$. To compute $x_1 + x_2$ at the relay, the two clients forward $c_1 = k \cdot x_1 + [\mathcal{F}(\mathbf{k}_1, t)]_i$ and $c_2 = k \cdot x_2 + [\mathcal{F}(\mathbf{k}_2, t)]_i$ for a certain t, i and suitable k. The relay computes $c = c_1 + c_2 - [\mathcal{F}(\mathbf{k}, t)]_i = kx_1 + kx_2 + \epsilon$. It computes $\frac{c}{k}$ and rounds the value to the nearest integer to obtain $x_1 + x_2$. Here, $[\mathcal{F}(\mathbf{k}, t)]_i$ is the i^{th} element of the vector $\mathcal{F}(\mathbf{k}, t)$. Refer Section 4.2 for details on the elimination of the introduced error.

$\mathcal{U}$	Set of all N users; $\mathsf{N} = \lvert U \rvert$
I	Set of all K setup servers; $\mathsf{K} = \lvert I \rvert$
G_j	The j-th guard node
R	The relay node
η	The security parameter
$a \xleftarrow{\$} [b, c]$	Draw uniformly at random from an integer range $[b, c]$
$\mathbb{Z}_\mathsf{p}, \mathbb{Z}_\mathsf{q}, \mathbb{Z}_\mathsf{v}$	Finite groups of integers of prime orders p, q, v resp.
$\lfloor x \rceil$	Nearest Integer of x
$x_1 \lvert\lvert x_2$	String x_1 is appended with string x_2
$\mathcal{H}(\cdot)$	Hash function used in the PRF that maps strings of arbitrary length to a element in ring R_v
$H(\cdot)$	Cryptographic hash function that maps $\{0,1\}^*$ to a finite field $\mathbb{F}$
$\mathcal{F}(\mathbf{k}, t)_i$	i^{th} element of the PRF obtained from key $\mathbf{k}$ for slot t

Figure 3. Protocol and system parameters for OrgAn

Additionally, in our protocol we use a digital signature scheme [46] that is existentially unforgeable under chosen-message attacks [42]. Let $(\mathcal{S}, \mathcal{V})$ be the signature scheme — given a private-public key pair (p, P), $\sigma = \mathcal{S}_p(x)$ denotes the signature of message x with the key p, and using $\mathcal{V}_P(x, \sigma)$ anyone can verify the signature.

4 Core Protocol

In this section we present the core OrgAn protocol, without protection against active attacks and without any client churn. In Section 5 we extend our protocol to defend against active attacks; in Section 8 we present our solution for client churn.

As mentioned earlier in Section 2.1, our system consists of the following set of parties:

- a set of N clients denoted as $u_1, \ldots, u_\mathsf{N}$ that (or some of them) want to communicate with the outside world;
- a set of K setup servers denoted as $G_1, \ldots, G_\mathsf{K}$ the reside outside the organizational network;
- one relay server R that acts as a gateway.

We assume that all the protocol parties in our system have access to a public key infrastructure (PKI), where each party X has a long term private-public key pair (p_X, P_X). We summarize the notations we use in this paper in Figure 3.

Our protocol first runs a one-time setup phase, where the clients receive random secret-shares of a value known to relay from the setup servers, and then start running the protocol. In our core protocol (without active attacks), the clients need to run the setup phase only once, and never again.

4.1 Setup Phase

Each setup server G_j splits a publicly known value $\mathbf{s}$ into N secret shares $\{\mathbf{r}_{1j}, \ldots, \mathbf{r}_{\mathsf{N}j}\}$ and distributes the shares among the clients, where each client u_i receives the share $\mathbf{r}_{ij}$, such that $\mathbf{s}, \mathbf{r}_{ij} \in \mathbb{Z}_\mathsf{v}^\mathsf{u}$ and $\sum_i \mathbf{r}_{ij} = \mathbf{s}$. Note here, all the setup servers use the same $\mathbf{s}$ value and that is a global parameter of the protocol, however, the shares for the clients generated by each honest setup server is independent of other servers, as well as unknown to other servers. We assume that the setup servers communicate with the clients using some authenticated and confidential channel (for example, using TLS [28, 44]).

After receiving one share from each setup server, each client u_i has the following secrets: $\{\mathbf{r}_{i1}, \ldots, \mathbf{r}_{i,\mathsf{K}}\}; \mathbf{r}_{ij} \in \mathbb{Z}_\mathsf{v}^\mathsf{u}$. Each client u_i computes the following.

$$\mathbf{r}_i = \sum_{j=1}^{m} \mathbf{r}_{ij} \in \mathbb{Z}_\mathsf{v}^u \tag{1}$$

We present the pseudo-code for the setup run by the setup servers and the clients in Figure 4.

Remark 1. *The adversary always knows that $\sum_i \mathbf{r}_{ij} = \mathbf{s}$. If there are two honest clients u_1 and u_2, the adversary can always compute $\mathbf{r}_{1j} + \mathbf{r}_{2j}$ for any honest setup server j. That is the only leakage after the setup phase — the adversary cannot guess the individual values of $\mathbf{r}_{1j}$ and $\mathbf{r}_{2j}$.*

$\mathbf{s} \in \mathbb{Z}_\mathsf{v}^u$; a global system-parameter

ServerSetup (setup server G_j, set $\mathcal{U} = \{u_1, \ldots, u_\mathsf{N}\}$) :
$\{\mathbf{r}_{1j}, \ldots, \mathbf{r}_{\mathsf{N}j}\}$ = split $\mathbf{s}$ into N shares
Send $\mathbf{r}_{ij}$ to user u_i over TLS for each $i \in \{1, \ldots, \mathsf{N}\}$

ClientSetup (user u_i, set $I = \{G_1, \ldots, G_\mathsf{K}\}$) :
$\mathbf{r}_{i1}, \ldots, \mathbf{r}_{i\mathsf{K}}$ = Wait for shares from each $G_j \in I$
$\mathbf{r}_i = \sum_{j=1}^{\mathsf{K}} \mathbf{r}_{ij}$
Send "Setup completed" to the relay R

RelaySetup() :
Initiate **ServerSetup**$(G_j, \mathcal{U})$ for each $G_j \in I$
Initiate **ClientSetup**(u_i, I) for each $u_i \in \mathcal{U}$
Wait for "Setup completed" from each $u_i \in \mathcal{U}$

Figure 4. Setup protocol in OrgAn

4.2 Base Protocol

Our protocol can be divided into several *rounds*; in one round each client u_i can send one message x_i. In every round, the relay R maintains N slots to receive N equations. The relay retrieves the N messages from N clients by solving those equations. For each round d, the protocol is run in the following steps:

Client Ciphertext generation. Each client u_i with a message $x_i \in \mathbb{Z}_\mathsf{p}$ computes the following, for each slot t in the round d:

$$\begin{aligned} x_i^t &\in \mathbb{Z}_\mathsf{p} \\ p_i(t) = \mathcal{F}\,(\mathbf{r}_i, d)_t &\in \mathbb{Z}_\mathsf{q} \qquad q > p \\ c_i(t) = \kappa \cdot x_i^t + p_i(t) &\in \mathbb{Z}_\mathsf{q} \qquad \kappa > 2\mathsf{N} \end{aligned}$$

Sender u_i then sends the ordered set $\{c_i(1), \ldots, c_i(\mathsf{N})\}$ tagged with the round number d to the relay.

Note that x_i^t is computed as a group element in $\mathbb{Z}_\mathsf{p}$, however when $\kappa \cdot x_i^t$, those are treated as integers as long as $\kappa \cdot x_i^t < \mathsf{q}$. We discuss shortly the exact relation between p, q and κ for which the equation system holds.

Slot value reveal. For a slot t, the relay R collects the ciphertexts $c_1(t), \ldots, c_{\mathsf{N}}(t)$ from all the users and computes the following:

$$\begin{aligned} P(t) =& c_1(t) + \cdots + c_{\mathsf{N}}(t) - \mathcal{F}(\mathsf{K} \cdot \mathbf{s}, d)_t \in \mathbb{Z}_{\mathsf{q}} \\ =& (\kappa \cdot (x_1^t + x_2^t + \cdots + x_{\mathsf{N}}^t) + e) \in \mathbb{Z}_{\mathsf{q}} \end{aligned}$$

After diving $P(t)$ by κ and rounding to the nearest integer, the relay gets:

$$E(t) = \lfloor P(t)/\kappa \rceil \mod \mathsf{p} = x_1^t + x_2^t + \cdots + x_{\mathsf{N}}^t \in \mathbb{Z}_{\mathsf{p}} \quad (2)$$

It is important to note that $E(t)$ is in $\mathbb{Z}_{\mathsf{p}}$. The structure of $E(t)$ remains unharmed as long as $\mathsf{q} > \mathsf{p}\kappa\mathsf{N} > 2\mathsf{p}\mathsf{N}^2$. We discuss more about the appropriate choices of p and q for satisfactory performance in Section 7, and in Section 6 security level for given values of security parameter η.

Once all the slot values from all the clients are received, the relay has $E(1), \ldots, E(\mathsf{N})$.

The relay can solve the above equation system to retrieve $x_1, x_2, \ldots, x_{\mathsf{N}}$ (without knowing which message belongs to which client) using $\mathsf{SolveEqn}(E(1), \ldots, E(\mathsf{N}))$. Once the individual values $x_1, x_2, \ldots, x_{\mathsf{N}}$ are retrieved, the relay can forward them to the outside world. We present the pseudocode for the protocol in Figure 5. The setup servers do not take part during the protocol run at all.

Remark 2. *$E(1) = x_1 + x_2 + \cdots + x_{\mathsf{N}}$ can be seen by the adversary in plain text. If all but two clients (without loss of generality let us assume u_1 and u_2) are compromised, the adversary knows $(x_1 + x_2)$. However, this leakage does not provide any additional information. Because, after solving the equation system the relay anyway knows the values.*

Remark 3. *Multiplying and diving by κ can be easily implemented with bit-shift. For $\mathsf{N} = 100$, we can choose $\kappa = 256$ — we can just add 8 bits (all zeros) at the end of a number to multiply with κ. Any error generated in the equation system will be contained in those 8 bits, and we can simply discard those last 8 bits before solving the equation system.*

4.3 Bulk Protocol: Scaling For Large Packets And A Larger Number Of Clients

The protocol described in Section 4.2 (what we call Base protocol) is a perfectly fine protocol to communicate to the outside world, except it does not scale well with number of clients. If the number of clients increases, solving the equation system becomes a bottleneck. Also, in order to send a single IP packet (typically 512 Bytes) using our Base protocol the client needs to break the packet into multiple *fragments* and send them over multiple rounds — and that adds additional complications. So, we present a *Bulk* protocol that avoids the above two problems.

Our Bulk protocol draws its inspiration from the Bulk protocol in Dissent [18]. To send one IP packet, each client participates in one round of Base protocol and one round of Bulk protocol.

$\mathbf{s} \in \mathbb{Z}_{\mathsf{v}}^u$: a global system-parameter
T: number of slots to be used in the current round

RelayProtocol(round d):
  $P_1, \ldots, P_T = -\mathcal{F}(\mathsf{K} \cdot \mathbf{s}, d)$
  for $i \in \{1, \ldots, \mathsf{N}\}$ **do**
    $(d_i, c_i(1), \cdots, c_i(T))$ = Wait for message from u_i
    for $t \in \{1, \ldots, T\}$ **do**
      $P_t = P_t + c_i(t) \mod \mathsf{q}$
    end for
  end for
  if $(d\%2 = 1)$ // Base round, $T = \mathsf{N}$ // **then**
    $E_1, \ldots, E_{\mathsf{N}} = \lfloor P_1/\kappa \rceil \mod \mathsf{p}, \ldots, \lfloor P_{\mathsf{N}}/\kappa \rceil \mod \mathsf{p}$
    $(x_1, t_1) \ldots, (x_{\mathsf{N}}, t_{\mathsf{N}}) = \mathsf{SolveEqn}(E_1, \ldots, E_{\mathsf{N}})$
    Store $(t_1, \ldots, t_{\mathsf{N}})$; Broadcast $(x_1, t_1) \ldots, (x_{\mathsf{N}}, t_{\mathsf{N}})$
  else
    // Bulk round, $T = t_1 + \ldots t_{\mathsf{N}}$ //
    $t = 0; x_1, \cdots, x_t = \lfloor P_1/\kappa \rceil \mod \mathsf{p}, \cdots, \lfloor P_t/\kappa \rceil \mod \mathsf{p}$
    for $i \in \{1, \ldots, \mathsf{N}\}$ **do**
      $\mathsf{packet}_i = x_{t+1} || \cdots || x_{t+t_i}; \; t = t + t_i$
    end for
    Send $\mathsf{packet}_1, \ldots, \mathsf{packet}_{\mathsf{N}}$ to internet
    Broadcast $(\mathsf{packet}_1, \ldots, \mathsf{packet}_{\mathsf{N}})$
  end if

ClientProtocol(client u_i, packet M, round d) :
  if $(d\%2 = 1)$ // Base round, $T = \mathsf{N}$ // **then**
    v_i = number of fragments required to send packet M
    x_i = pick a random number uniformly at random
    for each $j \in \{1, \ldots, T\}$ **do**
      $c_i(j) = \mathsf{GenCipher}((x_i || v_i), d, j)$
    end for
    Send $(d, c_i(1), \cdots, c_i(\mathsf{N}))$ to relay R
    $(x_{\pi_1}, t_1), \cdots, (x_{\pi_{\mathsf{N}}}, t_{\mathsf{N}})$ = wait for response from R
    if $!\exists (x_{\pi_j}, t_j) : (x_{\pi_j}, t_j) = (x_i, v_i)$ **then**
      Run Blame protocol
    else
      $\mathfrak{T} = t_1 + \cdots + t_{j-1}, T = t_1 + \cdots + t_{\mathsf{N}}$
    end if
  else
    // Bulk round, $T = t_1 + \ldots t_{\mathsf{N}}$ //
    $x_{i1}, \cdots, x_{iv}$ = Split packet M into v_i fragments
    for each $t \in \{1, \ldots, T\}$ **do**
      if $\mathfrak{T} < j \leq \mathfrak{T} + v$ **then**
        $c_i(t) = \mathsf{GenCipher}(x_{i,t-\mathfrak{T}}, d, t)$ // Send the packet //
      else
        $c_i(t) = \mathsf{GenCipher}(\mathbf{0}, d, t)$ // Send zeros //
      end if
    end for
    Send $(d, c_i(1), \cdots, c_i(T))$ to R
    $D_1, \ldots, D_{\mathsf{N}}$ = wait for response from R
    if $!\exists D_j : D_j = M$ **then** Run Blame protocol **end if**
  end if

$\mathsf{GenCipher}$**(message x, round d, slot t) :**
  if $(d\%2 = 1)$ **then** $y_1 = x^t \mod \mathsf{p}$; $y_2 = \kappa \cdot y_1 \mod \mathsf{q}$
  else $y_2 = \kappa \cdot x \mod \mathsf{q}$ **end if**
  $c = y_2 + \mathcal{F}(\mathbf{r}_i, d)_t \mod \mathsf{q}$; return c

Figure 5. Protocol run in OrgAn

Client Permutation Generation Using Base Protocol. Each client picks a random number $x_i \in \mathbb{Z}_{\mathsf{p}}$ and uses one round of Base protocol to broadcast (x_i, v_i) to all the users, where v_i is

the number of fragments required to send the IP packet the client wants to send. When the relay broadcasts the output set of pairs (x_i, v_i) that generate a random permutation Ψ of those pairs. Each client i checks the position $\Psi(i)$ of their own input in the output permutation. The client u_i sends v_i consecutive fragments in the immediate next Bulk round, starting from the slot $\sum_{j=1}^{\Psi(i)-1} v_{\Psi(j)} + 1$. The Bulk round will have a total of $T = \sum_{i=1}^{N} v_i$ slots.

Client Ciphertext Generation In Bulk Round. Each client u_i splits the IP packet into v_i fragments $x_{i1}, \ldots, x_{i,v_i}$. For each slot t in the Bulk round, the client computes the following:

$$T_i = \sum_{j=1}^{\Psi(i)-1} v_{\Psi(j)} \qquad p_i(t) = \mathcal{F}(\mathbf{r}_i, d)_t$$

$$c_i(t) = \begin{cases} p_i(t) & t \leq T_i \text{ or } t > T_i + v_i \\ \kappa \cdot x_{i,j} + p_i(t) & t = T_i + j, 0 < j \leq v_i \end{cases}$$

u_i then sends the ordered set $\{c_i(1), \ldots, c_i(T)\}$ tagged with the round number d to the relay R.

Message reveal. Similar to the Base protocol, for every slot t the relay R collects the ciphertexts $c_1(t), \ldots, c_N(t)$ from all the users and computes the following:

$$P(t) = c_1(t) + \cdots + c_N(t) - \mathcal{F}(K \cdot \mathbf{s}, d)_t$$

By computing $\lfloor P(t)/\kappa \rceil \mod \mathsf{p}$, the relay can reveal the corresponding slot value. Then the relay can retrieve the whole IP packets by bitwise concatenating the slot values of the associated slots. Then the relay sends the retrieved IP packets to the outside world.

Multiple rounds. The value of $\mathcal{F}(\mathbf{r}_i, d)$ can be computed for arbitrarily large value of d. Which means, the protocol can be run for a large number of rounds without the need to rerun the setup. Additionally, the relay does not need to receive ciphertexts for different rounds in the correct sequence, the relay can map them correctly using the round tag and slot id associated with each ciphertext.

Although the round number d can be arbitrarily large in $\mathcal{F}(\mathbf{r}_i, d)$, for the purpose of forward secrecy we recommend running the setup once in every few days. That procedure at regular intervals can be invoked by the relay. If the relay does not run the setup regularly as expected, the clients will suspect the relay's malicious intentions.

4.4 Performance Improvement With Preprocessed PRF Values

A lot of computation overhead on the client during the protocol run can be reduced using some preprocessing. The preprocessing can happen in the following two steps:

Preprocessed Hash Values. Recall that we use a lattice based PRF of the form $\mathcal{F}(\mathbf{r}_i, d) = \lfloor \mathbf{r}_i \cdot \mathcal{H}(d) \rfloor_{\mathsf{q}}$. The part $\mathcal{H}(d)$ is completely independent of any secret values associated with the client, and the input to $\mathcal{H}$ gradually increases with the round number. Therefore, for a large integer $\mathfrak{T}$, the hash values can be publicly available for the range of input values $[1, \mathfrak{T}]$, or provided to the clients as part of the setup. The clients only need to rerun the setup when those hash values are exhausted, and with the new set of secrets, they can start reusing the hash values. This takes away from the client the requirement and the computational overhead of computing the hash values altogether.

Preprocessing Overall PRF. Additionally, we assume that the clients have a preprocessing time everyday before they start using the system — using the already available hash values, the clients can preprocess the PRF computation for several rounds (or for the whole day). With the preprocessed $p_i(t) = \mathcal{F}(\mathbf{r}_i, d)_t$ values, each client needs to perform only one scalar multiplication and one addition to compute $c_i(t) = \kappa \cdot x_{i,j} + p_i(t)$ for every round.

5 Handling Disruption

We now describe how OrgAn can detect and defend against a compromised or malfunctioning participant might disrupt the protocol by sending wrong/malformed ciphertexts. Our methods here remains the same for both Base and Bulk protocols. As suggested in Section 2.2, we do not consider active attacks from outside the organizational network.

5.1 Disruption Detection

To detect if there is a disruption we modify our protocol for the relay R to broadcast a response message with the values $x_1, \ldots, x_N$ at the end of each round. Each client u_i checks if her x_i exists in the response message. If x_i cannot be found, the client can initiate the Blame protocol by sending the value of $\mathbf{r}_i$ and x_i to the relay R via a direct channel.

Note that the disruption detection technique depends on the fact that the relay R delivers the same response message to all the client. In case the relay does not do that, the technique mentioned in Section 5.3 defends against that.

A client u_i cannot find x_i in the response message means the relay has retrieved at least one wrong message. It is possible in three ways: (1) the relay has done the computations wrong or intentionally modfied the messages, (2) some malicious client has sent a bad ciphertext, (3) Some malicious setup server has distributed bad shares. If the relay did not act maliciously, it is the relay's responsibility to find the culprit.

5.2 Blame Protocol

The Blame protocol is invoked by at least one client and then it is run by the relay R. During the Blame protocol, all the x_i and $\mathcal{F}(\mathbf{r}_i, t)$ values are opened corresponding to that round. For a disruption in a Bulk round the last Base round is opened as well. This kind of leakage seem to weaken the anonymity guarantees, however, this is inherent to DC-net based systems in general [45]. In Section 9.1 and Appendix B.2 we discuss how to minimize the leakage and provide a trade-off between maintaining anonymity vs. punishing a disruptive party.

For the Blame protocol to work correctly, we need to slightly modify the setup phase as well as the protocol run. We describe the modifications below.

Modifications in the setup phase. In the setup phase, now the setup servers need to generate additional information along with $\mathbf{r}_{i,j}$ values so that those additional information can be used later to verify the correctness of the PRFs. Let the secret shared by G_j to client u_i be $\mathbf{r}_{ij} = \{\alpha_{ij1}, \ldots, \alpha_{iju}\}$ such that each $\alpha_{ij1}, \ldots, \alpha_{iju} \in \mathbb{Z}_{\mathsf{v}}$, and let all the setup servers and the relay mutually agree on a $g \in \mathbb{Z}_{\mathsf{v}}$. Then each guard server G_j computes $\Gamma_{\mathbf{r}_{ij}} = \{g^{\alpha_{ij1}}, g^{\alpha_{ij2}}, \ldots, g^{\alpha_{iju}}\}$. G_j sends a signature $\sigma_{ij} = \mathcal{S}_{p_{G_j}}(u_i, \Gamma_{\mathbf{r}_{ij}})$ along with the value $\mathbf{r}_{ij}$ to the client u_i.

Once the client u_i receives the tuple $(\mathbf{r}_{ij}, \sigma_{ij})$, u_i computes $\Gamma_{\mathbf{r}_{ij}}$ verifies the signature σ_{ij}. If those verifications are successful, u_i forwards $(\Gamma_{\mathbf{r}_{ij}}, \sigma_{ij})$ to the relay R. The relay additionally verifies the following two conditions:

- for each $\ell \in \{1, 2, \ldots, u\}$ if $\Pi_{i=1}^{\mathsf{N}} g^{\alpha_{ij\ell}} = g^{s_\ell}$ holds, where $\mathbf{s} = \{s_1, s_2, \ldots, s_u\}$;
- if each σ_{ij} is a valid signature of $(u_i, \Gamma_{\mathbf{r}_{ij}})$ generated by the setup server G_j.

The above steps in the setup phase ensures that each setup server has generated and distributed the $\mathbf{r}_{i,j}$ values correctly. Therefore, during the Blame protocol run, the relay or the clients will not require interaction with the setup servers.

Modifications in the protocol run. During the protocol run, each client u_i sends all the ciphertext values $\{c_i(1), \ldots, c_i(\mathsf{N})\}$ along with a signature $\sigma_i = \mathcal{S}_{p_{u_i}}((d, c_i(1), \ldots, c_i(\mathsf{N})))$, for round number d. The relay verifies the signature, and also stores σ_i for future use.

When Blame is invoked. If a client invokes the Blame protocol, it is run by the relay R. The idea of our Blame protocol is very similar that of Dicemix [45]. All x_i and $\mathcal{F}(\mathbf{r}_i, d)$ values are revealed for a round d, however, the $\mathbf{r}_i$ values are never opened. Unlike Dicemix, we do not need the clients to publish commitments of x_i and $\mathcal{F}(\mathbf{r}_i, d)$ values before every round, instead we need verifiability of correct PRF evaluation.

Once Blame is invoked, the relay asks all the clients to send their $\mathcal{F}(\mathbf{r}_i, d)$ and x_i values. Additionally, each client u_i needs to send a non-interactive zero-knowledge proof (NIZKP) λ_i of correct evaluation of the function $\mathcal{F}$. If a Base round is disrupted $\mathcal{F}(\mathbf{r}_i, d)$, x_i and λ_i values are sent for all the slots in that round, however for a Bulk round, it is sufficient to verify only the disrupted slot. We provide the detailed structure and the verification method of λ_i values in Appendix B.1. A non-compliant client will be blamed by the relay.

Once all the values are received, the relay recomputes the $c_i(t)$ values with the newly sent values of $\mathcal{F}(\mathbf{r}_i, d)$ and x_i, and verify that they are consistent with the previously sent values. Additionally, the relay verifies the correctness of $\mathcal{F}(\mathbf{r}_i, d)_t$ values using λ_i values. The client who sent a wrong $c_i(t)$ or λ_i will be blamed and excluded from the protocol.

If no client is found guilty (all clients have run the protocol honestly), that means the shares $\mathbf{r}_{ij}$ distributed by the setup servers to the clients are not correct, or the relay corrupted the messages. However, if it is the first case, the relay should have flagged that during the verifications in the setup phase — and hence, the relay is blamed. If the relay cannot find a party to blame, by default, the relay is blamed.

5.3 Equivocation Protection

With our key-homomorphic PRF based construction, we can achieve protection against equivocation almost for free. We include a summary of history till round $(d-1)$ as part of the key of the PRF function in round d:

$$p_i(t) = \mathcal{F}(h(d-1) \cdot \mathbf{r}_i, d)_t \in \mathbb{Z}_{\mathsf{q}}$$

where $h(d-1)$ is computed locally by each client as $h(d-1) = H(h(d-2), D(d-1))$ for the response message $D(d-1)$ sent by the relay in round $(d-1)$. $H(\cdot)$ is a cryptographically secure hash function that maps $\{0,1\}^*$ to a finite field $\mathbb{F}$. Note that $h(d-1)$ is a scalar value, whereas $\mathbf{r}_i$ is a vectors. And hence, the multiplication between $h(d-1)$ and $\mathbf{r}_i$ is a scalar multiplication.

Then the relay reveals the slot value for a slot t in the round d from the ciphertexts $c_1(t), \ldots, c_{\mathsf{N}}(t)$ using,

$$P(t) = c_1(t) + \cdots + c_{\mathsf{N}}(t) - \mathcal{F}(h(d-1) \cdot \mathsf{K} \cdot \mathbf{s}, d)_t.$$

With this minor modification, the relay will be unable to retrieve all future messages if it transmits different values for $D(d)$ to different clients in any round.

6 Security Analysis

In this section, we prove the (sender) anonymity property of our protocol OrgAn in the passive adversary setting. Then we prove the security properties offered by our protocol against relevant active attacks. We start with presenting the security definitions that we use in our proofs — we borrow those definitions from existing literature[7, 10, 39], and use them in our work.

6.1 Ring Learning-with-rounding Assumption

Below we define the Ring Learning-with-rounding (R-LWR) problem. It is assumed that the R-LWR problem is difficult to solve by a computationally bounded adversary.

Definition 1 (R-LWR [7])**.** *Let the Ring-LWR distribution $\mathcal{D}_s$ to be over $R_{\mathsf{v}} \times R_q$, $\mathsf{v} > \mathsf{q}$ obtained by choosing $\mathbf{a} \leftarrow R_{\mathsf{v}}$ uniformly at random for some $\mathbf{s} \in R_{\mathsf{v}}$ and outputting $(\mathbf{a}, \mathbf{b}) = (\mathbf{a}, \lfloor \mathbf{a} \cdot \mathbf{s} \rfloor_q)$. The decisional R-LWR problem R-LWR$_{u,v,q}$ consists of distinguishing samples $(\mathbf{a}_i, \mathbf{b}_i)$ from uniform and independently drawn samples $(\mathbf{a}_i, \mathbf{u}_i) \in R_{\mathsf{v}} \times R_q$.*

Then, R-LWR assumption states that the advantage of an adversary $\mathcal{A}$ in solving the decisional R-LWR problem $\mathbf{Adv}^{RLWR}_{u,v,q}(\mathcal{A}) = |Pr[\mathcal{A}(\mathbf{a}, \lfloor \mathbf{a} \cdot \mathbf{s} \rfloor_q) = 1] - Pr[\mathcal{A}(\mathbf{a}, \mathbf{u}) = 1]|$ is negligible, with the probabilities taken over $\mathbf{a} \sim U(R_{\mathsf{v}})$, $\mathbf{s} \sim U(R_{\mathsf{v}})$, and $\mathbf{u} \sim U(R_q)$.

6.2 Security Definition for PRFs

Let $\mathsf{Rand}(\mathcal{D}, \mathcal{O})$ denotes the set of all functions with domain $\mathcal{D}$ and range $\mathcal{O}$. We consider a distinguisher $\mathcal{A}$ that tries to distinguish if a function g has been picked randomly from a given function family $\mathcal{F}$ or from $\mathsf{Rand}(\mathcal{D}, \mathcal{O})$, when $\mathcal{A}$ is given oracle access to g. We write $\mathcal{A}(g)$ to denote that $\mathcal{A}$ is given oracle access to g. We define the following security game:

Definition 2. *Let $\mathcal{F} : \mathcal{K} \times \mathcal{D} \to \mathcal{O}$ be a family of efficient functions, and let $\mathcal{A}$ be an algorithm the takes an oracle for a function and returns a bit b. We consider the following two experiments:*

$$\begin{array}{r|l} \mathbf{Expt}_{PRF}(\mathcal{A}) & \mathbf{Expt}_{g}(\mathcal{A}) \\ \mathbf{K} \leftarrow \mathcal{K} & g \leftarrow \mathsf{Rand}(\mathcal{D}, \mathcal{O}) \\ b = \mathcal{A}(\mathcal{F}_{\mathbf{K}}) & b = \mathcal{A}(g) \end{array}$$

The adversarial advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{F}}(\mathcal{A}) = \Pr\left[\mathbf{Expt}_{PRF}(\mathcal{A})\right] - \Pr\left[\mathbf{Expt}_{g}(\mathcal{A})\right]$.

If we use $\mathcal{F}$ in a protocol that requires that the security can be broken with at most negligible probability for a security parameter η, we also want $\mathbf{Adv}_{\mathcal{F}}(\mathcal{A})$ to be negligible in η. Therefore, we use the following security definition for pseudorandom functions:

Definition 3. *$\mathcal{F}$ is a secure pseudorandom function family if, for all probabilistic polynomial time algorithms $\mathcal{A}$, the adversarial advantage* $\mathbf{Adv}_{\mathcal{F}}(\mathcal{A})$ *in the security game defined in Definition 2 is bounded by a negligible quantity in the security parameter η.*

6.3 Anonymity Definition

We focus on sender anonymity for our protocol. We formally define anonymity based on AnoA [39] framework as an indistinguishability-based interactive game between a challenger (running the protocol) and a PPT adversary. The goal of the adversary is to find out which of two adversarially-chosen senders has sent a message to a specific recipient (sender anonymity). More formally, the adversary tries to guess a challenge bit b of the challenger in the game.

Definition 4 ((α, δ)-IND−ANO)**.** *A protocol Π is (α, δ)-*IND−ANO *for the security parameter η, an anonymity function α and a distinguishing factor $\delta(\cdot) \geq 0$, if for all PPT machines $\mathcal{A}$,*
$\Pr[0 = \langle \mathcal{A} | \mathtt{Ch}(\Pi, \alpha, 0) \rangle] \leq \Pr[0 = \langle \mathcal{A} | \mathtt{Ch}(\Pi, \alpha, 1) \rangle] + \delta(\eta)$.

Definition 5 (Sender anonymity)**.** *A protocol Π provides δ-*sender anonymity *if it is (α_{SA}, δ)-*IND−ANO *for α_{SA} as defined in Figure 6.*

Adaptive AnoA Challenger $\mathtt{Ch}(\Pi, \alpha, b)$

Upon message $(\mathtt{Input}, u, R, m)$**:**
RunProtocol(u, R, m)

Upon message $(\mathtt{Chall}, u_0, u_1, R_0, R_1, m_0, m_1)$**:**
Compute $(u^*, R^*) \leftarrow \alpha(u_0, u_1, R_0, R_1, b)$
RunProtocol(u^*, R^*, m_0)

RunProtocol(u, R, m)**:**
Run Π on $r = (u, R, m)$ and forward all messages that are sent by Π to the adversary $\mathcal{A}$ and send all messages by the adversary to Π.

$\alpha_{SA}(u_0, u_1, R_0, R_1, b) = (u_b, R_0)$

Figure 6. Adaptive AnoA Challenger [39]

6.4 Anonymity Analysis

Theorem 1 (Sender Anonymity of Base Protocol)**.** *Assuming $\mathcal{F}()$ is a computationally secure pseudorandom function, the protocol Base protocol of OrgAn provides sender anonymity as defined in Definition 5 with negligible δ against any global passive adversary $\mathcal{A}$, as long as at least two users and one setup server are honest.*

Proof. Without loss of generality let us assume that users u_1 and u_2 are honest and the message associated sent by them in a given round are x_1 and x_2 respectively. Let us also assume that only one setup server G_1 is honest. Now we prove security in two part parts:

1. First we use a modified version $OrgAn^*$ of the protocol OrgAn and show that the adversary has a negligible advantage against $OrgAn^*$. In $OrgAn^*$, the user u_1 uses a random function $\mathcal{F}_{rand}(\cdot)$ instead of $\mathcal{F}(\mathbf{r}_1, \cdot)$ as the masks to compute the ciphertexts; and the user u_2 uses $\mathcal{F}(\mathbf{r}_1 + \mathbf{r}_2, \cdot) - \mathcal{F}_{rand}(\cdot)$.
2. Next we show that, if an adversary $\mathcal{A}_{anon}$ wins the game against OrgAn in the anonymity game, we can construct an adversary $\mathcal{A}_{PRF}$ that can win the PRF game.

As our first step, we consider the anonymity game with the protocol $OrgAn^*$. In $OrgAn^*$, all the protocol parties except u_1, u_2 and G_1 behave exactly the same as OrgAn. However, in the hypothetical protocol $OrgAn^*$ we assume that u_1 and u_2 collude in the following way: for a given slot t the user u_1 uses $\mathcal{F}_{rand}(t)$ as the mask to compute ciphertext $c_1(t) = \kappa x_1^t + \mathcal{F}_{rand}(t)$, and the user u_2 compute ciphertext $c_2(t) = \kappa x_2^t + \mathcal{F}(\mathbf{r}_1 + \mathbf{r}_2, t) - \mathcal{F}_{rand}(t)$. For the time being, let us consider only one round and we will extend the argument for multiple rounds shortly.

In this hypothetical protocol $OrgAn^*$, we can assume that the users u_1 and u_2 can exchange information about $\mathbf{r}_1, \mathbf{r}_2$ and $\mathcal{F}_{rand}()$ with each other.

Claim 1. *The protocol $OrgAn^*$ provides sender anonymity with $\delta = 0$ against any global passive adversary $\mathcal{A}$, for a one-round protocol run.*

Proof of Claim. Since $\mathcal{F}_{rand}()$ is a random function, the value $\mathcal{F}_{rand}(t)$ can be thought of as being chosen at random. Let, $f_1 = \mathcal{F}_{rand}(1)$ and $f_2 = \mathcal{F}(\mathbf{r}_1 + \mathbf{r}_2, 1) - \mathcal{F}_{rand}(1)$. Then the adversary $\mathcal{A}$ has the following set of equations for slot 1 with x_1, x_2, f_1, f_2 as unknowns (we skip the group notations for simplicity),

1. $x_1 + x_2 = a_1$
2. $\kappa x_1 + f_1 = a_2$
3. $f_2 + \kappa x_2 = a_3$
4. $f_1 + f_2 = a_4$

and the adversary knows $\langle x_1, x_2 \rangle = \langle b_1, b_2 \rangle$ or $\langle b_2, b_1 \rangle$, for some observer values of $a_1, a_2, a_3, a_4, b_1, b_2$. Note that the above equation system has a rank of at most 3; both $\langle b_1, b_2 \rangle$ or $\langle b_2, b_1 \rangle$ will yield valid values of $f1$ and $f2$. Therefore, slot 1 does not reveal anything about who sent x_1 or x_2.

Since $\mathcal{F}_{rand}$ is a random function, $\mathcal{F}_{rand}(t)$ is unrelated from $\mathcal{F}_{rand}(t')$ for any $t' \neq t$. And hence, a similar argument can be extended for any other slot t' as well, independent of slot t. Since the overall equation system to retrieve all the messages in a round is an identity, the adversary has $\delta = 0$ advantage in the sender anonymity game against the protocol $OrgAn^*$ for a one-round protocol run. ◇

Now let us consider the scenario when the protocol $OrgAn^*$ is run for many rounds. For every round d and slot t, the user u_1 can use $(d\mathsf{N} + t)$ as the input to the random function $\mathcal{F}_{rand}()$. In that case, the input to $\mathcal{F}_{rand}()$ is never repeated, and $OrgAn^*$ provides sender anonymity with $\delta = 0$ even for a multi-round protocol run. We skip the formal claim statement and proof here, since they are similar to that of Claim 1.

Now that we have proved δ-sender anonymity for $OrgAn^*$ with $\delta = 0$, we proceed to the next step where we prove the anonymity of OrgAn. We show that, if there exist an adversary $\mathcal{A}_{anon}$ that breaks sender anonymity for protocol $OrgAn$, we can construct an adversary $\mathcal{A}_{PRF}$ that break the security assumption on pseudorandom function $\mathcal{F}$.

Claim 2. *If there exist a PPT adversary $\mathcal{A}_{anon}$ with an adversarial advantage δ against the protocol OrgAn in the sender anonymity game defined in Definition 5, there exist an adversary $\mathcal{A}_{PRF}$ that can distinguish between $\mathcal{F}$ and $\mathcal{F}_{rand}$ with probability at least δ in the PRF game defined in Definition 2.*

Proof of Claim. We start with the construction of $\mathcal{A}_{PRF}$: our adversary $\mathcal{A}_{PRF}$ of the PRF game will run the whole sender anonymity game as the challenger, except one setup server G_1 (as per our threat model at least one setup server is honest, and without loss of generality we assume that to be G_1).

The key $\mathbf{K}$ for the PRF game is decided based on the random number $\mathbf{r}_{11}$ picked by G_1. We pick, $\mathbf{K} = \sum_j \mathbf{r}_{1j} = \mathbf{r}_1$ such that $\mathbf{r}_1 = \mathbf{K}$. Since G_1 as an independent honest party that does not collude with $\mathcal{A}_{PRF}$ or $\mathcal{A}_{anon}$[2], $\mathcal{A}_{PRF}$ does not know $\mathbf{r}_{11}$ or $\mathbf{K}$.

$\mathcal{A}_{PRF}$ runs each round of the sender anonymity game in the following way: for each slot value t $\mathcal{A}_{PRF}$ queries the PRF game with input value t and receives a value f_t. $\mathcal{A}_{PRF}$ asks the user u_1 in the sender anonymity game to use f_t to compute $c_1(t) = \kappa x_i^t + f_t$. Similarly, $\mathcal{A}_{PRF}$ asks u_2 to use $c_2(t) = \kappa x_i^t + \mathcal{F}(\mathbf{r}_1 + \mathbf{r}_2, t) - f_t$. $\mathcal{A}_{PRF}$ runs the sender anonymity game until $\mathcal{A}_{anon}$ halts. $\mathcal{A}_{PRF}$ returns 1 if and only if $\mathcal{A}_{anon}$ wins the sender anonymity game.

When f_t is an output of $\mathcal{F}_{rand}()$ the adversary $\mathcal{A}_{PRF}$ is effectively running $OrgAn^*$, however, when $f_t = \mathcal{F}(\mathbf{K}, t)$ it is running OrgAn. If $\mathcal{A}_{anon}$ has an advantage of δ in the sender anonymity game against OrgAn, there would be a difference of at least δ in the probability $\mathcal{A}_{PRF}$ outputs 1 when f_t is the output of $\mathcal{F}_{rand}()$ vs when it is the output of $\mathcal{F}()$, ◇

Following the above claim, if the adversarial advantage of $\mathcal{A}_{anon}$ is non-negligible against OrgAn, so is the adversarial advantage of $\mathcal{A}_{PRF}$ in the PRF game — which contradicts the assumption that $\mathcal{F}$ is a secure pseudorandom function. □

The above theorem shows that the Base protocol of OrgAn provides sender anonymity with negligible adversarial advantage when the protocol runs without any disruption. Now we extend the argument for Bulk protocol.

Theorem 2 (Sender Anonymity of OrgAn)**.** *Assuming $\mathcal{F}()$ is a computationally secure pseudorandom function, the protocol OrgAn provides sender anonymity (when Bulk protocol is employed) as defined in Definition 5 with negligible δ against any global passive adversary $\mathcal{A}$, as long as at least two users and one setup server are honest.*

It is important to note here, we are considering a whole IP packet as the message in our anonymity game. Therefore the challenge message in the anonymity game can be of varying length. Since the proof has a lot of similarity with the proof of Theorem 1, we only present a proof sketch below.

Proof Sketch. We use a similar technique as the proof of Theorem 1 to prove this theorem:

1. First we use a modified version $OrgAn^*$ of the protocol OrgAn that employs Bulk protocol and show that the adversary has a negligible advantage against $OrgAn^*$. Similar to the proof of Theorem 1, the user u_1 in $OrgAn^*$ uses a random function $\mathcal{F}_{rand}(\cdot)$ instead of $\mathcal{F}(\mathbf{r}_1, \cdot)$ as the masks to compute the ciphertexts, and the user u_2 uses $\mathcal{F}(\mathbf{r}_1 + \mathbf{r}_2, \cdot) - \mathcal{F}_{rand}(\cdot)$.
2. Next we show that, if an adversary $\mathcal{A}_{anon}$ wins the game against OrgAn in the anonymity game, we can construct an adversary $\mathcal{A}_{PRF}$ that can win the PRF game.

[2]More formally G_1 can modeled similar to hybrid functionalities in UC framework, and then the security game can be defined in that hybrid functionality setting. We skip the rigorous formalization in this paper.

Claim 3. *The protocol OrgAn* provides sender anonymity with $\delta = 0$ against any global passive adversary $\mathcal{A}$.*

The proof of this claim is similar to the argument of Claim 1 and its extension for multiple rounds, except, in every even round (bulk) round the clients use group addition instead of powersum equation system. For every slot t, the adversary can see $\sum_i x_i$ where every x_i are the values sent by users u_i. Since u_1 and u_2 use the masks $\mathcal{F}_{rand}(t)$ and $\mathcal{F}(\mathbf{r}_1 + \mathbf{r}_2, t) - \mathcal{F}_{rand}(t)$, the property of $\mathcal{F}_{rand}$ ensures that the adversary does not know if u_1 or u_2 is the actual sender for slot t.

It is crucial to notice that the Base round before a Bulk round reveals which slots are related, however, all those slots are used to send a single message. Therefore, that is not an actual leak, and does not break anonymity.

Now that we have proved sender anonymity for *OrgAn** with $\delta = 0$, we can prove anonymity for OrgAn exactly in the same way as in the proof of Theorem 1. We show that, if there exist an adversary $\mathcal{A}_{anon}$ that breaks sender anonymity for protocol *OrgAn*, we can construct an adversary $\mathcal{A}_{PRF}$ that break the security assumption on pseudorandom function $\mathcal{F}$.

Claim 4. *If there exist a PPT adversary $\mathcal{A}_{anon}$ with an adversarial advantage δ against the protocol OrgAn in the sender anonymity game defined in Definition 5, there exist an adversary $\mathcal{A}_{PRF}$ that can distinguish between $\mathcal{F}$ and $\mathcal{F}_{rand}$ with probability at least δ in the PRF game defined in Definition 2.*

We construct $\mathcal{A}_{PRF}$ and setup the anonymity game exactly in the same way as the proof of Theorem 1. If the adversarial advantage of $\mathcal{A}_{anon}$ is non-negligible against OrgAn, so is the adversarial advantage of $\mathcal{A}_{PRF}$ in the PRF game — which contradicts the assumption that $\mathcal{F}$ is a secure pseudorandom function. □

The above theorem consider both Base and Bulk rounds, and OrgAn provide anonymity with negligible δ as long as the PRF $\mathcal{F}$ is secure, given that the protocol is run without any disruption.

6.5 Security Against Active Attacks

When some protocol party tries to disrupt the protocol by sending malformed message, we want the honest clients to be able to detect that with overwhelming probability and be able to identify and punish the culprit. But first, we want to show that the relay cannot launch equivocation attacks, i.e., every honest client always receives the same response message after every round.

Lemma 1. *Assuming $\mathcal{F}$ is a secure PRF as well as (almost) key homomorphic with a bounded error e, and H is a collision resistant hash function, if the relay sends two different output messages D_i and D_j ($D_i \neq D_j$) to any two honest clients u_i and u_j in a round d, the relay lose the ability to run any later rounds with overwhelming probability.*

Proof Sketch. The retrieval of the messages depends on the (almost) homomorphic property of the PRF $\mathcal{F}$. Suppose $h_i = H(D_i)$ and $h_j = H(D_j)$; by collision resistance property $h_i \neq h_j$ when $D_i \neq D_j$. clients u_i and u_j uses the keys $h_i \cdot \mathbf{r}_i$ and $h_j \cdot \mathbf{r}_j$ as the keys to the PRF $\mathcal{F}$.

If the relay wants to be able to decrypt messages in the next round, the relay should be able to:

1. solve $(h_i \cdot \mathbf{r}_i + h_j \cdot \mathbf{r}_j)$ from $\mathbf{r}_i + \mathbf{r}_j$ without knowing the individual values of $\mathbf{r}_i$ and $\mathbf{r}_j$,
2. OR, somehow guess the value of $\mathcal{F}(\mathsf{K} \cdot \mathbf{s} + (h_i - 1) \cdot \mathbf{r}_i + (h_j - 1) \cdot \mathbf{r}_j)$ within as error bound of $\mathsf{N} \cdot e$ (there are total N additions).

The first part cannot be solved; if the relay can achieve the second part, using *that* knowledge we can construct an adversary $\mathcal{A}_{PRF}$ that wins the PRF game. □

A direct corollary of the above lemma is that the honest clients will be able to detect any disruption with overwhelming probability.

Corollary 1. *Assuming $\mathcal{F}$ is a secure PRF as well as (almost) key homomorphic with a bounded error e, and H is a collision resistant hash function, only with negligible probability the adversarial relay can disrupt the message x_i of an honest client u_i in round d without getting detected or losing the ability to run later rounds.*

When the Blame protocol is invoked, it is desired that the culprit is identified correctly with overwhelming probability and no honest client can be blamed wrongly for the disruption.

Theorem 3. *Assuming λ_i verifies the correctness of $\mathcal{F}(\mathbf{r}_i, t)$ for a client u_i overwhelming probability, and $\mathcal{S}()$ is cryptographically secure signature scheme, if the* Blame *protocol is run for a disrupted round d, with overwhelming probability at least one disruptive entity is identified, and an honest entity is not (mis-)identified as a disruptor.*

Proof Sketch. If a round is disrupted, the following three possible scenario or a combination of them:

1. at least one client has used bad $\mathcal{F}(\mathbf{r}_i, d\mathsf{N} + t)$ values,
2. at least one setup node has distributed bad $\mathbf{r}_{i,j}$ values,
3. the relay just decided to corrupt the output set.

At least one client has used bad $\mathcal{F}(\mathbf{r}_i, d\mathsf{N} + t)$ values. During the Blame protocol, each client sends their x_i and $\mathcal{F}(\mathbf{r}_i, d\mathsf{N} + t)$ values using a direct channel, along with the proof λ_i of correct computation of the PRF. And the relay recompute the c_i values using the newly received values. Unless λ_i is broken, the client cannot send a wrong $\mathcal{F}(\mathbf{r}_i, d\mathsf{N} + t)$ during value opening. And the client cannot send wrong x_i value because that will yield overall wrong c_i computation.

Since $\mathcal{S}()$ scheme is cryptographically secure, the relay cannot forge a signature for an honest client, and hence, cannot blame an honest client unless the signature verification or λ_i verification fails.

Even if the relay wants to collude with the malicious clients, it has to blame at least one such client; otherwise, the relay is blamed by default.

At least one setup node has distributed bad $\mathbf{r}_{i,j}$ values. In that case, the relay was supposed to verify the following two things during the setup phase:

- for each $\ell \in \{1, 2, \ldots, u\}$ if $\Pi_{i=1}^{\mathsf{N}} \alpha_{i,j,\ell} = g^{\mathbf{s}_\ell}$ holds, where $\mathbf{s} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_u\}$;
- if each $\sigma_{i,j}$ is a valid signature of $(u_i, \Gamma_{\mathbf{r}_{i,j}})$ generated by the setup node G_j.

Since, the signature is cryptographically secure, the clients could not have modified the values on the way. Which means the relay was colluding with the corrupted setup node if it did not flag a failure in verification. Therefore, such an incident gets detected during the setup phase itself, or the relay gets the blame.

The relay just decided to corrupt the output set. If the relay just corrupts the output set, because of the signature verification and λ_i verification the relay cannot blame an honest client. The relay cannot blame a setup node either because of the reasons mentioned above. Hence, the adversarial relay can either blame one of the clients controlled by the adversary, or take the blame for the disruption. □

Lemma 2. *For $z = \mathcal{F}(\mathbf{k}, t) = \lfloor \mathbf{k} \cdot H^u(t) \rfloor_q$ with $\mathbf{k}, H^u(t) \in \mathbb{Z}_v^u$, $\overline{z} = \mathbf{k} \cdot H^u(t)$ and for $e < q$ such that $q\overline{z} = vz + e$, Suppose $\lambda = (g^e, \pi_e)$ where π_e is a zero-knowledge proof that $0 < e < v$. It is computationally difficult to come up with a wrong $\lambda' = (g^{e'}, \pi'_e)$ that will evaluate to the correct verification of z when $g^{\mathbf{k}}$ is known to the verifier.*

Proof. Suppose, a malicious prover sends wrong evaluation of the PRF z', and corresponding $\lambda' = (g^{e'}, \pi'_e)$. Using $\mathbf{k}, H^u(t)$ the verifier can already calculate the following:

$$\begin{aligned}\Pi_{\ell=1}^{u} \gamma_\ell^{h_\ell q} &= \Pi_{\ell=1}^{u} \left(g^{\alpha_{i,\ell}}\right)^{h_\ell q} \\ &= g^{q \sum_\ell \alpha_{i,\ell} h_\ell} = g^{\overline{z} q} \\ &= g^{zu+e} = (g^z)^u \cdot g^e\end{aligned}$$

For $\left(g^{z'}\right)^u \cdot g^{e'}$ to be equal to the above quantity (first step of verification) — Since $z' \neq z$, we can say that $|z - z'_2| \geq 1$; and therefore,

$$\begin{aligned}zu + e = z'u + e' &\iff e' = (z - z'_2)u + e \\ &\iff -u + e \leq e' \leq u + e\end{aligned}$$

In that case, π'_e will fail the range proof for $0 < e' < v$ with overwhelming probability. □

As we already mentioned before, anonymity is broken for that round when the x_i and PRF values are opened during the Blame protocol. However, we do not want that to influence anonymity for any other rounds.

Theorem 4. *Assuming $\mathcal{F}$ is a secure pseudorandom function, and further assuming that at least one of the setup nodes is honest, the* Blame *protocol in round d does not break anonymity for any other round $d' \neq d$.*

Proof. We want to prove the above lemma using contradiction. For contradiction, let us assume that there exist an adversary $\mathcal{A}_{anon}$ that can break the anonymity of OrgAn for some round $d' \neq d$, given that the Blame protocol is run in round d; d and d' can be any arbitrary positive integers chosen by $\mathcal{A}_{anon}$, but less than a finite value T[3].

Here we use a construction similar to the proof of Theorem 2, and construct an adversary $\mathcal{A}_{PRF}$ using $\mathcal{A}_{anon}$. To reiterate the key features of $\mathcal{A}_{PRF}$: our PRF adversary $\mathcal{A}_{PRF}$ will run the whole sender anonymity game as the challenger, except one honest setup node G_j. We force G_j to use an $\mathbf{r}_{1j}$ such that $\mathbf{r}_1 = \mathbf{K}$, where $\mathbf{K}$ is the chosen key for the PRF game. For each slot value $t < T$, $\mathcal{A}_{PRF}$ queries the PRF game with input value t and receives a value f_t, $\mathcal{A}_{PRF}$ asks the client u_1 in the sender anonymity game to use f_t to compute $c_1(t) = \kappa x_i^t + f_t$. Similarly, the client u_2 uses $c_2(t) = \kappa x_i^t + \mathcal{F}(\mathbf{r}_1 + \mathbf{r}_2, t) - f_t$.

One key factor in this game is that $\mathcal{A}_{PRF}$ lets the adversary $\mathcal{A}_{anon}$ adaptively choose a round d when the protocol OrgAn will be disrupted, and a round d' when the challenge message will be sent. In all other rounds including round d, $\mathcal{A}_{anon}$ is allowed to send input messages to the protocol.

According to Theorem 2, the adversarial advantage of $\mathcal{A}_{anon}$ is negligible without any disruption. Our PRF adversary $\mathcal{A}_{PRF}$ returns 1 if and only if $\mathcal{A}_{anon}$ wins the game, otherwise returns 0.

Using a similar line of argument as in the proof of Theorem 2, if $\mathcal{A}_{anon}$ has an non-negligible advantage of δ in the sender anonymity game against OrgAn, there would be a difference of at least δ in the probability $\mathcal{A}_{PRF}$ outputs 1 when f_t is the output of $\mathcal{F}_{rand}()$ vs when it is the output of $\mathcal{F}()$, hence contradicting the security property of the PRF. □

7 Performance Evaluation

In this section we analyze the performance of OrgAn with the help of a prototype implementation[4]. We first describe our implementation details, then present the experimental results.

7.1 Implementation Details And System Optimizations

We developed a proof-of-concept Python implementation of OrgAn. Although the current version of implementation is not optimized for performance and parallelization, it encompasses the complete functionality to enable a successful test-run. We use the Charm [1] library for the different cryptographic operations, and Flint2 [2] for solving powersum equations. For multiplication of ring elements we use the NTT and InverseNTT trasnform (ntt,intt) [43] from the sympy library. Below we briefly discuss the considerations that we make for our implementation.

[3]Consider T as the computational bound of $\mathcal{A}_{anon}$. For a PPT adversary T is polynomially large in the security parameter η

[4]https://anonymous.4open.science/r/organsubmit-D1B1/

Parallelization of Slot Message Computations. The existence of 'slot' is only virtual and all the messages of all the slots per round are computed and forwarded to the relay together by the client. In every round d, each client computes the ciphertext masks for all the slots by computing $\mathcal{F}(\mathbf{r}_i, d)$. Additionally in a Base round, each client computes $\{x_i, x_i^2, \cdots x_i^{\mathsf{N}}\}$ for a message x_i. which takes exactly N multiplications. Therefore, the main computational overhead for a client involves computing the hash values $\mathcal{H}(d) \in R_{\mathsf{v}}$ and the ring product $h() \cdot \mathbf{r}_i \cdot \mathcal{H}()$ to compute $\mathcal{F}(\mathbf{r}_i, d)$. We reduce those overheads using pre-processing as described below.

Preprocessing. We assume that the hash values are available to the clients as a part of the installation. As we already mentioned earlier, we assume that the clients have a preprocessing time everyday before they start using the system; using the available hash values, the clients preprocess the expensive part of PRF computation: $\mathfrak{f} = \mathbf{r}_i \cdot \mathcal{H}()$. During the protocol run, to compute a PRF value a client only needs to compute the multiplication between $\mathfrak{f}$ and $h()$, where $h()$ is the summary of history and thus cannot be preprocessed.

Fragmentation And Defragmentation Of Packets. Once the slot order is decided in a Base round, the clients use the following Bulk round to forward the data packets. The clients transmit η_{p} bits of data per slot in around; the typical Ethernet IP packet (with a Maximum Transmission Unit (MTU) of 1500 bytes) is broken down into multiple fragments of η_{p} bits each and forwarded to the relay in multiple slots. The relay node after computing all the messages, identifies the fragments of each client, forms the full IP packets by combining the fragments and forwards it to the network outside.

Parallelization of Base and Bulk Rounds. The Base and Bulk rounds are not required to be run sequentially, instead they can be run on independent parallel threads by the relay, with only requirement that a Base round corresponding to a Bulk round has run before the Bulk round. For this performance optimization, the protocol requires one minor modification — the computation of PRF in a Base round only includes the history $h()$ of previous Base rounds; however the Bulk rounds still consider the history of both Base and Bulk rounds. In terms of equivocation protection the only difference now is that, if the (adversarial) relay equivocates in a given Bulk round, he will be caught in the next Bulk round instead of the immediate next Base round.

System Parameters. The values of $(\mathsf{p}, \mathsf{q}, \mathsf{v}, \mathsf{u})$ are chosen such that the PRF construction offers at-least 128 bits of security estimated using the lwe-estimator [3, 5]. $(\mathsf{p}, \mathsf{q}, \mathsf{v})$ are of bit-lengths $(\eta_{\mathsf{p}}, \eta_{\mathsf{q}}, \eta_{\mathsf{v}})$ respectively. For the base round, each client chooses a 32 bit random id value for slot selection; hence we use $\eta_{\mathsf{p}} = 32$. In the bulk round, the clients forward 256 bits of data in each slot, with $\eta_{\mathsf{p}} = 256$. The corresponding parameters for base and bulk rounds are:

- Base: $(\eta_{\mathsf{p}}, \eta_{\mathsf{q}}, \eta_{\mathsf{v}}, \mathsf{u}) = (32, 56, 112, 2048)$
- Bulk: $(\eta_{\mathsf{p}}, \eta_{\mathsf{q}}, \eta_{\mathsf{v}}, \mathsf{u}) = (226, 256, 293, 8192)$

The modulus v of the ring used for the base and bulk protocols are $(57 \times 2^{96} + 1)$ and $(7 \times 2^{290} + 1)$ respectively such that $\mathsf{v} \bmod 2\mathsf{u} = 1$. The PRF output is a ring element of order u with coefficients in $\mathbb{Z}_{\mathsf{q}}$ i.e., a vector of length u with each element in $\mathbb{Z}_{\mathsf{q}}$.

7.2 Microbenchmarks

In order to evaluate the overhead and throughput offered by our protocol, we first evaluate the overheads of different steps for the clients and the relay individually. All the benchmarks are average of the measurements after repeating the same experiment 10 times, unless otherwise specified.

7.2.1 Overhead for the clients. The client device computes the PRF values and ciphertexts for any given round. Since the PRF computation is dependent only on the secret shares, it is pre-processed by the client device. Each PRF output is a vector of u elements and hence each such computation can be used for 8192 slots in the bulk round. In one base round, only the values equal to the number of clients is consumed for slot selection.

Preprocessing Overhead for Clients. On an 8-core AWS EC2 t3a.2xlarge instance, for the base round, each PRF computation takes 5.2 milliseconds and for the bulk round it takes 29.32 milliseconds, the difference arising from the orders of the ring elements 2048 and 8192. A total of 34.52 milliseconds of pre-processing for one round of protocol allowing to forward $\frac{8192 \times 226}{100}$ bits for 100 client system, would amount to $\sim$ 4.14 hours of pre-processing for 1 GB of anonymous communication. On a 2015 MacOSX laptop with 4-core Intel i7 process, the PRF computation takes 13.01 and 79 milliseconds respectively with normal usage - amounting to 10.8 hours of pre-processing for 1 GB of anonymous data.

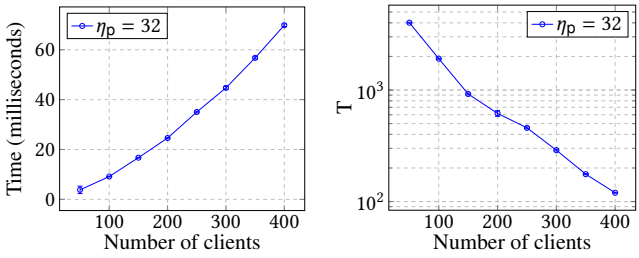
Table 1. Preprocessing and storage overhead for clients evaluated on an 8-core AWS EC2 t3a.2xlarge instance.

Preprocessing	Hash Storage	Setup Data	Anon. Data
34.52 ms	321 KB	863 KB	1 KB
4.14 hours	10 GB	28.67 MB	1 GB

Real-time Overhead for Clients. In the base protocol round, computing each ciphertext takes 2 multiplications and one addition and hence involves $2\mathsf{N}$ multiplications and N additions per round. Thus for a 100 client setting, in one base round each client performs 200 multiplications and 100 additions which takes less 400 microseconds for the η_{p} considered. Thus with pre-computed PRF values, the message generation of the client takes less than half a millisecond for transmission of η_{p} bits of any message. For the bulk round, the client adds the message to the PRF value of the allotted slot and forwards values of all the slots to the relay. If a bulk round involves more than 8192 slots, the required number of slot values are obtained by consuming PRF values in multiples of $\mathsf{u} = 8192$.

7.2.2 Overhead for Relay to Solve Equations System. We use an Amazon AWS EC2 c5.24xlarge instance to run this benchmark for the relay. In every Base round the relay needs to solve the equations system with N equations where N is the number of clients. We choose $\eta_p = 32$ bits as the message size for each message in Base rounds; Fig. 7a shows the time taken by the relay to solve one equation system using a single thread for different number of clients for the chosen η_p. Each equation system corresponds to one base protocol round which is used to send one IP packet for each client in the bulk protocol round.

For 100 clients, the relay solves $\sim$ 1900 equation systems per second and hence can support $\sim$ 1900 packets per client per second. Fig. 7b shows the number of equation systems solved per second by a multi-threaded relay node. For a packet size μ of 1 KB, this corresponds to a throughput of 1900 KBps and 950 KBps if the average packet size is 512 bytes. With increasing clients in the system, the time taken to solve the equation system in the base protocol increases rapidly, reducing the number of total equation systems solved and the throughput.



(a) Time taken (y-axis) by a single threaded relay node to solve the equation system for different number of clients (x-axis).

(b) Number of equation systems (T) solved per second (y-axis) by a multi-threaded relay for different number of clients (x-axis).

Figure 7. Time taken for solving a single equation system and number of such equation systems solved by the relay in the base round with message length $\eta_p = 32$. The values show mean of 100 runs of the protocol.

7.3 End-to-end Latency And Comparison With Other Protocols

Here we evaluate the end-to-end latency offered by our protocol. Additionally, we compare our protocol with PriFi and show that they are comparable in terms of performance. In the PriFi paper [8] paper they already show that PriFi outperforms other existing protocols with significant margins, and therefore, it is sufficient to compare our performance with only PriFi.

Experimental Setup. We use an AWS EC2 c5.24xlarge instance acting as the relay and five EC2 t3a.2xlarge instances simulating all the clients, each instance simulating multiple clients. We run the real-time phase of both OrgAn and PriFi in the same setup. However, we want to note that PriFi has to run its setup phase once for every round to offer same level of anonymity as OrgAn. In our experiments we ignore the overhead of such setup runs[5] in PriFi. Similar to the clients in OrgAn, the relay node also has precomputed hash values, and preprocesses the expensive part of the PRF computation.

Round-trip-time (RTT) of Base Round. In Fig. 8 we show the round trip time (RTT), the time from sending a message to receiving the response message from the relay, in the Base protocol of OrgAn. The time is computed from the last client who sends the message to the time the response message is received from the relay. For a 100 client system, the base protocol introduces a delay of less than 21 milliseconds.

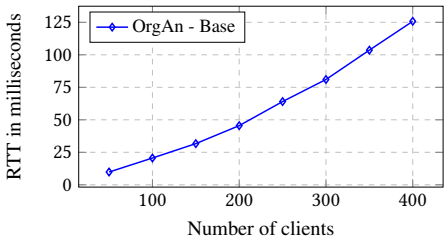


Figure 8. Client-relay-client round trip time (RTT) for the slot selection using the base protocol phase in OrgAn.

End-to-end Latency. We consider that the Base and Bulk rounds of OrgAn are run in parallel threads as mentioned in Section 7.1. Fig. 9 shows the round trip time (RTT) of Bulk rounds of OrgAn while pinging an external server (the IP address google.com), and it is more than the RTT of Base rounds for any number of clients. Therefore we compare the RTT of Bulk rounds in OrgAn with PriFi.

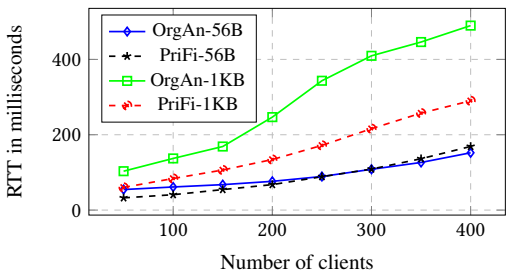


Figure 9. Round trip time (RTT) to ping an external server google.com by forwarding 56 byte and 1KB message using the bulk protocol of OrgAn and PriFi. The suffix in the legend indicates the packet size used.

We measure the round-trip time of the ping experienced by the last client thread that sends the message to the relay. For PriFi we assume that the PRG values from the guard nodes

[5]In order to allow 1 GB of data for each client, assuming an average IP packet size of 1KB, the setup in PriFi needs to be run $\sim 10^6$ times, and an equal number of verifiable shuffle among the guard nodes in PriFi.

for each slot are received by the relay before the round starts. In a 100 client system, PriFi has as RTT of 40 milliseconds and OrgAn has 61 milliseconds when the packets sent by all the clients are small (56 Bytes); when the packet sent by each client is of size 1 KB the RTT becomes 83 and 137 milliseconds respectively.

A delay less than 137 milliseconds per IP packet is typical for supporting audio and video calls. While OrgAn introduces a (slightly) higher latency than PriFi, as discussed earlier, it comes with all the suggested advantages including packet level unlinkability. We note that the performance of OrgAn can be significantly improved with a better implementation that can optimally parallelize the group operations.

Overhead of the Blame **protocol.** In the blame protocol, the relay verifies the PRF outputs of the clients for all the slots in the base protocol and slots for which complaint is raised in the bulk protocol. For a 100 client system, the relay takes $\sim$ 102.686 seconds for all the slots of the base protocol and $\sim$ 2.193 seconds for one disrupted slot in the bulk protocol. The verification process is highly parallelizable across clients and slots.

7.4 Storage Overhead for Hash Data

We assume that the hash values required for the PRF computations are already available to the clients (as part of installation). The amount of hash data used is 112×2048 bits for one base round and 293×8192 bits for one bulk round. With one base round and one bulk round each client can transmit one IP packet. Therefore, if a storage of 10 GB is allocated for hash data, each client can transmit $\frac{10^{10}*8}{(112*2048)+(293*8192)} \approxeq 30422$ IP packets anonymously. Assuming an average size of 1 KB per IP packet, this would correspond to $\sim$ 29.72 MB of anonymous data for each client. This is for a single set of shares from the setup servers.

During the setup phase, the setup servers forward multiple secret sharings of the vector $\mathbf{s}$ instead of a single set of shares to the clients. If they forward 34 such sharings amounting to 9.72 MB of shares per server (and 18.95 MB of proof of correct secret sharing), each client can transmit 1 GB of data anonymously. If each setup server forwards a total of 286.7 MB of shares and commitments, the downloaded hash data (10 GB) is useful to forward 10 GB of anonymous communication. When the allocated budget of anonymous communication is exhausted, the client triggers a setup for re-sharing by the setup servers; the stored hash data is reused after every setup phase. In Table 1 we present the trade-off between the storage requirement and the quota of anonymous data.

For interested readers, we present the evaluation of the time taken for the setup phase in Appendix A.

8 Absentees and Churn

One practical challenge is to deal with unannounced absentees, new employee joining and departures. As we aim to use the pre-processing for the entire day and the setup for the entire week, it is important to deal with absentees and churn. We propose the following modification in our protocol to handle joining or leaving clients. Our technique can be of interest to other DC-net styled protocols.

8.1 Modification In OrgAn Protocol

In the original protocol, each setup server splits $\mathbf{s}$ into N shares and sends them to N clients. If any one client is absent or does not send her ciphertexts in a round, the relay will not be able to decrypt the messages. The system cannot proceed without opening the $\mathbf{r}_i$ value of that client, or running the setup again. To avoid such a problem, each setup server splits $\mathbf{s}$ between two clients, and does the same for each pair of clients. Therefore, each client receives $(\mathsf{N}-1)$ values from each setup server. Similar to the core OrgAn protocol Section 4, the clients sum all the secrets received from all the setup servers to derive its local secret $\mathbf{r}_i$. It is important to note here, if at least two clients and one setup server are honest the adversary cannot guess the $\mathbf{r}_i$ values for the honest clients.

When all N clients participate in the protocol, the real-time protocol remains exactly the same, except the relay uses $\left\lfloor \mathsf{K} \cdot \binom{\mathsf{N}}{2} \cdot \mathbf{s} \right\rfloor$ as the key for the PRF function for decryption.

When Some Clients Are Absent. Suppose only $\mathfrak{N} < \mathsf{N}$ are active in a round. For our protocol to work we have the following requirements: 1. all the clients and the relay know the set of active clients; 2. there are at least two honest active clients. We can represent the system as a completely connected directed weighted graph with each client representing each vertex, the two directed edges between each pair of vertices represent the shared secret between the two clients and the sum of those two edges is $\mathsf{K} \cdot \mathbf{s}$. For notational purposes we assume that the edge directed towards a client represents the sum of the secrets received by that client from the setup servers. To derive the key $\mathbf{r}_i$, the client sums all the edges directed towards her from all the $\mathfrak{N}$ active clients. After that, the real-time protocol remains the same, except the relay uses $\left\lfloor \mathsf{K} \cdot \binom{\mathfrak{N}}{2} \cdot \mathbf{s} \right\rfloor$ as the key for the PRF function for decryption.

Note that the requirement of at least two honest active clients is not easy to achieve if we assume only 2 honest clients out of N. If we relax the assumption and consider at least N_H honest clients, the system can provide anonymity even with only $\mathfrak{N} = (\mathsf{N} - \mathsf{N}_H + 2)$ active clients.

When New Clients Join. When a new client u_{new} joins the system it notifies all the clients, the relay, and all the setup server. For each existing client u_{old}, each setup server splits $\mathbf{s}$ for and distribute the shares with u_{new} and u_{old}. The relay locally updates the value of $\mathfrak{N}$ and the key for the PRF function; each client recomputes the $\mathbf{r}_i$ value; and then the protocol can proceed as usual. The protocol cannot accept new clients if we assume that the setup servers completely disappear after the first setup is done, however, we do not

need regular communication from the setup servers with the clients or the relay.

8.2 Scalability with Client Churn

The method mentioned above has two main limitations as described below.

The communication overhead between a setup server and a client increases by a factor of $(N-1)$. Each client has to always maintain $(N-1)$ keys. In our lattice-based PRF the size of each key is already really huge (a vector of length 256, each element 56 bits, overall around 1.8 KB). With 400 clients in the system, each client has to maintain around 720 KB for the keys. How to reduce that overhead can be an interesting future work.

The requirement of knowing the set of all active clients comes with its own overheads. However, all DC-net based protocols require that all the clients are known in advance, and suffers from this problem. Therefore, we consider this problem orthogonal to our current work. One possible solution to the problem is that each active client sends a heartbeat message before every round, and the relay broadcasts the list of active clients. How to make such a heartbeat protocol work, or in general, how to agree on the list of active clients would be a great future work in the overall DC-net area of research.

9 Discussion

9.1 About (Loss Of) Anonymity During Blame Protocol

When a client to decides to run the Blame protocol, all the information associated with a round is opened, and therefore, anonymity is completely broken. This problem does not only exist in OrgAn but in any other Dc-net based protocol. However, we ensure that this leakage does not affect the anonymity of any other rounds in any way (c.f. Section 6.5).

PriFi tries to defend against this problem by leaking only one bit of information; however, the message bit at the disrupted bit position has to be 0 if the client wants to maintain its anonymity while invoking the Blame protocol. An adversarial relay can always disrupt message bits that are 1 and get away with it. (refer to Appendix C for the detailed analysis).

Even our Bulk protocol can be modified to achieve a similar trade-off using a somewhat similar strategy, where each client sends $g^{\mathcal{F}(\mathbf{r}_i, dN+t)}$ instead of sending x_i and $\mathcal{F}(\mathbf{r}_i, dN+t)$ in plaintext. We detail the protocol in Appendix B.2. If the Base protocol is disrupted it is alright to open all the information associated with the round, since no actual information is transmitted during a Base round. Once the culprit is detected, the rest of the protocol parties can rerun the Base round again.

9.2 Supporting Mobile And Other Lightweight Clients

If we want to run the protocol on mobile and other lightweight devices we can do that by offloading the preprocessing on a laptop or desktop machine. During non-working hours that system can run the preprocessing and store all the preprocessed data in a removable storage (e.g., micro-SD cards[6]). At the beginning of the work hours the removable storage can be put into the mobile device and the mobile device can now use the preprocessed data to run the protocol.

9.3 Application Scenarios

ICRC Scenario. In the beginning of the paper it is established that multinational organizations such as ICRC has a strong need for traffic-analysis-resistant anonymous communication for its delegates. Similar to PriFi, OrgAn can also be deployed in such a scenario. However, OrgAn has a major advantage over PriFi — when each delegation brings their own local trusted server, the communication between the relay and the servers can become a bottleneck in case of PriFi as the relay needs to receive ciphertexts from each of the server for every round. That kind of bottleneck is not present in OrgAn since the servers and the relay do not communicate with each other.

Additionally, OrgAn can handle client churn, it has a huge advantage over PriFi in a conference or seminar setting. Participants can take breaks, participant's system battery might die — OrgAn is equipped to handle such scenarios.

VPN. Two separate organizational networks can setup a site-to-site virtual private network (VPN) between them by utilizing the OrgAn setup. client A of organization X can communicate to client B of organization Y without revealing who is talking to whom. In Figure 10 we demonstrate an example setup.

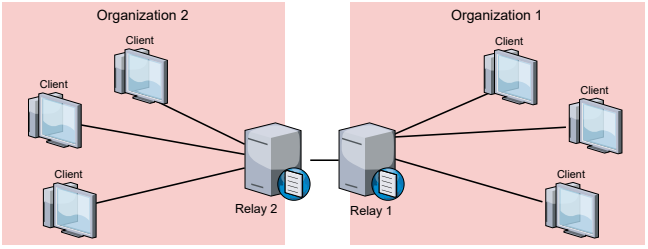


Figure 10. An overview of VPN setup using OrgAn

Similarly a remote access VPN can be implemented using OrgAn where different services inside the organizational network also take part as clients. A remote access client can connect to those services or internal clients without revealing whom it is connecting to. In both the above scenarios (ICRC and VPN), our protocol can be used for different kinds of applications like browsing, DNS queries, VoIP calls, video-conferencing etc.

Mixing Bitcoin Transactions. The Base protocol of OrgAn can be used for mixing Bitcoin transactions exactly in the same way as Dicemix [45]. However, since OrgAn eliminates the necessity to run a key-agreement for every round (which

[6] a standard micro-SD card can store can store up to 128 GB of data.

exists in Dicemix), OrgAn will yield a much faster transaction mixing protocol. As a future work, we plan to integrate OrgAn into CoinShuffle++ protocol [45].

Anonymous Broadcast, Peer-to-peer (P2P) Messaging. A trivial application of our protocol is anonymous broadcast — instead of sending a packet outside, a client can use the organizational broadcast address. An extension of that scenario is P2P messaging, where a client encrypts the message with the public key of the intended recipient; although every other client receives the encrypted message, only the intended receiver can decrypt it. Anonymous group chat, or private chat application can be built using OrgAn with very small end-to-end latency.

Providers In Loopix Type Network Model. The providers in Loopix [41] can be replaced with a setup of OrgAn. That would reduce the dependency and trust on the providers for anonymity, and will always ensure mixing among the smaller subset — which in turn improves the overall anonymity of the Loopix system. The anonymity analysis of Loopix with such a setup would be a great future work.

9.4 Other Related Works

Onion Routing. Onion routing based protocol Tor [24, 25] is designed as an overlay on the internet, and achieves low latency and low bandwidth overhead. However, Tor does not provide enough traffic analysis resistance, as well as not suitable for intra-organization communication.

Mixnets. Many mixnet based protocols [16, 34–36, 49, 50] can offer traffic analysis resistance, but at the cost of high latency overhead (at least several seconds). Although mixnets conceptually can be deployed inside the organizational network for intra-organization communications, the deployment overhead becomes really high (because of the number of mix servers required) to provide strong anonymity guarantees. Systems like Stadium [49], Vuvuzela [50], Karaoke [34] can only provide differential privacy to achieve a meaningful latency overhead (several seconds) and resistance against adversarial packet dropping.

References

[1] . Charm cryptographic library. https://github.com/JHUISI/charm.

[2] . FLINT: Fast Library for Number Theory. https://www.flintlib.org/.

[3] . LWE Estimator. https://bitbucket.org/malb/lwe-estimator/src/master/.

[4] Ittai Abraham, Benny Pinkas, and Avishay Yanai. 2020. Blinder – Scalable, Robust Anonymous Committed Broadcast. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20)*. 1233–1252.

[5] Martin R Albrecht, Rachel Player, and Sam Scott. 2015. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology* 9, 3 (2015), 169–203.

[6] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. 2017. MCMix: Anonymous Messaging via Secure Multiparty Computation. In *Proceedings of the 26th USENIX Security Symposium*. USENIX Association, 1217–1234.

[7] Abhishek Banerjee, Chris Peikert, and Alon Rosen. 2012. Pseudorandom Functions and Lattices. In *Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques* (Cambridge, UK) *(EUROCRYPT'12)*. Springer-Verlag, Berlin, Heidelberg, 719–737. https://doi.org/10.1007/978-3-642-29011-4_42

[8] Ludovic Barman, Italo Dacosta, Mahdi Zamani, Ennan Zhai, Bryan Ford, Jean-Pierre Hubaux, and Joan Feigenbaum. 2017. PriFi: A Low-Latency Local-Area Anonymous Communication Network. *CoRR* abs/1710.10237 (2017). arXiv:1710.10237 http://arxiv.org/abs/1710.10237

[9] K. S. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. C. Sicker. 2007. Low-resource routing attacks against tor. 11–20.

[10] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. 2013. Key Homomorphic PRFs and Their Applications. In *Advances in Cryptology – CRYPTO 2013*, Ran Canetti and Juan A. Garay (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 410–428.

[11] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. 2016. Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting. In *Advances in Cryptology – EUROCRYPT 2016*, Marc Fischlin and Jean-Sébastien Coron (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 327–357.

[12] Jurjen Bos and Bert den Boer. 1990. Detection of Disrupters in the DC Protocol. In *Advances in Cryptology — EUROCRYPT '89*, Jean-Jacques Quisquater and Joos Vandewalle (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 320–327.

[13] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 315–334.

[14] David Chaum. 1981. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM* 4, 2 (1981), 84–88.

[15] David Chaum. 1988. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology* 1, 1 (1988), 65–75.

[16] David Chaum, Debajyoti Das, Farid Javani, Aniket Kate, Anna Krasnova, Joeri de Ruiter, and Alan T. Sherman. 2017. cMix: Mixing with Minimal Real-Time Asymmetric Cryptographic Operations. In *15th International Conference on Applied Cryptography and Network Security 2017*.

[17] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. 2015. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 321–338.

[18] Henry Corrigan-Gibbs and Bryan Ford. 2010. Dissent: Accountable Anonymous Group Messaging. 340–350.

[19] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. 2013. Proactively Accountable Anonymous Messaging in Verdict. In *Proc. 22nd USENIX Security Symposium*. 147–162.

[20] G. Danezis, R. Dingledine, and N. Mathewson. 2003. Mixminion: design of a type III anonymous remailer protocol. In *2003 Symposium on Security and Privacy, 2003.* 2–15. https://doi.org/10.1109/SECPRI.2003.1199323

[21] D. Das, S. Meiser, E. Mohammadi, and A. Kate. 2018. Anonymity Trilemma: Strong Anonymity, Low Bandwidth Overhead, Low Latency - Choose Two. In *2018 IEEE Symposium on Security and Privacy (SP)*. 108–126. https://doi.org/10.1109/SP.2018.00011 extended version under https://eprint.iacr.org/2017/954.

[22] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. 2020. Comprehensive Anonymity Trilemma: User Coordination is not enough. *Proceedings on Privacy Enhancing Technologies* 2020 (07 2020), 356–383. https://doi.org/10.2478/popets-2020-0056

[23] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. 2020. Comprehensive Anonymity Trilemma: User Coordination is not enough. *Proc. Priv. Enhancing Technol.* 2020, 3 (2020), 356–383.

[24] R. Dingledine and N. Mathewson. [n.d.]. Tor Protocol Specification. https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=tor-spec.txt. Accessed Nov 2011.

[25] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13* (San Diego, CA) *(SSYM'04)*. USENIX Association, USA, 21.

[26] Saba Eskandarian, Henry Corrigan-Gibbs, M. Zaharia, and D. Boneh. 2019. Express: Lowering the Cost of Metadata-hiding Communication with Cryptographic Privacy. *ArXiv* abs/1911.09215 (2019).

[27] N. S. Evans, R. Dingledine, and C. Grothoff. 2009. A Practical Congestion Attack on Tor Using Long Paths. 33–50.

[28] Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk. 2008. Universally Composable Security Analysis of TLS. In *Provable Security*, Joonsang Baek, Feng Bao, Kefei Chen, and Xuejia Lai (Eds.). Springer Berlin Heidelberg, 313–327.

[29] Philippe Golle and Ari Juels. 2004. Dining Cryptographers Revisited. In *Proc. of Eurocrypt 2004*.

[30] Henry W Gould. 1999. The Girard-Waring power sum formulas for symmetric functions, and Fibonacci sequences. *Fibonacci Quarterly* 37, 2 (1999), 135–140. https://www.fq.math.ca/Issues/37-2.pdf.

[31] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. 2013. Users get routed: Traffic correlation on Tor by realistic adversaries. In *Proc. ACM SIGSAC conference on Computer & communications security 2013*. 337–348.

[32] Anna Krasnova, Moritz Neikes, and Peter Schwabe. 2016. Footprint Scheduling for Dining-Cryptographer Networks. In *Financial Cryptography and Data Security (FC)*, Jens Grossklags and Bart Preneel (Eds.). 385–402.

[33] Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. 2017. Atom: Horizontally Scaling Strong Anonymity. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 406–422. https://doi.org/10.1145/3132747.3132755

[34] David Lazar, Yossi Gilad, and Nickolai Zeldovich. 2018. Karaoke: Distributed Private Messaging Immune to Passive Traffic Analysis. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 711–725. https://www.usenix.org/conference/osdi18/presentation/lazar

[35] Stevens Le Blond, David Choffnes, William Caldwell, Peter Druschel, and Nicholas Merritt. 2015. Herd: A Scalable, Traffic Analysis Resistant Anonymity Network for VoIP Systems. In *Proc. ACM Conference on Special Interest Group on Data Communication (SIGCOMM 2015)*. 639–652.

[36] Stevens Le Blond, David Choffnes, Wenxuan Zhou, Peter Druschel, Hitesh Ballani, and Paul Francis. 2013. Towards Efficient Traffic-analysis Resistant Anonymity Networks. In *Proc. ACM SIGCOMM 2013*. 303–314.

[37] Stevens Le Blond, Alejandro Cuevas, Juan Ramón Troncoso-Pastoriza, Philipp Jovanovic, Bryan Ford, and Jean-Pierre Hubaux. 2018. On enforcing the digital immunity of a large humanitarian organization. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 424–440.

[38] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. 2019. 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*. 887–903.

[39] M. Backes, A. Kate, P. Manoharan, S. Meiser, and E. Mohammadi. 2016. AnoA: A Framework For Analyzing Anonymous Communication Protocols. *Journal of Privacy and Confidentiality (JPC)* 7(2), 5 (2016). Issue 2.

[40] USA Office of the Director of National Intelligence (ODNI). 2021. GLOBAL TRENDS 2040. https://www.dni.gov/index.php/gt2040-home/

[41] Ania Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. 2017. The Loopix Anonymity System. In *Proc. 26th USENIX Security Symposium*.

[42] David Pointcheval and Jacques Stern. 2001. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology* 13 (10 2001). https://doi.org/10.1007/s001450010003

[43] John M Pollard. 1971. The fast Fourier transform in a finite field. *Mathematics of computation* 25, 114 (1971), 365–374.

[44] RFC 8446. [n.d.]. The Transport Layer Security (TLS) Protocol Version 1.3. https://tools.ietf.org/html/rfc8446. Accessed April 2021.

[45] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. 2017. P2P Mixing and Unlinkable Bitcoin Transactions. In *Proc. 25th Annual Network & Distributed System Security Symposium (NDSS)*.

[46] C. P. Schnorr. 1990. Efficient Identification and Signatures for Smart Cards. In *Advances in Cryptology — CRYPTO' 89 Proceedings*, Gilles Brassard (Ed.). Springer New York, New York, NY, 239–252.

[47] Craig R Scott and Stephen A Rains. 2005. Anonymous communication in organizations: Assessing use and appropriateness. *Management Communication Quarterly* 19, 2 (2005), 157–197.

[48] Skype Support. [n.d.]. How much bandwidth does Skype need? https://support.skype.com/en/faq/FA1417/How-much-bandwidth-does-Skype-need. Accessed April 2021.

[49] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. 2017. Stadium: A Distributed Metadata-Private Messaging System. 423–440. https://doi.org/10.1145/3132747.3132783

[50] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. 2015. Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis. In *Proc. 25th ACM Symposium on Operating Systems Principles (SOSP 2015)*.

[51] Luis von Ahn, Andrew Bortz, and Nicholas J. Hopper. 2003. K-Anonymous Message Transmission. In *Proceedings of the 10th ACM Conference on Computer and Communications Security* (Washington D.C., USA) *(CCS '03)*. Association for Computing Machinery, 122–130.

[52] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. 2018. Doubly-Efficient zkSNARKs Without Trusted Setup. In *Proceedings - 2018 IEEE Symposium on Security and Privacy, SP 2018 (Proceedings - IEEE Symposium on Security and Privacy)*. Institute of Electrical and Electronics Engineers Inc., 926–943. https://doi.org/10.1109/SP.2018.00060

[53] Michael Waidner. 1990. Unconditional Sender and Recipient Untraceability in Spite of Active Attacks. In *Advances in Cryptology — EUROCRYPT '89*, Jean-Jacques Quisquater and Joos Vandewalle (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 302–319.

[54] Michael Waidner and Birgit Pfitzmann. 1990. The Dining Cryptographers in the Disco: Unconditional Sender and Recipient Untraceability with Computationally Secure Serviceability. In *Advances in Cryptology*

— *EUROCRYPT '89*. 690–690.

[55] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. 2012. Dissent in Numbers: Making Strong Anonymity Scale. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. 179–182.

A Overhead Evaluation Of Setup Phase

Since we use different message size for Base and Bulk rounds, each setup server shares different secrets of different ring size for Base and Bulk rounds. The time taken by a setup-server run on EC2 3a.2xlarge instance for generation of the shares and proof of correct share generation related to the base and bulk protocol rounds can be found in Table 2 and Fig. 11.

Table 2. Time taken per client in seconds by the setup-server to generate commitments of shares and signature for base and bulk protocol rounds

	Commitments generation	Serialize and sign
Base protocol	2.55487	0.075846
Bulk protocol	21.89879	0.460579

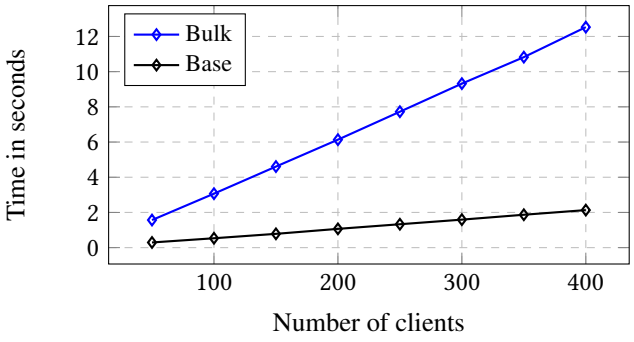


Figure 11. Time taken to generate one set of shares for clients by a setup-server related to Base and Bulk rounds of OrgAn.

B More About Blame Protocol

B.1 Verifiability Of PRF

Recall that the PRF values were generated by first performing ring product as element wise multiplication of the NTT transforms of $\mathbf{k}$ and $\mathcal{H}(\cdot)$, and then applying inverse NTT transform on the output from the first step.

More specifically, let the NTT transforms of $\mathbf{r}_i$ be $\hat{\mathbf{r}}_i = \{\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{i\mathsf{u}}\}$. and of $\mathcal{H}(\cdot)$ be $\hat{\mathbf{H}} = \{h_1, h_2, \ldots, h_{\mathsf{u}}\}$; also let $\mathbf{L}_i = \{\ell_{i1}, \ell_{i2}, \cdots, \ell_{i\mathsf{u}}\}$ denote the element-wise product of $\hat{\mathbf{H}}, \hat{\mathbf{r}}_i$. Then we have, $\ell_{i,k} = \alpha_{i,k} \cdot h_k$. Let the Inverse NTT of $\mathbf{L}_i$ is denoted by $\mathbf{W}_i = \{w_1, w_2, \cdots, w_{\mathsf{u}}\}$. Then we have: $w_{i,j} = \mathsf{v}^{-1}\gamma^{-j} \sum_{k=0}^{\mathsf{u}-1} \ell_{i,k}\Omega^{-jk} \bmod \mathsf{v}$, where $\gamma = \sqrt{\Omega} \bmod \mathsf{v}$ with Ω the u-th primitive root of unity. The final PRF output set is obtained as $\lfloor \mathbf{W}_i \rfloor_{\mathsf{p}} = \{\lfloor w_{i,1} \rfloor_{\mathsf{p}}, \lfloor w_{i,2} \rfloor_{\mathsf{p}}, \cdots, \lfloor w_{i,\mathsf{u}} \rfloor_{\mathsf{p}}\} = \{z_{i1}, z_{i2}, \cdots, z_{i\mathsf{u}}\}$. Therefore, the following holds for each index $1 < k < \mathsf{u}$: $\mathsf{q}z_{ik} = \mathsf{p}w_{i,k} + e$ for some $0 < e < \mathsf{v}$.

Below we describe the steps to enable the relay to verify the correctness of the PRF computations during the Blame protocol run:

Additional Steps During Setup Phase. Let all the setup servers and the relay mutually agree on a generator g of a multiplicative group G of order $\tau > \mathsf{qv}$. Then each setup server G_j computes $\rho_{\mathbf{r}_{ij}} = \{\mathsf{g}^{\alpha_{ij1}}, \mathsf{g}^{\alpha_{ij2}}, \ldots, \mathsf{g}^{\alpha_{iju}}\}$ and a signature $\sigma_{ij} = \mathcal{S}_{p_{G_j}}(u_i, \rho_{\mathbf{r}_{ij}})$, and sends them to client u_i. The client verifies the signature and the computation of $\rho_{\mathbf{r}_{ij}}$, and forwards them to the relay. After receiving the $\rho_{\mathbf{r}_{ij}}$ values from each client, the relay verifies the signatures as well.

Client Side Computation. During the blame protocol, a client computes g^e, g^e and two non-interactive zero knowledge proof (NIZKP): a range proof [11, 13, 52] π_e verifying $0 < e < \mathsf{v}$ in g^e, and a NIZKP [**?**] ι_e verifying that the exponent e is same in both g^e and g^e (recall that $g \in \mathbb{Z}_{\mathsf{v}}$). Then the client sends as a NIZKP for correct PRF computation, $\lambda = (g^e, \mathsf{g}^e, \pi_e, \iota_e)$ to the relay R, without revealing e.

Relay Side Computation. The relay R can compute the values $\{\mathsf{g}^{\alpha_{i1}}, \ldots, \mathsf{g}^{\alpha_{i\mathsf{u}}}\}$ using the values it received during the setup phase. Using the above information, the relay can compute:

$$\mathsf{g}^{w_{i,k}} = \left(\prod_{j=0}^{\mathsf{u}-1} (\mathsf{g}^{\alpha_{ij}})^{\gamma^{-k} \cdot h_j \cdot \Omega^{-kj} \cdot \mathsf{v}^{-1}} \right).$$

Note here, $\mathsf{v}, \gamma^{-k}, \Omega^{-kj}, h_j$ are public information.

To verify that a PRF value z_{ik} is correct, the relay needs to verify $\mathsf{g}^{\mathsf{q}z_{ik}} = \mathsf{g}^{\mathsf{p}w_{i,k}} \cdot \mathsf{g}^e$. Additionally, the relay needs to verify the NIZKPs π_e and ι_e.

B.2 Another Version of Blame Protocol With Only 1-bit Leakage

We modify our Base protocol to include an additional flag bit that tells the relay to run the Blame protocol for the last Bulk round — a client can initiate the Blame protocol anonymously by setting the flag bit to 1. As the content of the message in Base protocol, the client sends the slot index and bit index for that slot that he wants to open, instead of a random number. Similar to PriFi, the client invokes the Blame protocol for a bit position where the original message bit value was 0.

This allows the clients to invoke the Blame protocol without revealing their identities if a Bulk round is disrupted. We do not need to change the Blame protocol if the Base round is disrupted, since no actual information is transmitted in a Base round — the clients can just rerun the Base round and use the new slot agreement for the next Bulk round.

Suppose, the Blame protocol is invoked for bit position ℓ. In our new Blame protocol, a clients does not send the x_i and $\mathcal{F}(\mathbf{r}_i, d)_t$ values to the relay; instead sends the ℓ-th bit of $\mathcal{F}(\mathbf{r}_i, d)_t$ and a tuple (g^{ψ_1}, g^{ψ_2}) where ψ_1 is the first $(\ell - 1)$ bits of $\mathcal{F}(\mathbf{r}_i, d)_t$ and ψ_2 is the last $(q - \ell)$ bits of

$\mathcal{F}(\mathbf{r}_i, d)_t$. Additionally, the client sends λ_i, a non-interactive zero knowledge range proof for correct PRF evaluation (refer to Appendix B.1 for more details). This allows the client to reveal the ℓ-th bit of $\mathcal{F}(\mathbf{r}_i, d)_t$ without revealing the value, and allows the relay to verify that the bit is indeed the ℓ-th bit of correct PRF value. If a client fails to provide the above values, the relay blames that client as the culprit. If no client is detected as the culprit, by default, the relay is blamed. Recall that, the correctness of the shares from the setup servers are verified during the setup phase itself.

The setup phase remains same as our original Blame protocol in Section 5. The exact same PRF verification method and equivocation protection strategy works here as well.

We do not provide a formal security analysis for this new Blame protocol because an adversarial relay can always choose to disrupt a bit position where the message bit is 1. Then we are again back to the trade-off: compromise anonymity to punish the adversary or let the adversary get away with it.

C Additional Anonymity Analysis for PriFi

Here we present some lemma proving some of the (lack of) security properties of PriFi. Although these (or similar) properties are common across many DC-net protocols, formally stating these properties helps us compare PriFi with our protocol as well as other protocols in the literature.

The following lemma shows that PriFi cannot be run for multiple anonymize runs with the same permutation, if unlinkability across multiple runs are required. The same is true for any other a symmetric encryption based DC-net [8, 15, 18, 19, 55].

Lemma 3. *There exist a global passive adversary $\mathcal{A}$ that has adversarial advantage $\delta = 1$ in the sender anonymity as defined in Definition 5 against the protocol PriFi with multiple runs of* anonymize *and one setup run.*

Proof. Recall the AnoA game, in which the adversary can send some input messages to observe the protocol run, and then sends the challenge message to be sent by either Alice or Bob. The adversary needs to figure out who among Alice and Bob has sent the challenge message.

Let us construct an adversary $\mathcal{A}^*$ that can achieve $\delta = 1$ against the protocol PriFi when there are multiple runs of *anonymize* after one setup run. Our adversary $\mathcal{A}^*$ sends an input message x to be sent by Alice in the first anonymize run. $\mathcal{A}^*$ observes the position π_x of message x in the output messages. The adversary $\mathcal{A}*$ chooses (Alice, Bob) as the challenge senders and sends the challenge message in the second anonymize run. $\mathcal{A}^*$ checks if the challenge message is at position π_x or not — if it is at the position π_x, $\mathcal{A}^*$ outputs 0, otherwise 1.

Since, the permutation is not changed between first and second anonymize runs, $\mathcal{A}^*$ always wins. □

C.1 Anonymity vs. Performance Tradeoff In PriFi

Somewhat similar to OrgAn, PriFi also has a setup phase to improve performance of its real-time phase, which they call *anonymize*. When only one round of anonymize is executed after one round of setup, PriFi protocol provides strong (negligible adversarial advantage) sender anonymity, in the presence of a passive network level adversary.

However, when the anonymize phase is run for multiple rounds after only one setup (as recommended by the authors of PriFi), every round of anonymize uses the same permutation. And hence, all the messages from the same user can be linked together — the anonymity property is broken (refer to Appendix C for a formal proof). To consider a real world example, if the relay (controlled by the organization) observes that the same user (but the identity is unknown) is trying to contact multiple journalists it might suspect a whistle-blowing, and use additional physical means to find out that user or even disrupt the relay service itself.