Debajyoti Das*, Easwar Vivek Mangipudi*, and Aniket Kate

OrgAn: Organizational Anonymity with Low Latency

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 introduce high latency overhead, provide weaker anonymity guarantees, or are difficult to deploy 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 lattice-based (almost) key-homomorphic pseudorandom function 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 to overcome several major problems with any realistic PriFi instantiation: (a) unlike PriFi, OrgAn avoids frequent, interactive, slot-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. We demonstrate how to make this public-key cryptographic solution scale equally well as the symmetric-cryptographic PriFi with practical pre-computation and storage requirements. Through a prototype implementation, we show that OrgAn provides similar throughput and end-to-end latency guarantees as PriFi, while still discounting the setup challenges in PriFi.

Keywords: privacy, anonymity, protocol

DOI Editor to enter DOI
Received ..; revised ..; accepted ...

***Corresponding Author: Debajyoti Das:** imec-COSIC, KU Leuven, Leuven, Belgium, E-mail: debajyoti.das@esat.kuleuven.be,
***Corresponding Author: Easwar Vivek Mangipudi:** Department of Computer Science, Purdue University, West Lafayette, USA, E-mail: emangipu@purdue.edu,
Aniket Kate: Department of Computer Science, Purdue University, West Lafayette, USA, E-mail: aniket@purdue.edu

1 Introduction

In an influential work, Le Blond et al. [34] recognize an urgent need for traffic-analysis-resistant meta-data hiding (anonymous) communication at multinational organizations such as the 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 and powerful state-sponsored network eavesdroppers, there is a clear demand for anonymity for intra-organization communication and their interactions with the global services. As a recent US national Intelligence Council global trends report [38] indicates we are moving towards a more contested world post-pandemic, and anonymity needs at the international organizations [43] are bound to grow.

Among different anonymous communication (AC) protocols, dining-cryptographers network (DC-net) [13] and its successors [3, 5, 10, 16, 17, 26, 29, 35, 41, 46–49] are suitable for this purpose, for their high traffic-analysis resistance while maintaining low latency overhead. Tor [22, 23] and mixing network-based protocols [12, 14, 18, 30–33, 44, 45] choose a tradeoff between low latency and stronger anonymity [19]; and they are not suitable for organizational networks as they 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.

Nevertheless, most standard DC-net designs have two major overheads: (i) First, all the users need to run a key agreement protocol among themselves to agree on shared secret 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.

With the above organizational anonymity problem in mind, PriFi [8] introduces a novel client/relay/server model that avoids the key agreement overhead — a set of few servers (we call them *setup servers*) help the clients establish shared secrets among themselves. The actual anonymous communication happens through another relay node and the servers only need to push messages to this relay node. As its key features,

PriFi removes all server-to-server communications from the latency-critical path. As demonstrated by Barman et al. [8], the client/relay/server model is well-suited for organizational anonymity; however, we observe three key problems with the current cryptographic design[1]: (i) PriFi continues with the one user per slot model from Dissent [16] and its successors [49], and a PriFi instantiation has to choose between running a regular slot selection protocol (where the servers run expensive verifiable shuffle and communicate with all the clients) and message linkability across different rounds. (ii) While the servers do not talk to each other in the online phase, they still need to be online and communicate with the relay node. This can allow any server to predict the typical PriFi usage pattern at the organization. (iii) Finally, every server and the relay has to know and interact with each other, which may facilitate coercion and collusion (possibly even in the future).

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-mentioned three 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 perfect shuffling 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, and OrgAn servers are omitted from the online phase; thus, we call them setup servers. In fact, the setup servers and the relay nodes do not even need to be aware of each others' presence, which can be particularly useful against coercion and collusion.

While providing the above features using a public key cryptographic primitive (i.e., key-homomorphic PRF), OrgAn also maintains latency overhead in milliseconds using 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 46 milliseconds for a system of 100 clients when communicating to the outside world compared to a typical RTT of 18 milliseconds. Our evaluation demonstrates that OrgAn can even scale for latency-sensitive and throughput demanding applications — it can support up to 550 Kbps throughput for every client in an organization with 100 clients. We also compare the performance of OrgAn with the existing state-of-art (PriFi) and show that performance-wise OrgAn is not very different from PriFi, despite solving the issues mentioned above.

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 and connect to services outside the organizational network without revealing which client is accessing which service.

There exists a *relay* R that acts as a gateway between the organizational network and the outside world. It allows the clients of the OrgAn protocol to transmit/receive messages to/from the outside would and intra-organization services but does not act as a trusted third party in the anonymization process. We assume that the relay has high availability and high computation power.

Additionally, there is a set of K servers $G_1, \ldots, G_{\mathsf{K}}$, we call them *setup* servers. These setup servers help the clients in the setup or key agreement process but do not participate in the anonymization process. These servers are logical entities and can be run by anyone (volunteers, paid services) inside or outside the organization, even by the clients.

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 even require the setup servers to be online except for the setup phase. We only focus on the anonymity of outgoing traffic. We consider the solution provided by PriFi for incoming traffic adequate and can easily fit into our system; hence, we do not consider incoming 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 up to $(\mathsf{N}-2)$ clients and $(\mathsf{K}-1)$ setup servers. Even the relay can be under adversarial control, but

1 For reference, we add a short description of PriFi in Appendix B.

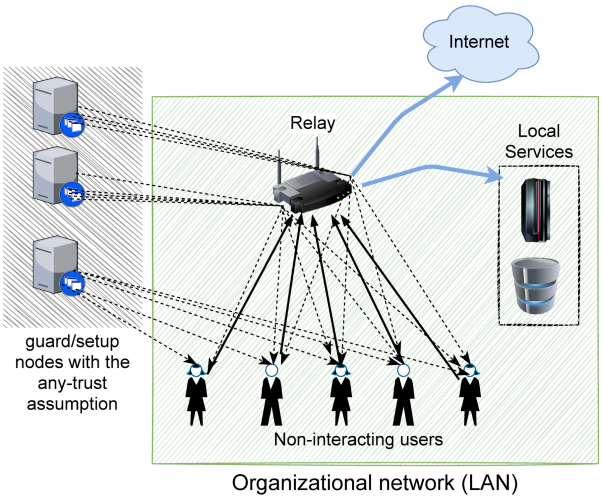


Fig. 1. System overview of OrgAn, where a set of non-interacting users wants to access local services and the Internet through a relay server but without revealing who is accessing what. A group of setup servers help them establish DC-net keys so that the users can run a DC-net protocol using Newton's power-sum method utilizing the relay as a bulletin-board, but without trusting the relay.

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 the privacy of their members. However, the relay may launch other active attacks as long as it can not be detected/blamed by the honest clients.

We allow the adversary to launch active attacks from inside the organizational network. For simplicity of description, we do not consider any active attacks 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 *strong sender anonymity* for the outgoing traffic, i.e., the ability of the adversary to figure out which user has sent a specific message should at most be negligibly better than random guessing, even if all but two clients and all but one setup server are compromised.

Accountability. In our protocol, we want the honest clients to be able to detect the scenario where some malicious client(s) or the relay tries to disrupt the protocol by sending incorrect/malformed messages. In such an event, the disruptive party should be identified with significant probability, and no honest party should be misidentified as a malicious party.

Low Latency. The system should operate with low latency overhead to support latency-sensitive applications.

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

2.3.1 Non-goals

Similar to other DC-net based protocols [8, 16, 17, 41, 49], our protocol does not consider an adversary whose sole purpose is to launch DoS attacks. In fact, this remains a challenging issue for all AC systems in general. Nevertheless, similar to most other systems, the DoS attacks are detectable and can result in at least one faulty node being detected and dropped.

Similar to other DC-net protocols [5, 8, 16, 17, 49], even OrgAn does not provide a solution for reliable packet delivery. If packet loss occurs between the user and the relay, the receiving end can request the packet again after a timeout.

Moreover, as with other DC-net based protocols [8, 16, 41], it is out of scope to consider clients joining or leaving the network between two setups.

2.4 Protocol Idea

Our protocol design is based on the idea that each client u_i has an individual secret share $\mathbf{r}_i$ — here $\sum_i \mathbf{r}_i$ is known to everyone, but the individual $\mathbf{r}_i$ 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 server, the setup guarantees can be achieved using an additive secret sharing scheme between the setup servers and the clients.

In our protocol design, we use an almost key-homomorphic PRF [9] $\mathcal{F}$ that satisfies $\mathcal{F}(\mathbf{k}, d) = \mathcal{F}(\mathbf{k}_1, d) + \mathcal{F}(\mathbf{k}_2, d) + e$ 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 $\mathcal{F}\left(\sum_i \mathbf{r}_i, d\right)_t$. We use some extra bits to eliminate the error e so the relay can retrieve the message correctly.

The main rationale behind using an *almost* key-homomorphic PRF instead of a perfectly key-homomorphic PRF [6] is that the latter is computationally expensive. Using a lattice-based additive *almost* key-homomorphic PRF we improve the performance drastically (cf. Section 8) as compared to the (perfectly key-homomorphic) PRF.

Base Protocol. We solve the slot agreement problem generally faced by DC-net based protocols by constructing power-sums of messages similar to Dicemix [41]. However, we do it

(significantly) more efficiently than Dicemix by exploiting the client/relay/server network model and the (almost) key homomorphic property of $\mathcal{F}$. We describe our protocol in Section 4. However, there is one crucial problem with the power-sum equation system — with a large number of clients, solving the equation system becomes the bottleneck of the whole system. Additionally, for each IP packet (e.g., of 512 bytes length), the relay should be able to link all the *fragments*, considering each round allows each client 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 [16] to solve the above two problems. First, OrgAn uses the power-sum equation system in one round to generate a permutation for the clients. In the following round, the clients use that permutation (slot agreement) to run a typical DC-net. The clients still generate the masks for the ciphertexts using $\mathcal{F}$. The bulk protocol provides unlinkability for IP packets, but not the fragments of a single IP packet (by design).

2.5 What We Achieve

In our design, we solve two latency-critical problems inherent in most DC-net designs: (i) We get rid of the slot agreement overhead (that is present also in PriFi) by using the Dicemix-like power-sum equation system [41]; (ii) We avoid the requirement for regular key-agreement in other DC-net designs, using key-homomorphic PRFs. Even though we use public-key cryptographic constructions to solve those, we demonstrate that parties can push most computation-heavy tasks to the pre-processing phase. By getting rid of these overheads, we achieve a round-trip-time (RTT) of 46 milliseconds for a system of 100 clients when communicating to the outside world compared to a typical RTT of 18 milliseconds. We demonstrate that OrgAn can scale for latency-sensitive and throughput demanding applications by utilizing several hours of pre-processing per day.

We formally show that OrgAn provides strong sender anonymity against global passive adversaries (in Section 6.2). We also prove the security of our protocol against relevant active attacks (in Section 6.3) — we show that honest clients can detect such active attacks with overwhelming probability and identify a malicious party with significant probability.

2.6 Comparison with Relevant Protocols

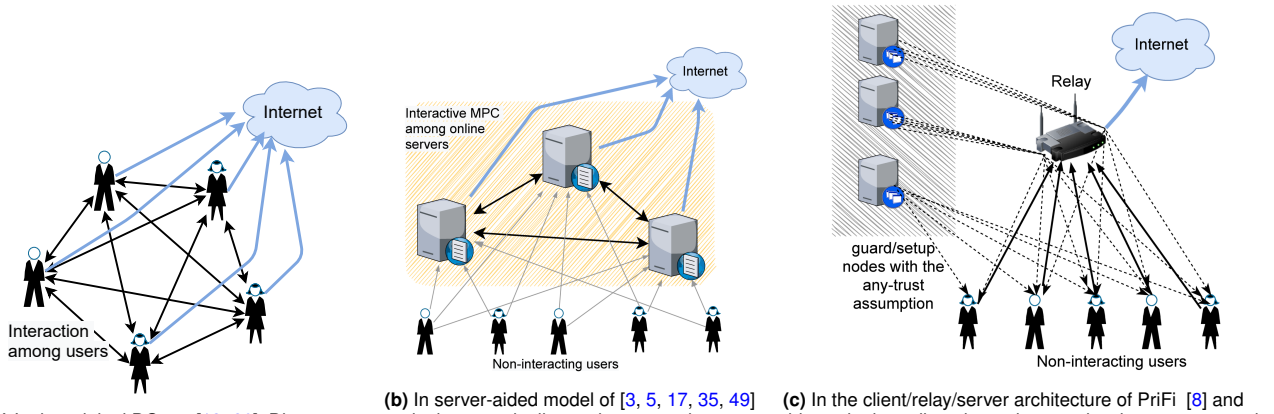
Onion routing systems such as the Tor network [22, 23] and typical deployment of mixnet designs [12, 14, 18, 30–33, 44, 45] are not suitable for the organizational network or the LAN setting under consideration here as they route all the traffic over the Internet outside the network and can introduce high latency overhead. 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. Moreover, Das et al. [20] shows that protocols inspired from dining cryptographers' network (DC-net), with co-ordination among the users, 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. 2, 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 our protocol OrgAn with prominent DC-net based solutions across these architectures.

P2P DC-nets. In P2P DC-net protocols [16, 26, 41], the clients perform the anonymous broadcast themselves without involving any external computational server. These designs incur significant overhead (computation, as well as bandwidth) for key agreement and slot agreement. For example, DiceMix [41] takes several seconds to complete a protocol run for a small number of clients. OrgAn can be considered as an overhaul of DiceMix [41], where we remove the interaction among clients by utilizing the client/relay/server model.

Client-server DC-nets. P2P DC-nets do not scale well as the numbers of clients increase. Client-server DC-net protocols [3, 5, 15, 17, 24, 35, 49] 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 to 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 geopolitically 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. 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 round. Thus, PriFi achieves high throughput with com-



(a) In the original DC-net [13, 26], Dissent [16], Dicemix [41] 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 [3, 5, 17, 35, 49] and others, each client only communicates with a set of servers, which interact with each other (performing a multi-party computation) 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; in this work, we demonstrate that the servers also do not need to communicate among themselves. Based on the network setting, the relay node may or may not know the servers

Fig. 2. Illustrative Examples for Different Architectures for DC-net Inspired Protocols. The Internet cloud in these examples can be conveniently replaced by any bulletin board or broadcast channel.

promised 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 setup 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.

3 Preliminaries

In this section we describe some building blocks that we use in our protocol.

Power-sum Equations and Solution[27, 41]. 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 [27, 41]. 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 to find an unordered set of client messages.

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 $R_v \times R_q$, $v > q$ obtained by choosing $\mathbf{a} \leftarrow R_v$ uniformly at random for some $\mathbf{s} \in R_v$ 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 $(\mathbf{a}_i, \mathbf{u}_i) \in R_v \times R_q$.*

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 $\mathbf{a} \sim U(R_v), \mathbf{s} \sim U(R_v)$, and $\mathbf{u} \sim U(R_q)$.

Almost Key-homomorphic Pseudorandom Function. A key-homomorphic pseudo random function family (PRF) [9] is a PRF which is homomorphic in the key-input of the function. Our protocol uses the lattice based pseudorandom function family [9] $\mathcal{F}(\mathbf{k}, d) = \lfloor \mathbf{k} \cdot \mathcal{H}(d) \rfloor_{\mathsf{q}} \in R_q$, for $\mathbf{k}, \mathcal{H}(d) \in R_{\mathsf{v}}$, $d \in \{0,1\}^*$. 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., $\mathcal{F}(\mathbf{k}, d) = \mathcal{F}(\mathbf{k}_1, d) + \mathcal{F}(\mathbf{k}_2, d) + \epsilon$ where $\epsilon \in R_{\mathsf{v}}$ with each coefficient of $\epsilon \in \{0, 1\}$. The security of the PRF considered here is based on the above Ring-LWR assumption.

For any element $a \in \mathbb{Z}_{\mathsf{v}}$, the rounding-down function $\lfloor \cdot \rfloor_{\mathsf{q}}$ is defined as $\lfloor a \rfloor_{\mathsf{q}} = \lfloor a \cdot \frac{\mathsf{q}}{\mathsf{v}} \rfloor \in \mathbb{Z}_{\mathsf{q}}$. 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 $c_1 = \kappa \cdot x_1 + \mathcal{F}(\mathbf{k}_1, d)_t$ and $c_2 = \kappa \cdot x_2 + \mathcal{F}(\mathbf{k}_2, d)_t$ for a certain d, t and suitable positive integer κ. The relay computes $c = c_1 + c_2 - \mathcal{F}(\mathbf{k}, d)_t = \kappa x_1 + \kappa x_2 + e$. It computes $\frac{c}{\kappa}$ and rounds the value to the nearest integer to obtain $x_1 + x_2$. Here, $\mathcal{F}(\mathbf{k}, d)_t$ is the t^{th} element of the vector $\mathcal{F}(\mathbf{k}, d)$. We refer to Section 4.2 for details on the elimination of the introduced error.

Additionally, our protocol uses a digital signature scheme [42] that is existentially unforgeable under chosen-message attacks [39]. 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 the function $\mathcal{V}_P(x, \sigma)$ anyone can verify the signature.

4 Core Protocol

In this section, we present the core OrgAn protocol, and in Section 5 we extend our protocol to defend against active attacks. As mentioned in Section 2.1, our system consists of the following set of parties: (i) 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; (ii) a set of K setup servers denoted as $G_1, \ldots, G_{\mathsf{K}}$ that reside outside the organizational network; (iii) 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 in Table 1.

Our protocol first runs a one-time setup phase, and then starts running the protocol. In our protocol, the setup phase is run only once and never again.

4.1 Setup Phase

Each setup server G_j splits a publicly known constant $\mathbf{s}$ into N secret shares $\{\mathbf{r}_{1j}, \ldots, \mathbf{r}_{\mathsf{N}j}\}$ such that $\mathbf{s}, \mathbf{r}_{ij} \in \mathbb{Z}_{\mathsf{v}}^{\mathsf{u}}$ and $\sum_i \mathbf{r}_{ij} = \mathbf{s}$. It distributes the shares among the clients, where each client u_i receives the share $\mathbf{r}_{ij}$. Note here, all the setup servers use the same $\mathbf{s}$ value.That is a global parameter of the protocol; however, the shares for the clients generated by each honest setup server are independent of other servers and unknown to other servers. We assume that the setup servers communicate with the clients using some authenticated and confidential channel (for example, using TLS [25, 40]).

Table 1. Protocol and system parameters for OrgAn

$\mathcal{U}$	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 setup server
R	The relay node
η	The security parameter
$a \xleftarrow{\$} [b, c]$	Draw uniformly at random from an integer range $[b, c]$
$\mathbb{Z}_{\mathsf{p}}, \mathbb{Z}_{\mathsf{q}}, \mathbb{Z}_{\mathsf{v}}$	Groups 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 $\{0,1\}^\eta$
$\mathcal{F}(\mathbf{k}, t)_i$	i^{th} element of the PRF obtained from key $\mathbf{k}$ for slot t

$\mathbf{s} \in \mathbb{Z}_{\mathsf{v}}^{u}$; 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 $u_i \in \mathcal{U}$

Fig. 3. Setup protocol in OrgAn

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: $\mathbf{r}_i = \sum_{j=1}^{m} \mathbf{r}_{ij} \in \mathbb{Z}_{\mathsf{v}}^{u}$. We present the pseudo-code for the setup run by the setup servers and the clients in Figure 3.

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 a 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 a round d:

$\mathbf{s} \in \mathbb{Z}_{\mathsf{v}}^{u}$: 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**
$P_t = P_t + c_i(t) \mod \mathsf{q}$
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**
$\mathsf{packet}_i = x_{t+1} || \cdots || x_{t+t_i}$; $t = t + t_i$
Send $\mathsf{packet}_1, \ldots, \mathsf{packet}_{\mathsf{N}}$ to internet
Broadcast $(\mathsf{packet}_1, \ldots, \mathsf{packet}_{\mathsf{N}})$

ClientProtocol(client u_i, packet M, round d) :
if $(d\%2 = 1)$ // Base round, $T = \mathsf{N}$ // **then**
$\nu_i =$ 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**
$c_i(j) = \mathsf{GenCipher}((x_i || \nu_i), d, j)$
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 $!\exists (x_{\pi_j}, t_j) : (x_{\pi_j}, t_j) = (x_i, \nu_i)$ **then**
Run Blame protocol
else
$\mathfrak{T} = t_1 + \cdots + t_{j-1}, T = t_1 + \cdots + t_{\mathsf{N}}$
else
// Bulk round, $T = t_1 + \ldots t_{\mathsf{N}}$ //
$x_{i1}, \cdots, x_{i\nu} =$ Split packet M into ν_i fragments
for each $t \in \{1, \ldots, T\}$ **do**
if $\mathfrak{T} < j \leq \mathfrak{T} + \nu$ **then**
$c_i(t) = \mathsf{GenCipher}(x_{i,t-\mathfrak{T}}, d, t)$ // Send packet //
else
$c_i(t) = \mathsf{GenCipher}(\mathbf{0}, d, t)$ // Send zeros //
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

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

Fig. 4. Protocol run in OrgAn. The texts in // – // blocks denote comments, and this color relates to defenses against disruption that we describe in Section 5. The exact details of those defenses are skipped here.

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

Client 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 for computing $\kappa \cdot x_i^t$, it is treated as integer 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 clients and computes:

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

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

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

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 the appropriate choices of p and q for satisfactory performance in Section 7. 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 4. The setup servers do not take part during the protocol run at all.

4.3 Scaling with Bulk Protocol

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 the 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 a computational/management overhead. So, we present a *Bulk* protocol that minimizes the above two problems. Our Bulk protocol draws its inspiration from the Bulk protocol in Dissent [16]. Each client participates in one round of Base protocol and one round of Bulk protocol to send one IP packet.

Client Permutation Generation with 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, ν_i) to all the users, where ν_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, ν_i) it generates 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 ν_i consecutive fragments in the immediate next Bulk round, starting from the slot $\sum_{j=1}^{\Psi(i)-1} \nu_{\Psi(j)} + 1$. The Bulk round will have a total of $T = \sum_{i=1}^{\mathsf{N}} \nu_i$ slots.

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

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

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

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 $c_1(t), \ldots, c_\mathsf{N}(t)$ from all the users and computes the following:

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

The relay can reveal the corresponding slot value by computing $\lfloor P(t)/\kappa \rceil \mod \mathsf{p}$. The relay retrieves 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. This 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 forward secrecy we recommend running the setup once every few days. The relay can invoke this procedure at regular intervals. 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 Pre-processed PRF Values

Much of the computation overhead on the client during the protocol run can be reduced using some pre-processing. The pre-processing can happen in the following two steps:

Pre-processing Hash Computation. 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 entirely 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 for the range of input values $[1, \mathfrak{T}]$ can be pre-processed and provided to the clients as part of the installation. 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 eliminates the hash computation overhead from the client altogether.

Pre-processing Overall PRF Computation. 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. In Section 7, we explore this pre-processing and real-time split in detail.

5 Handling Disruption

We now describe how OrgAn can detect and defend against a compromised or malfunctioning participants that might disrupt the protocol by sending wrong/malformed ciphertexts. As suggested in Section 2.2, we do not consider active attacks from outside the organizational network.

5.1 Disruption Detection

To detect a disruption we modify our protocol for the relay R to broadcast a response message with the values $x_1, \ldots, x_\mathsf{N}$ 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 for a disrupted round d by broadcasting the tuple $(d, \mathcal{F}(\mathbf{r}_i, d), x_i)$.

Note that the disruption detection technique depends on the relay R delivering the same response message to all the client. In case the relay does not do that, the technique mentioned in Section 5.4 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 modified the messages, (2) some malicious client has sent a bad ciphertext, (3) Some malicious setup server has distributed wrong shares. If the relay did not act maliciously, it is its responsibility to find the culprit.

5.2 Blame Protocol

The Blame protocol for a round is invoked by at least one client, and then it is run by the relay R. For the Blame protocol to work correctly, we need to slightly modify the setup phase as well as the protocol run as described 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 to verify the correctness of the PRF computation. Let the secret shared by G_j to client u_i be $\mathbf{r}_{ij} = \{\alpha_{ij1}, \ldots, \alpha_{ij\mathsf{u}}\}$ such that each $\alpha_{ij1}, \ldots, \alpha_{ij\mathsf{u}} \in \mathbb{Z}_{\mathsf{v}}$ and let all the setup servers and the relay mutually agree on g,h of order $\tau > \mathsf{qv}$. Then each guard server G_j computes $\Gamma_{\mathbf{r}_{ij}} = \left\{\mathsf{g}^{\alpha_{ij1}}\mathsf{h}^{\beta_{ij1}}, \mathsf{g}^{\alpha_{ij2}}\mathsf{h}^{\beta_{ij1}}, \ldots, \mathsf{g}^{\alpha_{ij\mathsf{u}}}\mathsf{h}^{\beta_{ij\mathsf{u}}}\right\}$ with blinding factors $\beta_{(\cdot)} \xleftarrow{\$} \mathbb{Z}_{\mathsf{v}}$. The setup server also computes $\mathfrak{k}_\ell$ such that for each ℓ, $\sum_{i=1}^{\mathsf{N}} \alpha_{ij\ell} = s_\ell + \mathfrak{k}_\ell \cdot \mathsf{v} \bmod \tau$ and a range proof $\pi_{\mathfrak{k},\ell}$ proving $0 < \mathfrak{k}_\ell < \mathsf{N}$. Since we use a non-standard order v, we use commitments on a standard elliptic curve of higher order τ and use suitable verification procedure (c.f. Section 5.3). G_j sends a signature $\sigma_{ij} = \mathcal{S}_{p_{G_j}}(u_i, \Gamma_{\mathbf{r}_{ij}}, \pi_{\mathfrak{k},\ell})$ along with the values $\mathbf{r}_{ij}, \pi_{\mathfrak{k},\ell}$ to the client u_i.

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

- for each $\ell \in \{1, 2, \ldots, \mathsf{u}\}$ if $\Pi_{i=1}^{\mathsf{N}} \mathsf{g}^{\alpha_{ij\ell}}\mathsf{h}^{\beta_{ij\ell}} = \mathsf{g}^{s_\ell}\mathsf{h}^{s'_\ell} \cdot (\mathsf{g}^{-\mathsf{v}\mathfrak{k}_\ell}\mathsf{h}^{-\mathsf{v}\mathfrak{k}'_\ell})$ holds, where $\mathbf{s} = \{s_1, s_2, \ldots, s_{\mathsf{u}}\}$; it also verifies $\pi_{\mathfrak{k},\ell}$. The quantities $\mathbf{s}', \mathfrak{k}'_\ell$ are defined analogously from the blinding factors $\beta_{(\cdot)}$.
- 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 ensure 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.

Our protocol handles the Blame scenario slightly differently when a Base round is disrupted vs. when a Bulk round is disrupted. If a Base round is disrupted since no actual information is transmitted, the clients can reveal the $\mathcal{F}(\mathbf{r}_i, d)$ and x_i values — then they can just rerun the Base round and use the new slot agreement for the next Bulk round. However, if a Bulk round is disrupted, the clients cannot reveal the $\mathcal{F}(\mathbf{r}_i, d)$ and x_i values. In that case, the protocol is designed to leak only 1-bit without deanonymizing the client.

When a Base Round Is Disrupted. A client can invoke the Blame protocol for a Base round d by broadcasting the tuple $(d, \mathcal{F}(\mathbf{r}_i, d), x_i)$ and x_i corresponding to round d.

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 Section 5.3. 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 the Blame has been raised wrongly, the relay can prove that by broadcasting the $\mathcal{F}(\mathbf{r}_i, d)$ and x_i values it received from all the clients.

If at least one slot is disrupted and no client is found guilty (all clients have run the protocol correctly), that means the shares $\mathbf{r}_{ij}$ distributed by the setup servers to the clients were not correct, or the relay is the disruptor. 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.

When a Bulk Round Is Disrupted. The clients cannot reveal the $\mathcal{F}(\mathbf{r}_i, d)$ and x_i values if a Bulk round is disrupted. If Bulk round is disrupted, we modify the protocol to leak only 1-bit without deanonymizing the client.

For the Blame protocol to work for Bulk rounds, we further modify our Bulk 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 part of the content of the message in Bulk protocol, instead of a random number, the client sends the slot index and bit index for the slot that he wants to open. Similar to PriFi, the clients in OrgAn invoke the Blame protocol for a bit position where the original value is 0 (if the original message bit is 1 for a client, that trivially reveals that the slot belongs to the client). This allows the clients to invoke

the Blame protocol without revealing their identities if a Bulk round is disrupted. Additionally, since the Blame protocol is invoked without revealing the client's identity, the client also needs to include a proof to prove that the disrupted slot was actually owned by her — we describe the proof below. It is important to note here, in this version of the Blame protocol the preceding Base round is not opened.

For the client to be able to prove the ownership of a slot without revealing the identity, we make a minor modification to our Base protocol — instead of publishing a random number x during the Base round, the client publishes the hash $H_1(D, x)$. Here D is the response message sent by the relay in the last Bulk round, and $H_1(\cdot)$ is a cryptography secure hash function that maps $\{0,1\}^*$ to $\{0,1\}^{\eta_\mathsf{p}}$. If the Bulk round is disrupted, the client can publish x along with the slot index and bit index to prove that the slot was owned by her without revealing her own identity. The security level of this technique depends on the value of η_p — that determines the probability of the adversary finding a collision. However, the adversary needs to find a collision before the next round to use it against an honest client.

Suppose, the Blame protocol is invoked for bit position ℓ of slot index t. In this version of Blame protocol, a client 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 $(\mathsf{g}^{\psi_1}\mathsf{h}^{\psi'_1}, \mathsf{g}^{\psi_2}\mathsf{h}^{\psi'_2})$ where ψ_1 is the first $(\ell-1)$ bits and ψ_2 is the last $(q-\ell)$ bits of $\mathcal{F}(\mathbf{r}_i, d)_t$. ψ'_1, ψ'_2 are the corresponding parts of the blinding factor. Additionally, the client sends λ_i, a non-interactive zero knowledge range proof for correct PRF evaluation (refer to Section 5.3 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. However, if the Blame has been raised wrongly, the relay can prove its innocence by broadcasting the values it has received from all the clients.

5.3 Verifiability of PRF

Recall that the PRF values were generated by first performing ring product as element-wise multiplication of the NTT transform 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 $\hat{\mathbf{r}}_\mathbf{i} = \{\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{i\mathsf{u}}\}$. 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, $\ell_{ik} = \alpha_{ik} \cdot h_k$ for $k \leq \mathsf{u}$. Let the Inverse NTT of $\mathbf{L}_i$ is denoted by $\mathbf{W}_i = \{w_{i1}, w_{i2}, \cdots, w_{i\mathsf{u}}\}$. Then we have: $w_{ik} = \mathsf{u}^{-1} \sum_{j=1}^{\mathsf{u}} \ell_{ij} \Omega^{-k(j-1)} \bmod \mathsf{v}$, where Ω is the u-th primitive root of unity.

The final PRF output set is obtained as $\lfloor \mathbf{W}_i \rfloor_\mathsf{q} = \{\lfloor w_{i1} \rfloor_\mathsf{q}, \lfloor w_{i2} \rfloor_\mathsf{q}, \cdots, \lfloor w_{i\mathsf{u}} \rfloor_\mathsf{q}\} = \{z_{i1}, z_{i2}, \cdots, z_{i\mathsf{u}}\}$. Therefore, the following holds for each index $k < \mathsf{u}$: $\mathsf{v}z_{ik} + e_{ik} = \mathsf{q}w_{ik}$ for some $0 < e_{ik} < \mathsf{v}$.

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

Additional Steps during Setup Phase. Let all setup servers and the relay mutually agree on generator g, h of a multiplicative group G of order $\tau > \mathsf{qv}$. Each setup server G_j computes $\rho_{\mathbf{r}_{ij}} = \left\{\mathsf{g}^{\alpha_{ij1}}\mathsf{h}^{\beta_{ij1}}, \mathsf{g}^{\alpha_{ij2}}\mathsf{h}^{\beta_{ij2}}, \ldots, \mathsf{g}^{\alpha_{iju}}\mathsf{h}^{\beta_{iju}}\right\}$ for $\beta_{(\cdot)} \xleftarrow{\$} \mathbb{Z}_\mathsf{v}$ 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 e_{ik} and range proofs [11] $\pi_{e,i,k}, \pi_{\mathfrak{k},i,k}$ verifying $0 < e_{ik} < \mathsf{v}$, $0 < \mathfrak{k}_{ik} < min(\tau, \mathsf{uv}^2)$, where $w_{ik} + \mathfrak{k}_{ik}\mathsf{v} = \mathsf{u}^{-1} \sum_{j=1}^{\mathsf{u}} \ell_{ij} \cdot \Omega^{-k(j-1)} \bmod \tau$. Then the client sends as a NIZKP for correct PRF computation, $\lambda = (\mathsf{g}^{e_{ik}}\mathsf{h}^{e'_{ik}}, \pi_{e,i,k}, \pi_{\mathfrak{k},i,k})$ to the relay, without revealing e_{ik}. Here, e'_{ik} is defined and computed analogously to e_{ik} with the corresponding blinding factors.

Relay Side Computation. The relay R can compute the values $\left\{\mathsf{g}^{\alpha_{i1}\mathsf{h}^{\beta_{i1}}, \alpha_{i2}}\mathsf{h}^{\beta_{i2}}, \ldots, \mathsf{g}^{\alpha_{iu}}\mathsf{h}^{\beta_{iu}}\right\}$ using the values it received during the setup phase. Using this, the relay can compute:

$$\mathsf{g}^{w_{ik}}\mathsf{h}^{w'_{ik}} = \mathsf{g}^{-\mathsf{v}\mathfrak{k}_{ik}}\mathsf{h}^{-\mathsf{v}\mathfrak{k}'_{ik}} \prod_{j=1}^{\mathsf{u}} (\mathsf{g}^{\alpha_{ij}}\mathsf{h}^{\beta_{ij}})^{h_j \cdot \Omega^{-k(j-1)} \cdot \mathsf{u}^{-1}}$$

for $k \leq \mathsf{u}$. Note here, v, Ω, h_j are public information, and $\mathfrak{k}'_{ik}$ is the blinding factor for $\mathfrak{k}_{ik}$.

To verify that a PRF value z_{ik} is correct, the relay needs to verify $\mathsf{g}^{\mathsf{v}z_{ik}}\mathsf{h}^{\mathsf{v}z'_{ik}} \cdot \mathsf{g}^{e_{ik}}\mathsf{h}^{e'_{ik}} = \mathsf{g}^{\mathsf{q}w_{ik}}\mathsf{h}^{\mathsf{q}w'_{ik}}$. Here, $z'_{ik}, w'_{ik}, e'_{ik}$ are defined analogously to z_{ik}, w_{ik}, e_{ik} with the corresponding blinding factors.

Note that the relay can compute $\mathsf{g}^{\mathsf{q}z_{ik}}\mathsf{h}^{\mathsf{q}z'_{ik}} = \left(\mathsf{g}^{\psi_1}\mathsf{h}^{\psi'_1}\right)^{2^\ell} \cdot \mathsf{g}^{\psi_2}\mathsf{h}^{\psi'_2}$, corresponding to the Blame for a Bulk round. Additionally, the relay needs to verify the range proofs $\pi_{e,i.k}$ and $\pi_{\mathfrak{k},i,k}$.

5.4 Equivocation Protection

With our key-homomorphic PRF based construction, we 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 $\{0,1\}^\eta$ for the security parameter η. Note that the multiplication between $h(d-1)$ and $\mathbf{r}_i$ is a scalar multiplication since $h(d-1)$ is a scalar value.

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 argue the anonymity properties of the protocol in the passive adversary setting, as well as against relevant active attacks. Here we present the security theorems and their implications, and postpone the proofs to Appendix A. The security definition of PRFs is also presented in Appendix A.

6.1 Anonymity Definition

We focus on sender anonymity for our protocol. We formally define anonymity based on AnoA [36] 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 the two adversarially-chosen senders has sent a message to a specific recipient (sender anonymity). More formally, the adversary can send polynomial number of input messages of the form $(\texttt{Input}, u, R, m)$, then one challenge message $(\texttt{Chall}, u_0, u_1, R_0, R_1, m_0, m_1)$, and tries to guess a challenge bit b of the challenger in the game.

Definition 2 ((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* $\mathcal{A}$*,*
$\Pr[0 = \langle \mathcal{A} | \texttt{Ch}(\Pi, \alpha, 0)\rangle] \leq \Pr[0 = \langle \mathcal{A} | \texttt{Ch}(\Pi, \alpha, 1)\rangle] + \delta(\eta)$.

Definition 3 (Sender anonymity)**.** *A protocol* Π *provides* δ-sender anonymity *if it is* (α_{SA}, δ)-IND−ANO *for* α_{SA} *as defined in Figure 5.*

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

RunProtocol(u, R, m)

Upon message $(\texttt{Chall}, u_0, u_1, R_0, R_1, m_0, m_1)$**:**

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.

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

Fig. 5. Adaptive AnoA Challenger $\texttt{Ch}(\Pi, \alpha, b)$ [36]

6.2 Anonymity Analysis

Theorem 1 (Sender Anonymity of Base Protocol)**.** *Assuming* $\mathcal{F}()$ *is a computationally secure pseudorandom function, the Base protocol of OrgAn provides sender anonymity as defined in Definition 3 with negligible* δ *against any global passive adversary* $\mathcal{A}$*, as long as at least two clients and one setup server are honest.*

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, OrgAn provides sender anonymity (when Bulk protocol is employed) as defined in Definition 3 with negligible* δ *against any global passive adversary* $\mathcal{A}$*, as long as at least two clients and one setup server are honest.*

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

6.3 Security against Active Attacks

When a malicious client tries to disrupt the protocol by sending a 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., all honest clients receive the same response message after a 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.*

A direct corollary of the above lemma is that the honest clients can 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.*

If 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 proves correct computation of $\mathcal{F}(\mathbf{r}_i, t)$ for a client u_i with 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 party is identified, and an honest party is not (mis-)identified as a disruptor.*

Unlike the Base protocol, the x_i and $\mathcal{F}(\mathbf{r}_i, t)$ values are not opened if a Bulk round is disrupted. Therefore, we additionally want that for a disrupted slot ℓ in a Bulk round only the client owning that slot can launch the Blame protocol.

Lemma 2. *Assuming H_1 is a collision resistant hash function, and the computation power of the adversary is limited by T hash computations between two consecutive Base rounds where T is polynomial in η_p, the* Blame *protocol in disrupted Bulk round d can be invoked by a malicious client u^* for a slot ℓ which is not owned by u^* only with a probability negligible in η_p.*

The above lemma is important to ensure that a malicious client cannot launch the Blame protocol for an arbitrary slot just to break anonymity of a Bulk round. Recall that, when a Base round is disrupted, the x_i and PRF values are anyway opened during the Blame phase. And therefore, anonymity is trivially broken for a disrupted Base round when the x_i and PRF values are opened. To solve that, OrgAn reruns the Base round and uses the new slot agreement for the next Bulk round. Hence, we do not want a disrupted Base round to influence anonymity for 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 for a Base round d does not break anonymity for any other round $d' \neq d$.*

Recall that this leakage is not there where a Bulk round is disrupted. Because when client chooses to invoke the Blame protocol, only the bit position is opened where the original bit value was 0. Therefore, anonymity of the client is protected, and Blame is invoked with probability almost $\frac{1}{2}$ in case of a disruption. According to Theorem 3, the disruptor is identified with overwhelming probability when Blame is invoked.

7 Implementation

We have developed a proof-of-concept implementation[2] of OrgAn in Rust. 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 Flint2 [1] library for solving powersum equations. In the following, we discuss the considerations that we make for our implementation.

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.

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 every day 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 the history and thus cannot be preprocessed.

Fragmentation and Defragmentation of Packets. The clients transmit η_p bits of data per slot in a Bulk round; 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. After computing all the messages, the relay node identifies the fragments of each client, forms the full IP packets by combining the fragments, and forwards it to the outside network.

2 https://github.com/zhtluo/organ

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 the 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 PRFs 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 (p, q, v, u) are chosen such that the PRF construction offers at-least 128 bits of security estimated using the lwe-estimator [2, 4]. (p, q, v) are of bit-lengths $(\eta_{\mathsf{p}}, \eta_{\mathsf{q}}, \eta_{\mathsf{v}})$ respectively. For the base round, each client chooses a 64 bit random value for slot selection (64 bit hash output of random corresponding value from previous round); hence we use $\eta_{\mathsf{p}} = 64$. In the bulk round, the clients forward 226 bits of data in each slot, with $\eta_{\mathsf{p}} = 226$. The corresponding parameters for base and bulk rounds are:

- Base: $(\eta_{\mathsf{p}}, \eta_{\mathsf{q}}, \eta_{\mathsf{v}}, \mathsf{u}) = (64, 84, 112, 2048)$
- 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 $\mathsf{v} \bmod 2\mathsf{u} = 1$. For error elimination in the PRF computation, we use $\kappa = 1000$ in our implementation.

Note that, even though our choice of $\eta_{\mathsf{p}} = 64$ for Base rounds restricts the security of the hash function H_1 to maximum 64 bits, for all other cryptographic constructions we choose at least 128-bit security. we find this choice of η_{p} to be satisfactory as the adversary has to find a pre-image before the next Base round to use that against an honest client.

8 Performance Evaluation

8.1 Microbenchmarks

Here we evaluate the overheads of different steps for the clients and the relay individually using our prototype implementation. All the measurements are average of 10 runs of the same experiment, unless otherwise specified.

8.1.1 Overhead for the Clients

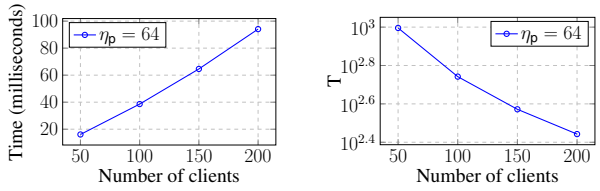
Preprocessing Overhead for Clients. We evaluate the preprocessing overhead for clients on an 16-virtual core AWS EC2 c4.4xlarge instance. Each PRF (pre-)computation corresponding to a base round takes 0.5 milliseconds; and for a bulk round, it takes 5 milliseconds — the difference arising from the orders of the ring elements 2048 and 8192. A total of 5.5 milliseconds of preprocessing for one set of base and bulk rounds of protocol allowing to forward $\frac{8192 \times 226}{100}$ bits for each client in a 100 client system — that would amount to ~ 39.6 minutes of preprocessing for 1 GB of anonymous communication for each client. For 12 hours of preprocessing per day, our current implementation can support 18.1GB of anonymous communication data per client, in a 100 client system. We want to note that the preprocessing overhead can be further reduced with an optimal implementation with improved parallelism.

Real-time Overhead for Clients. The real-time overhead for clients only involves additions and multiplications and does not involve any costly computations. The total computation for a client in a base round, as well as bulk round, takes less than half a millisecond (for a 100 client system). The base round involves a few hundred additions and multiplications for each client. For a bulk round, the client only needs one scalar multiplication per slot, one addition (of the message), in addition to the history computation. The history is computed only once for each round.

8.1.2 Overhead for Relay to Solve Equations System

We use an Amazon AWS EC2 c5.24xlarge instance, with 96 virtual cores 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 $\eta_{\mathsf{p}} = 64$ bits as the message size for each message in Base rounds.

Fig. 6a 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 ~ 550 equation systems per second (using multi-threading) and hence can support ~ 550 packets per client per second. Fig. 6b 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 550 KBps and 225 KBps if the average packet size is 512 bytes. With increasing clients in the system, the time taken to solve the equations system 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 the 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).

Fig. 6. 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 $\eta_p = 64$. The values show mean of 100 runs of the protocol.

Table 2. Client-relay-client round trip time (RTT) for the slot selection using the base protocol phase in OrgAn.

Nodes	50	100	150	200
RTT (msec)	37	56	84	116

8.2 End-to-end Latency Evaluation

Experimental Setup. We use an AWS EC2 c5.24xlarge instance acting as the relay and ten EC2 c4.4xlarge 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 the same level of anonymity as OrgAn. In our experiments, we ignore the overhead of such setup runs[3] as well as the costs of equivocation protection 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 Table 2 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 60 milliseconds.

End-to-end Latency. We consider that the Base and Bulk rounds of OrgAn are run in parallel threads as mentioned in Section 7. Fig. 7 shows the round trip time (RTT) of Bulk rounds of OrgAn while pinging an external server (the IP address google.com); we compare the RTT of Bulk rounds in

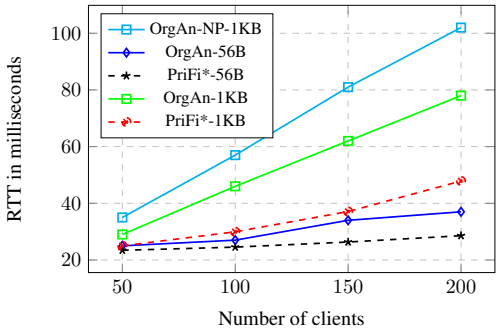


Fig. 7. Round trip time (RTT) to ping an external server google.com by forwarding 56 byte and 1KB message using the bulk protocol of OrgAn and PriFi* (that excludes the cost of setup and equivocation protection from PriFi). The suffix in the legend indicates the packet size used. OrgAn-NP is OrgAn with no preprocessing.

OrgAn with PriFi. Barman et al. [8] already show that PriFi outperforms other existing protocols with significant margins, and therefore, it is sufficient to compare our performance with only PriFi.

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 for each slot are received by the relay before the round starts. In a 100 client system, PriFi has as RTT of 24 milliseconds and OrgAn has 27 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 29 and 46 milliseconds respectively. While OrgAn introduces a (slightly) higher end-to-end 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 computations including group operations.

Throughput Comparison. We compare the throughput of OrgAn with PriFi protocol in Fig. 8 using a Rust implementation of the protocols . We use an AWS EC2 c5.24xlarge instance acting as the relay and ten EC2 c4.4xlarge instances simulating all the clients, each instance launching multiple clients. To compute the throughput for OrgAn, we compute the time taken by the client to run 100 base and bulk rounds sequentially. For each set of base and bulk rounds, for a 100 client system, each client forwards 1KB of message data. Since each slot supports a client data of 226 bits, to forward 1KB, each client uses 37 slots, thus totaling 3700 slots (of the bulk round) for a 100 client system. For 100 rounds, each forwards 100KB; dividing this value by the time taken to complete 100 base and bulk rounds provides the throughput achieved. While we show throughput using sequential rounds, in a deployment, multiple rounds can be processed in parallel for higher throughput.

3 In order to allow 1 GB of data for a 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 to achieve unlinkability for each IP packet.

For PriFi, similar to OrgAn we consider 1KB messages. In a round, each client uses one slot for its messages, thus the total number of slots in a round is the number of clients. To compute the throughput, we compute the total time taken by the clients to forward 100 such messages (rounds) to the relay. After collecting each set of messages from the clients, the relay forwards a packet to indicate the end of the round. To achieve the packet level of anonymity as OrgAn, PriFi needs to run a setup phase for each round of communication by the clients. Fig. 8 shows the throughput achieved by our implementation of PriFi without setup. To estimate the time taken to run the setup phase of PriFi, we use a publicly available implementation [21] of Neff's verifiable shuffle protocol [37]. Each such verifiable shuffle, including shuffle and its verification take ~ 0.432 seconds for 100 values on an AWS EC2 c4.4xlarge server machine. Considering 10 servers for each verifiable shuffle protocol would lead to 4.3 seconds of setup time per round for the PriFi protocol. Since the time taken for setup is much higher compared to the client message generation, the latency is dominated by the setup time reducing the throughput. In Fig. 8, PriFi* is the throughput of the system without the setup phase. PriFi** is the estimated throughput of PriFi with setup phase for each packet (1KB) using a verifiable shuffle with 10 guard servers. Fig. 8 also shows the throughput of OrgAn with and without pre-processing (shown as OrgAn-NP). Since the processing and network delays dominated when running sequentially on a single thread, preprocessing the PRF values offered only minor improvement in the throughput. PriFi** shows that for the same (packet) level of anonymity, OrgAn outperforms PriFi.

Communication and Computation Costs. In both OrgAn and PriFi all the clients forward values for each slot in the bulk rounds. Considering a data size of 1KB, each client forwards 1KB of message data in PriFi, wherein OrgAn, each client forwards a total of 1.13KB of data. This is because, to forward a data size of 226 bits in each slot, the client masks with a pseudo-random value of 256 bits in OrgAn whereas, in PriFi, it is just XORed with a pseudo-random value retaining the bit-length. In OrgAn, the relay collects all the slot messages from the clients and performs group addition and scaling; in PriFi all the slot messages are XORed.

8.3 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 the 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 $\frac{10^{10} \times 8}{(112 \times 2048) + (293 \times 8192)} \approxeq 30422$ IP packets anonymously. Assuming an average size of 1 KB per IP packet, this would correspond to ~ 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. 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.

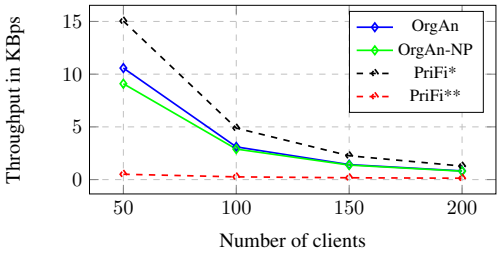


Fig. 8. Throughput comparison between OrgAn and PriFi systems for sequentially running 100 1KB message rounds. PriFi* is PriFi protocol without the setup phase. PriFi** is PriFi assuming 10 servers are running the shuffle protocol (setup) for each round. OrgAn-NP is OrgAn with no pre-processing.

9 Application Considerations

9.1 Problem of Enumerating All Clients

The protocol described in Section 4 requires the setup servers to know all the clients in the system to be able to share the $\mathbf{r}_{ij}$ values. However, even if each setup server does not know all the clients, the system will still work. A setup server can split $\mathbf{s}$ only among the clients it knows. As long as the relay knows the total number K of setup servers, the relay can retrieve the messages correctly because $\sum_{i,j} \mathbf{r}_{ij} = \mathsf{K} \cdot \mathbf{s}$ still holds. As long as each client receives a secret share from at least one server it trusts, the security guarantees also remain the same.

9.2 Client Churn

If a client is unavailable during a round, the shares from that client will not be available and the relay will be unable to retrieve the message. This client churn issue is a challenge for most DC-net inspired protocols, especially in types (a) and (c) in Fig. 2. For OrgAn, if such a situation occurs the setup can

be run again with the new set of clients; however, all the advantages from the precomputations will be lost in that case. To avoid such problems, we assume that the clients are run on office desktops (considering an office setting) that are less likely to go offline. We welcome the community for a robust solution for client churn in DC-net inspired systems.

Additionally, if we assume that the setup servers can be contacted (even though they are not involved in the usual protocol runs), they can collectively provide the $\mathcal{F}(\mathbf{r}_i, d)_t$ values for an unavailable client i for the upcoming rounds. That provides an interesting possibility to mitigate the client churn problem for OrgAn as well as PriFi. However, the system needs to agree that a client is indeed unavailable as otherwise a malicious relay may deanonymize the client by claiming its unavailability to the setup servers — we leave the exact solution to that problem for future work.

9.3 Application Scenarios

ICRC Scenario. In Section 1, we discuss a strong need for traffic-analysis-resistant anonymous communication for delegates at multinational organizations such as ICRC. 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 the case of PriFi as the relay needs to receive ciphertexts from each of the servers for every round. Such a bottleneck is not present in OrgAn since the servers and the relay do not communicate with each other. Moreover, the servers in OrgAn are logical entities and can be run by the clients, removing the requirement of having external servers completely.

In the above scenario, the clients can use our protocol to anonymously access different kinds of applications like browsing, DNS queries, calls etc.

VPN. Two separate organizational networks can set up 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. Similarly, a remote access VPN can also be implemented using OrgAn.

Mixing Bitcoin Transactions. The Base protocol of OrgAn can be used for mixing Bitcoin transactions exactly in the same way as Dicemix [41]. Since OrgAn eliminates the necessity to run a key agreement for every round, OrgAn will yield a much faster transaction mixing protocol. As future work, we plan to integrate OrgAn into CoinShuffle++ protocol [41].

9.4 Supporting Lightweight Clients

If client churn is not a concern (e.g., a meeting scenario where all the clients remain online throughout the whole meeting or mixing Bitcoin scenario), we can run the protocol on mobile and other lightweight devices. We can do that by offloading the preprocessing onto a desktop machine — during non-working hours that machine can preprocess and store the data in a removable storage (e.g., micro-SD cards can store up to 128 GB of data). At the beginning of the meeting, the removable storage can be put into the mobile device and the mobile device can now use the preprocessed data to run the protocol.

10 Conclusion

In this paper, we have presented a new AC protocol OrgAn to provide strong anonymity guarantees in an organizational network. Our protocol solved a crucial bottleneck in the state-of-art PriFi – PriFi had to choose between regularly running the setup phase involving expensive verifiable shuffle among the setup servers and linkability among IP packets between two setup runs. Although we have used public-key cryptography in our design, we demonstrated that OrgAn is still very practical with the help of some preprocessing and storage requirements. Moreover, OrgAn has removed the dependency on setup servers during the real-time phase completely. Further, if the application scenario demands, the clients can take up the roles of the setup servers during the setup phase, thus, completely eliminating the need for any external servers. Conceptually, we could have used our Base protocol to just solve the slot selection in PriFi, instead of using our Bulk protocol. However, such a scheme will still be dependent on the setup servers in the real-time phase.

We do not solve the problem of client churn where some clients in the system are suddenly absent/unavailable; it remains an open problem for all DC-net inspired protocols.

Acknowledgement. We thank Zhongtang Luo for his help with running the experiments. We thank Bryan Ford and Ludovic Barman for discussions during the early phase of the project, and the anonymous reviewers and Tim Ruffling for the insightful feedback. The work has been partially supported by the National Science Foundation (NSF) under grant CNS-1846316. Most of the work was completed when Debajyoti was a student at Purdue University. Both the corresponding authors (Debajyoti and Easwar) have contributed equally to the paper and have been ordered alphabetically.

References

[1] *FLINT: Fast Library for Number Theory*, . https://www.flintlib.org/.

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

[3] I. Abraham, B. Pinkas, and A. Yanai, *Blinder – scalable, robust anonymous committed broadcast*, in Proceedings of the 2020 ACM SIGSAC CCS, 2020, p. 1233–1252.

[4] M. R. Albrecht, R. Player, and S. Scott, *On the concrete hardness of learning with errors*, Journal of Mathematical Cryptology, 9 (2015), pp. 169–203.

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

[6] A. Banerjee and C. Peikert, *New and improved key-homomorphic pseudorandom functions*, in Advances in Cryptology – CRYPTO 2014, 2014.

[7] A. Banerjee, C. Peikert, and A. Rosen, *Pseudorandom functions and lattices*, in Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques, EUROCRYPT'12, 2012, pp. 719–737.

[8] L. Barman, I. Dacosta, M. Zamani, E. Zhai, A. Pyrgelis, B. Ford, J. Feigenbaum, and J. Hubaux, *Prifi: Low-latency anonymity for organizational networks*, Proc. Priv. Enhancing Technol., 2020 (2020), pp. 24–47.

[9] D. Boneh, K. Lewi, H. Montgomery, and A. Raghunathan, *Key homomorphic prfs and their applications*, in Advances in Cryptology – CRYPTO 2013, 2013.

[10] J. Bos and B. den Boer, *Detection of disrupters in the dc protocol*, in Advances in Cryptology — EUROCRYPT '89, J.-J. Quisquater and J. Vandewalle, eds., 1990, pp. 320–327.

[11] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, *Bulletproofs: Short proofs for confidential transactions and more*, in 2018 IEEE Symposium on Security and Privacy (SP), IEEE, 2018, pp. 315–334.

[12] D. Chaum, *Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms*, Communications of the ACM, 4 (1981), pp. 84–88.

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

[14] D. Chaum, D. Das, F. Javani, A. Kate, A. Krasnova, J. de Ruiter, and A. T. Sherman, *cmix: Mixing with minimal real-time asymmetric cryptographic operations*, in ACNS, 2017.

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

[16] H. Corrigan-Gibbs and B. Ford, *Dissent: Accountable Anonymous Group Messaging*, in Proc. ACM SIGSAC CCS, 2010, pp. 340–350.

[17] H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford, *Proactively Accountable Anonymous Messaging in Verdict*, in Proc. 22nd USENIX Security Symposium, 2013, pp. 147–162.

[18] G. Danezis, R. Dingledine, and N. Mathewson, *Mixminion: design of a type iii anonymous remailer protocol*, in 2003 Symposium on Security and Privacy, 2003., 2003, pp. 2–15.

[19] D. Das, S. Meiser, E. Mohammadi, and A. Kate, *Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency - choose two*, in 2018 IEEE Symposium on Security and Privacy (SP), 2018, pp. 108–126.

[20] D. Das, S. Meiser, E. Mohammadi, and A. Kate, *Comprehensive anonymity trilemma: User coordination is not enough*, Proceedings on Privacy Enhancing Technologies, 2020 (2020), pp. 356–383.

[21] DEDIS, *DEDIS Advanced Crypto Library for Go*. https://github.com/dedis/kyber/tree/master/shuffle. 2019.

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

[23] R. Dingledine, N. Mathewson, and P. Syverson, *Tor: The second-generation onion router*, in Proceedings of the 13th USENIX Security Symposium, 2004, p. 21.

[24] S. Eskandarian, H. Corrigan-Gibbs, M. Zaharia, and D. Boneh, *Express: Lowering the cost of metadata-hiding communication with cryptographic privacy*, in 30th USENIX Security Symposium, M. Bailey and R. Greenstadt, eds., 2021, pp. 1775–1792.

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

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

[27] H. W. Gould, *The girard-waring power sum formulas for symmetric functions, and fibonacci sequences*, Fibonacci Quarterly, 37 (1999), pp. 135–140. https://www.fq.math.ca/Issues/37-2.pdf.

[28] J. Katz and Y. Lindell, *Introduction to modern cryptography*, CRC press, 2020.

[29] A. Krasnova, M. Neikes, and P. Schwabe, *Footprint scheduling for dining-cryptographer networks*, in FC, J. Grossklags and B. Preneel, eds., 2016, pp. 385–402.

[30] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford, *Atom: Horizontally scaling strong anonymity*, in Proceedings of the 26th SOSP, 2017, p. 406–422.

[31] D. Lazar, Y. Gilad, and N. Zeldovich, *Karaoke: Distributed private messaging immune to passive traffic analysis*, in 13th USENIX OSDI), 2018, pp. 711–725.

[32] S. Le Blond, D. Choffnes, W. Caldwell, P. Druschel, and N. Merritt, *Herd: A Scalable, Traffic Analysis Resistant Anonymity Network for VoIP Systems*, in Proc. ACM SIGCOMM 2015, 2015, pp. 639–652.

[33] S. Le Blond, D. Choffnes, W. Zhou, P. Druschel, H. Ballani, and P. Francis, *Towards Efficient Traffic-analysis Resistant Anonymity Networks*, in Proc. ACM SIGCOMM 2013, 2013, pp. 303–314.

[34] S. Le Blond, A. Cuevas, J. R. Troncoso-Pastoriza, P. Jovanovic, B. Ford, and J.-P. Hubaux, *On enforcing the digital immunity of a large humanitarian organization*, in 2018 IEEE Symposium on Security and Privacy (SP), 2018, pp. 424–440.

[35] D. Lu, T. Yurek, S. Kulshreshtha, R. Govind, A. Kate, and A. Miller, *Honeybadgermpc and Asynchromix: Practical asynchronous mpc and its application to anonymous communication*, in Proceedings of the 2019 ACM SIGSAC

CCS, 2019, pp. 887–903.
[36] M. Backes, A. Kate, P. Manoharan, S. Meiser, and E. Mohammadi, *AnoA: A Framework For Analyzing Anonymous Communication Protocols*, Journal of Privacy and Confidentiality (JPC), 7(2) (2016).
[37] C. A. Neff, *A verifiable secret shuffle and its application to e-voting*, ACM CCS, 2001, p. 116–125.
[38] U. Office of the Director of National Intelligence (ODNI), *Global trends 2040*, 2021.
[39] D. Pointcheval and J. Stern, *Security arguments for digital signatures and blind signatures*, Journal of Cryptology, 13 (2001).
[40] RFC 8446, *The Transport Layer Security (TLS) Protocol Version 1.3*. https://tools.ietf.org/html/rfc8446. Accessed April 2021.
[41] T. Ruffing, P. Moreno-Sanchez, and A. Kate, *P2P Mixing and Unlinkable Bitcoin Transactions*, in Proc. 25th NDSS, 2017.
[42] C. P. Schnorr, *Efficient identification and signatures for smart cards*, in Advances in Cryptology — CRYPTO' 89 Proceedings, G. Brassard, ed., 1990, pp. 239–252.
[43] C. R. Scott and S. A. Rains, *Anonymous communication in organizations: Assessing use and appropriateness*, Management Communication Quarterly, 19 (2005), pp. 157–197.
[44] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich, *Stadium: A distributed metadata-private messaging system*, in Proceedings of ACM SOSP, 10 2017, pp. 423–440.
[45] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich, *Vuvuzela: Scalable private messaging resistant to traffic analysis*, in Proc. 25th ACM SOSP, 2015.
[46] L. von Ahn, A. Bortz, and N. J. Hopper, *K-anonymous message transmission*, in Proceedings of the 10th ACM SIGSAC CCS, 2003, p. 122–130.
[47] M. Waidner, *Unconditional sender and recipient untraceability in spite of active attacks*, in Advances in Cryptology — EUROCRYPT '89, 1990, pp. 302–319.
[48] M. Waidner and B. Pfitzmann, *The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability*, in Advances in Cryptology — EUROCRYPT '89, 1990, pp. 690–690.
[49] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson, *Dissent in Numbers: Making Strong Anonymity Scale*, in 10th USENIX OSDI'12, 2012, pp. 179–182.

A Postponed Proofs

A.1 Security Definition for PRFs

We borrow the security definition for PRFs from existing literature [28] and use it in our security arguments.

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

$\mathbf{Expt}_{PRF}(\mathcal{A})$	$\mathbf{Expt}_{g}(\mathcal{A})$
$\mathbf{K} \leftarrow \mathcal{K}$	$g \leftarrow \mathsf{Rand}(\mathcal{D}, \mathcal{O})$
$b = \mathcal{A}(\mathcal{F}_{\mathbf{K}})$	$b = \mathcal{A}(g)$

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

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 5. *$\mathcal{F}$ 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 4 is bounded by a negligible quantity in the security parameter η.*

A.2 Anonymity Proofs

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 3 with negligible δ against any global passive adversary $\mathcal{A}$, as long as at least two users and one setup server are honest.*

Proof of Theorem 1. Without loss of generality, let us assume that users u_1 and u_2 are honest and their message in a given round is 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 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 $c_2(t) = \kappa x_2^t + \mathcal{F}(\mathbf{r}_1 + \mathbf{r}_2, d)_t - \mathcal{F}_{rand}(t)$. 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 $f_2 = \mathcal{F}(\mathbf{r}_1 + \mathbf{r}_2, d)_t - \mathcal{F}_{rand}(1)$. 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 to prove the anonymity of OrgAn. We show that if there exists an adversary $\mathcal{A}_{anon}$ that breaks sender anonymity for protocol $OrgAn$, we can construct an adversary $\mathcal{A}_{PRF}$ that breaks the security assumption on pseudorandom function $\mathcal{F}$.

Claim 2. *If there exists a PPT adversary $\mathcal{A}_{anon}$ with an adversarial advantage δ against the protocol $OrgAn$ in the sender anonymity game defined in Definition 3, 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 4.*

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 $\mathcal{A}_{anon}$[4], $\mathcal{A}_{PRF}$ 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 $c_2(t) = \kappa x_i^t + \mathcal{F}(\mathbf{r}_1 + \mathbf{r}_2, d)_t - f_t$. $\mathcal{A}_{PRF}$ 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 $f_t = \mathcal{F}(\mathbf{K}, d)_t$ 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. □

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 3 with negligible δ against any global passive adversary $\mathcal{A}$, as long as at least two users and one setup server are honest.*

Proof Sketch for Theorem 2. We use a similar technique as the proof of Theorem 1 to prove this theorem: First, we use a modified version $OrgAn^*$ of the protocol OrgAn that employs

4 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.

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 $\mathcal{F}_{rand}(t)$ instead of $\mathcal{F}(\mathbf{r}_1, d)_t$ as the masks to compute the ciphertexts, and the user u_2 uses $\mathcal{F}(\mathbf{r}_1 + \mathbf{r}_2, d)_t - \mathcal{F}_{rand}(t)$. 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.

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 exists 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 the PRF $\mathcal{F}$.

Claim 4. *If there exists a PPT adversary $\mathcal{A}_{anon}$ with an adversarial advantage δ against the protocol $OrgAn$ in the sender anonymity game defined in Definition 3, 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 4.*

We construct $\mathcal{A}_{PRF}$ and set up 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 PRF. □

A.3 Integrity Proofs

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 for Lemma 1. 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 $\mathcal{F}(\mathsf{K} \cdot \mathbf{s} + (h_i - 1) \cdot \mathbf{r}_i + (h_j - 1) \cdot \mathbf{r}_j, d)$ within the 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. □

Theorem 3. *Assuming λ_i proves correct computation of $\mathcal{F}(\mathbf{r}_i, t)$ for a client u_i with 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 for Theorem 3. A round is disrupted in the following three possible scenarios 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 corresponding to a Base round, each client sends their x_i and $\mathcal{F}(\mathbf{r}_i, d)_t$ 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 $\mathcal{F}(\mathbf{r}_i, d)_t$ during value opening. The client cannot send wrong x_i value because that will yield overall wrong c_i computation.

In case a Bulk round is disrupted, if the ℓ-th bit of a slot is disrupted, the client sends the ℓ-th bit of $\mathcal{F}(\mathbf{r}_i, d)_t$ and a tuple $(\mathsf{g}^{\psi_1}\mathsf{h}^{\psi_1'}, \mathsf{g}^{\psi_2}\mathsf{h}^{\psi_2'})$ where ψ_1 is the first $(\ell - 1)$ bits and ψ_2 is the last $(q - \ell)$ bits of $\mathcal{F}(\mathbf{r}_i, d)_t$. Using those values and λ_i, the relay verifies the correct computation of PRF. Similar to the previous case, unless λ_i is broken, the client cannot send wrong values.

Given the security of signing $\mathcal{S}()$, 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 $\ell \in \{1, 2, \ldots, \mathsf{u}\}$ if $\Pi_{i=1}^{\mathsf{N}} \mathsf{g}^{\alpha_{ij\ell}} \mathsf{h}^{\beta_{ij\ell}} = \mathsf{g}^{s_\ell} \mathsf{h}^{s'_\ell} \cdot (\mathsf{g}^{-\mathsf{v}\mathfrak{k}_\ell} \mathsf{h}^{-\mathsf{v}\mathfrak{k}'_\ell})$ holds, where $\mathbf{s} = \{s_1, s_2, \ldots, s_\mathsf{u}\}$; it also verifies $\pi_{\mathfrak{k},\ell}$. The quantities $\mathbf{s}', \mathfrak{k}'_\ell$ are defined analogously from the blinding factors $\beta_{(\cdot)}$.
– if each σ_{ij} is a valid signature of $(u_i, \Gamma_{\mathbf{r}_{ij}})$ generated by the setup server 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, or the relay is blamed.

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. *Assuming H_1 is a collision resistant hash function, and the computation power of the adversary is limited by T hash computations between two consecutive Base rounds where T is polynomial in η_p, the* Blame *protocol in disrupted Bulk round d can be invoked by a malicious client u^* for a slot ℓ which is not owned by u^* only with a probability negligible in η_p.*

Proof Sketch for Lemma 2. The proof directly translates from the security of the hash function H_1. Each client client$_i$ publishes $H_1(D, x)$ in the base round where x is randomly picked, and D is the response message of the last Bulk round.

To launch the Blame protocol for a Bulk round for a slot ℓ that does not belong to the client client*, the malicious client has to find an x^* such that $H_1(D, x) = H_1(D, x^*)$ with the computation limit T — which breaks the collision resistant property of H_1. Since the range of H_1 is only η_p, the security of H_1 is limited by η_p. □

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 of Theorem 4. We want to prove the above theorem 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.[5]

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 $c_2(t) = \kappa x_i^t + \mathcal{F}(\mathbf{r}_1 + \mathbf{r}_2, d)_t - f_t$.

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 in round d'. 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. □

B Short Description of PriFi

PriFi utilizes the client/relay/server model to solve the bottleneck of running key-agreement protocol before every round of DC-net-based protocols. In this client/relay/server model, PriFi protocol has three sets of entities: 1. clients $u_1, \ldots, u_\mathsf{N}$ that want to communicate anonymously to outside services, 2. some servers $G_1, \ldots, G_\mathsf{K}$ (or 'guardnodes' as they call them) that help in the anonymization process, 3. and a *relay* or gateway server R. PriFi has a *Setup* phase when the clients and the guardnodes establish some shared secrets among themselves; and agree on a permutation of slots where each client knows only its own slot. Using the shared secrets, the clients (and servers) can generate keys for several *Anonymize* rounds. The clients can transmit actual data in those rounds. However, if the clients want to agree on a new permutation of slots, they need to re-run the setup phase.

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

Setup Phase. Each client u_i generates a pair of ephemeral private-public keypair p_i, P_i. Then each client u_i runs an authenticated Diffie-Hellman key exchange using the ephemeral keypair with each server G_j to agree on a shared secret r_{ij}. Additionally, the servers run a verifiable shuffle algorithm to generate a permuted output of the public keys $\pi = \{\tilde{P}_{\alpha_1}, \tilde{P}_{\alpha_2}, \ldots, \tilde{P}_{\alpha_\mathsf{N}}\}$ — where $\tilde{P}_{\alpha_i} = c \cdot P_i$ for a permutation α and some constant c; and therefore, only a client with private key p_i can recognize the pseudonym key in π that corresponds to P_i. The shuffled output π is made public by the servers.

Anonymize Phase. For round $k \in \{1, \ldots, \mathsf{N}\}$, each client u_i generates the DC-net mask as $X_i = \oplus_{j=0}^{\mathsf{K}} PRG(r_{ij})$. If client u_i is supposed to send their message in round k (based on the permutation π), u_i sends the ciphertext $c_i = m_i \oplus X_i$ to the relay R, otherwise u_i sends $c_i = X_i$. Each server G_j generates their ciphertext as $Y_j = \oplus_{i=0}^{\mathsf{N}} PRG(r_{ij})$ and sends it to the relay.

After receiving all the $\mathsf{N}+\mathsf{K}$ ciphertexts, the relays XORs them together to retrieve m_k corresponding to the round k. The servers can send their ciphertexts to the relay ahead of time to avoid any delay because of the servers. It is assumed that m_k is a full IP packet or part of an IP packet (albeit null sourced) so that the relay can buffer it and send it to the appropriate destination. When the relay receives a response d_k corresponding to a message m_k, the relay encrypts d_k with the public key $\tilde{P}_k$ and broadcasts that to all the clients.

Handling Disruption

PriFi uses a hash-based disruption detection mechanism. The relay sends the hash of an upstream message with the downstream traffic. The sender can detect an incorrect hash and invoke the Blame protocol as described below.

Blame Protocol. The basic PriFi protocol is modified to include an additional bit in the upstream messages; by setting the bit to 1, a client can invoke the Blame protocol. Additionally, the client includes the disrupted bit position ℓ and a non-interactive zero-knowledge proof (NIZKP) of knowledge of the key $\tilde{p}_k$ corresponding to $\tilde{P}_k$ of slot k. Therefore, only the owner of slot k can invoke Blame if slot k is disrupted.

Once the relay receives that information the relay broadcasts them. Each client and guard verifies the NIZKP, and if the verification is successful reveals the ℓ-th bit of $PRG(r_{ij})$ for slot k using a non-anonymous signed message. The relay verifies the signature and checks that the values shared by the clients (and servers) are consistent with the c_i values (and Y_j values). If a mismatch is detected then that party is identified as the disrupting party. If no mismatch is detected, the relay compares the $PRG(r_{ij})_\ell$ value sent by the client and $PRG(r_{ij})_\ell$ by the server, for each pair of client and server. If a mismatch is detected there, the client needs to prove that the rij value was indeed sent by the server during the setup phase, and the server needs to prove that the rij value it sent was generated correctly. Whoever fails to provide the proof is considered as the disrupting party. Note that, in the last step (and only if the Blame protocol reaches this step), the r_{ij} value between the conflicting client and server is opened.

Equivocation Protection. PriFi achieves defense against equivocation attacks from the relay by utilizing the history of downstream messages. Each client u_i keeps a personal copy of the history h_i. Each upstream message is symmetrically encrypted with a fresh key γ and γ is sent to the relay blinded by a function of the history h_i. The relay can unblind γ and decrypt the message only if all the clients have the same copy of the history. If the relay sends two different downstream messages to two different clients, the relay will not be able to decrypt messages in any subsequent rounds.