A Universally Composable Framework for the Privacy of Email Ecosystems*

Pyrros Chaidos[1], Olga Fourtounelli[1], Aggelos Kiayias[2,3], and Thomas Zacharias[2]

[1] National and Kapodistrian University of Athens, Greece
{pchaidos,folga}@di.uoa.gr
[2] The University of Edinburgh, UK
{akiayias,tzachari}@inf.ed.ac.uk
[3] IOHK, UK

Abstract. Email communication is amongst the most prominent online activities, and as such, can put sensitive information at risk. It is thus of high importance that internet email applications are designed in a privacy-aware manner and analyzed under a rigorous threat model. The Snowden revelations (2013) suggest that such a model should feature a *global adversary*, in light of the observational tools available. Furthermore, the fact that protecting metadata can be of equal importance as protecting the communication context implies that end-to-end encryption may be necessary, but it is not sufficient.

With this in mind, we utilize the Universal Composability framework [Canetti, 2001] to introduce an expressive cryptographic model for email "ecosystems" that can formally and precisely capture various well-known privacy notions (unobservability, anonymity, unlinkability, etc.), by parameterizing the amount of leakage an ideal-world adversary (simulator) obtains from the email functionality.

Equipped with our framework, we present and analyze the security of two email constructions that follow different directions in terms of the efficiency vs. privacy tradeoff. The first one achieves optimal security (only the online/offline mode of the users is leaked), but it is mainly of theoretical interest; the second one is based on parallel mixing [Golle and Juels, 2004] and is more practical, while it achieves anonymity with respect to users that have similar amount of sending and receiving activity.

1 Introduction

During the last decade, internet users increasingly engage in interactions that put their sensitive information at risk. Social media, e-banking, e-mail, and e-government, are prominent cases where personal data are collected and processed in the web. To protect people's personal data, it is important that applications

* This is the full version of [8] that appears in the proceedings of the ASIACRYPT 2018 conference, in Brisbane, Australia. The said work was supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 653497 (project PANORAMIX).

intended for communication of such information over the internet are designed in a privacy-aware manner and analyzed under a rigorous threat model.

The recent revelations by Snowden (2013) on massive surveillance of citizens' internet interactions, confirmed researchers' views that current technology is sufficient to provide adversaries with the tools to monitor the entire network. This was a turning point in that, henceforth, treating internet security and privacy in a threat model that considers a *global adversary* seems not only desirable, but imperative for the design of state-of-the-art cryptographic protocols.

As far as standard security is concerned, i.e., hiding the context between communicating internet users, there have been significant advancements on the aforementioned matter, mainly to due to wide deployment of end-to-end (E2E) encryption tools, even for some of the world's most popular applications, such as WhatsApp, Viber, Facebook Messenger and Skype (over Signal). However, it is well understood that E2E encryption is not enough to protect the users' metadata (e.g. users' identities and location, or the communication time), that often can be of equal importance. The protection of metadata is studied in the context of *anonymous communications*, that were introduced by the seminal works of Chaum with the concept of mix-nets [11], followed by DC-nets a few years later [9]. A mix-net is a message transmission system that aims to decouple the relation of senders to receivers by using intermediate servers to re-encrypt and re-order messages. The operation of mix-nets relies on messages from A to B making intermediate stops in mix servers, with appropriate delay times so that multiple messages "meet" at each server. The server re-encrypts messages before forwarding them, thus breaking the link between incoming and outgoing messages. We will analyse a mix-based system in Sect. 6 and contrast its overhead to the more expensive broadcast solution in Sect. 5. Nowadays, the most scalable solutions of anonymous communications in the real-world rely on *onion-routing* [34], and mostly on the Tor anonymous browser [17]. Although very efficient and a major step forward for privacy-preserving technologies, it has been pointed out (e.g., [22,33,35]) that onion-routing can provide anonymity only against adversaries with local views with respect to the (three) relay routing nodes, whereas a global observer can easily derive the addresses of two entities that communicate over onion-routing applications. Towards the goal of communication anonymity against a global adversary [2,10,12–14,25–27,30,36], various schemes have been proposed, and several recent ones achieving reasonable latency [1,10,26,27,30,36].

Modeling privacy for email ecosystems. In this work, we focus on the study of privacy (as expressed via several anonymity-style notions cf. [28]) for email ecosystems. The reason why we choose to focus on the email case is threefold:

1. Email is one of the most important aspects of internet communication, as email traffic is estimated to be in the order of $\sim 10^{11}$ messages per day, while there are approximately 2.5 billion accounts worldwide [4].

[4] https://www.radicati.com/wp/wp-content/uploads/2014/10/Email-Market-2014-2018-Executive-Summary.pdf

2. The actual network infrastructure of an email ecosystem has some special features that encourage a separate study from the general case of private messaging. Namely, the users dynamically register, go online/offline, and communicate, in a client-friendly environment and the management of the protocol execution is mainly handled by their service providers (SPs) that manage their inboxes. In turn, the client interface allows the user to log in/log out and while online, submit send and fetch requests to their SP. Moreover, adding a subsystem of mix-nodes which, in principle, are functionally different than the clients and the SPs, stratifies the observational power of the global adversary into three layers (i) the client↔SP channels, (ii) the SP↔mix-node channels, and (iii) the channels within the mix-node system. Under this real-world setting, exploring the feasibility and the trade off between efficiency and privacy for anonymous email routing poses restrictions on the expected secrecy, that would not be present in a generic peer-to-peer setting (e.g. users jointly engaging in an MPC execution).

3. To the best of our knowledge, there is no prior work on general modeling of email privacy in a computational model, that captures protocol flow under a composition of individual email messaging executions. The *Universal Composability (UC)* framework [6] is the ideal tool for such a modeling.

Contributions. Our contributions are as follows:

1. In Section 3, we introduce a framework for the formal study of email ecosystems in the real-ideal world paradigm of the UC model [6]. The real-world entities involved in our framework comprise the set of clients, the of SPs and the subsystem of mix-nodes; all entities are synchronized via a *global clock* functionality $\mathcal{G}_{\mathsf{clock}}$ and communicate over an *authenticated channel functionality with bounded message delay* Δ_{net}, denoted by $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}$. In the ideal-world, an *email privacy functionality* $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}$ manages email traffic among dummy parties that forward their inputs. The functionality is parameterized by Δ_{net} and a *leakage function* Leak, defined over the history transcript, that formally expresses the underlying privacy notion the studied email ecosystem should satisfy. To illustrate the expressibility of our framework, in Section 4, we show how to formally capture intuitively well understood privacy notions by properly defining the leakage function. In particular, we express and study the relation of notions of anonymity, unlinkability, unobservability and pseudonymity defined in [28], as well as E2E encryption, and a notion we call *weak anonymity* that, although a relaxed version of standard anonymity (still stronger than E2E encryption), provides reasonable privacy guarantees given the setting.

2. In Section 5, we present and formally analyze a theoretical construction with quadratic communication overhead that we prove it achieves unobservability (i.e, only the online/offline mode of the clients is leaked), which we argue that it sets the optimal level of privacy that can be expected under the restrictions posed in our client-SP setting, even against a global adversary that only observes the network. As a result, the said construction shows that in principle, optimal privacy is feasible, while the challenge of every real-world email ecosystem is to balance the privacy vs. efficiency trade off.

3. In Sections 6 and 7 we analyze a construction similar to the classical parallel mix of Golle and Juels [19], to illustrate the expressiveness of our model in a more practice-oriented protocol. We focus on the UC simulation in Section 6, and in Section 7, we use Håstad's matrix shuffle to model the permutation's distribution. This in turn makes our analysis relevant to Atom [26], a state of the art anonymity system using similar permutation strategies. At the same time, as we only assume an adversary that is a global passive observer, Atom's techniques to mitigate corruptions are complementary, even if orthogonal, to our work.

2 Background

2.1 Notation

We use λ as the security parameter and write $\mathsf{negl}(\lambda)$ to denote that some function $f(\cdot)$ is negligible in λ. We write $[n]$ to denote the set $\{1, \ldots, n\}$ and $[\![\cdot]\!]$ to denote a multiset. By $X \approx_\epsilon Y$, we denote that the random variable ensembles $\{X_\lambda\}_{\lambda\in\mathbb{N}}, \{Y_\lambda\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable with error $\epsilon(\cdot)$, i.e., for every probabilistic polynomial time (PPT) algorithm $\mathcal{A}$, it holds that

$$\left| \Pr[w \leftarrow X_\lambda : \mathcal{A}(w) = 1] - \Pr[w \leftarrow Y_\lambda : \mathcal{A}(w) = 1] < \epsilon(\lambda) \right| .$$

We simply write $X \approx Y$ when the error ϵ is $\mathsf{negl}(\lambda)$. The notation $x \xleftarrow{\$} S$ stands for x being sampled from the set S uniformly ar random.

2.2 IND-CPA security of public-key encryption schemes

In our constructions, we utilize public-key encryption (PKE). We require that a PKE scheme $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies the property of *multiple challenge IND-CPA (m-IND-CPA) security*, which is equivalent to standard IND-CPA security (up to negligible error). We recall that m-IND-CPA with error $\epsilon(\lambda)$ dictates that any adversary $\mathcal{B}$ that (a) obtains the public key, and (b) sends (polynomially many) challenge queries of the form (M_0, M_1) to the challenger receiving encryption of M_b, where b is the random bit of the challenger, can not guess b with more than $1/2 + \epsilon(\lambda)$ probability.

2.3 Related Work

Early works treating anonymity followed the intuitive definition of Pfitzmann and Hansen (formerly Khöntopp) [28], originally [29], as "the state of not being identifiable within a set of subjects", and aimed to augment it by quantifying the degree of non-identifiability. One of the first efforts in that direction (predating [29]) was the concept of "k-anonymity" by Samarati and Sweeney [31], that (in the context of databases) attempts to identify an individual produce at least k candidates.

In [16,32], anonymity is quantified by measuring the probability that a message M was sent by a user U.Thus, we are no longer interested only in the size of the

set of candidates, but also their relative probabilities. This definition improved upon the "folklore" metric of only measuring the size of the subject set, even if the probability distribution on that set was highly non-uniform –e.g. [24].

The seminal work of Dwork [18] on Differential Privacy, while originating in the realm of databases, highlights and formalizes the strength of combining different pieces of seemingly privacy-respecting information to arrive at a privacy-impacting conclusion. Influenced in part by Differential Privacy, AnoA [3] is a game-based privacy analysis framework that is flexible enough to model various privacy concepts.Their approach is based on games between a challenger and an adversary who has control of the execution of the game, apart from a challenge message representing the scenarios the adversary is trying to distinguish.

In a different direction, the Universal Composability (UC) framework, [6] models security as a simulation not against an adversary, but a malicious environment, given strong control over the inputs of each party as well as a complete view of the system. This rigorous approach produces strong and composable security guarantees but is quite demanding in that the simulation must operate with the bare minimum of data (i.e. what we assume the protocol leaks). This precision in both simulation and leakage is a key motivation of this work.

On the other hand, state of the art anonymous communication solutions such as Loopix [30] which aims for high performance while maintaining strong anonymity properties, as well as unobservability, are analyzed under a weaker adversary. Moreover, Atom [26] is engineered to provide statistical indistinguishable shuffling with strong safeguards against malicious servers, but lacks formal proofs. In our work, we analyze a construction that shares a similar design (namely Håstad's matrix shuffle), so that we are able to offer a suggested T value (i.e mix length) as a side contribution in Section 7. A key difference between Loopix and Atom is that Loopix uses a free routing approach (i.e a message's path is determined by its sender) as opposed to allowing mix nodes to route messages. The first approach is more agreeable with high-efficiency solution aiming for a practical level of resilience against active adversaries while the second approach is easier to reason about but requires a passive adversary or measures such as NIZKs or trap messages to ensure correct behavior.

Camenish and Lysyanskaya [5] offer a treatment of onion routing in the Universal Composability model. The defining characteristic of onion routing, is that routing is entirely determined by the initial sender and is not influenced by the intervening nodes. As such, their analysis focuses on defining security with regards to the encryption, padding, structuring and layering of onions rather than the routing strategy itself. This is orthogonal to our approach: we focus on evaluating the anonymity of different mixing strategies under what we view as realistic requirements about the message encapsulation.

Wikström [37] covers the UC-security of a specific mix construction. His analysis is well-suited to voting but is hard to generalize over other use cases and performance parameters. In contrast, our work, while focusing on email, is more general and flexible in regards to leakage, timings and network topology.

In the work of Alexopoulos *et al.* [1], anonymity is studied in the concept of messaging via a stand-alone simulation-based model. Even though formally treated, anonymity in [1] is defined under a framework that is weaker than UC.

3 A UC framework for the privacy of email ecosystems

In this section, we present our UC framework for email privacy. As in standard UC approach, privacy will be defined via the inability of an environment $\mathcal{Z}$, that schedules the execution and provides the inputs, to distinguish between (i) a real-world execution of an email ecosystem $\mathbb{E}$ in the presence of a (global passive) adversary $\mathcal{A}$ and (ii) an ideal-world execution handled by an email privacy functionality interacting with a PPT simulator Sim. More specifically, we adjust our definitions to the *global UC* setting [7], by incorporating a global clock functionality (cf. [4, 23]) that facilitates synchronicity and is accessed by all parties, including the environment.

3.1 Entities and protocols of an email ecosystem

The entities that are involved in a private email "ecosystem" $\mathbb{E}$ are the following:

- The *service providers (SPs)* $\mathsf{SP}_1, \ldots, \mathsf{SP}_N$ that register users and are responsible for mailbox management and email transfer.
- The *clients* $C_1, \ldots, C_n$ that wish to exchange email messages and are registered to the SPs. For simplicity, we assume that each client is linked with only one SP that manages her mailbox. We write $C_\ell@\mathsf{SP}_i$ to denote that C_ℓ is registered to SP_i, where registration is done dynamically. We define the set $\mathbf{C}_i := \{C_\ell \mid C_\ell@\mathsf{SP}_i\}$ of all clients whose mailboxes SP_i is managing.
- The *mix node* subsystem $\mathbf{MX}$ that consists of the mix nodes $\mathsf{MX}_1, \ldots, \mathsf{MX}_m$ and is the core of the anonymous email routing mechanism.

An email ecosystem $\mathbb{E}$ has the two following phases:

■ **Initialization** is a setup phase where all SPs and mix nodes generate any possible private and public data, and commune their public data to a subset of the ecosystem's entities.

■ **Execution** is a phase that comprises executions of the following protocols:

- The $\mathsf{REGISTER}$ protocol between client C_s and her service provider SP_i. For simplicity, we assume that registration can be done only once.
- The SEND protocol between client C_s and her service provider SP_i. In particular, C_s that wishes to send a message M to some client address $C_r@\mathsf{SP}_j$ authenticates to SP_i and provides her with an encoding $\mathsf{Encode}(M, C_r@\mathsf{SP}_j)$ of $(M, C_r@\mathsf{SP}_j)$ (that may not necessarily include information about the sender). At the end of the protocol, $\mathsf{Encode}(M, C_r@\mathsf{SP}_j)$ is at the outbox of $C_s@\mathsf{SP}_i$ managed by SP_i.
- The ROUTE protocol that is executed among $\mathsf{SP}_1, \ldots, \mathsf{SP}_N$ and $\mathsf{MX}_1, \ldots, \mathsf{MX}_m$. Namely, the encoded message $\mathsf{Encode}(M, C_r@\mathsf{SP}_j)$ is forwarded to the $\mathbf{MX}$ subsystem, which in turn delivers it to SP_j that manages the inbox of C_r.

– The $\mathsf{RECEIVE}$ protocol between client C_r and her service provider SP_j, where C_r can retrieve the messages from the inbox of $C_r@\mathsf{SP}_j$ via fetch requests.

Remark 1. In this work, we consider email solutions that follow the realistic client-side approach, where the client-side operations are relatively simple and do not include complex interaction with the other entities for the execution of heavy cryptographic primitives (e.g. pairwise secure MPC). As we will explain shortly, the client-friendly approach poses some limitations on the privacy level that the email ecosystem can achieve.

3.2 A global clock functionality

In our setting, the protocol flow within the email ecosystem $\mathbb{E}$ advances in *time slots*, that could refer to any suitable time unit (e.g. ms). The entities of $\mathbb{E}$ are synchronized via a *global clock functionality* $\mathcal{G}_{\mathsf{clock}}$ that interacts with a set of parties $\mathbf{P}$, a set of functionalities $\mathbf{F}$, the UC environment $\mathcal{Z}$ and the adversary $\mathcal{A}$. In the spirit of [4,23], the functionality $\mathcal{G}_{\mathsf{clock}}$, presented in Fig. 1, advances when all entities in $\mathbf{P}$ and $\mathbf{F}$ declare completion of their activity within the current time slot, whereas all entities have read access to it.

The global clock functionality $\mathcal{G}_{\mathsf{clock}}(\mathbf{P}, \mathbf{F})$.

The functionality initializes the global clock variable as $\mathsf{Cl} \leftarrow 0$ and the set of advanced parties as $L_{\mathsf{adv}} \leftarrow \emptyset$.

– Upon receiving $(\mathsf{sid}, \textsc{Advance\_Clock}, P)$ from $\mathcal{F} \in \mathbf{F}$ or $P \in \mathbf{P}$, if $P \notin L_{\mathsf{adv}}$, then it adds P to L_{adv} and sends the message $(\mathsf{sid}, \textsc{Advance\_Ack}, P)$ to $\mathcal{F}$ or P, repsectively, and notifies $\mathcal{A}$ by forwarding $(\mathsf{sid}, \textsc{Advance\_Clock}, P)$. If $L_{\mathsf{adv}} = \mathbf{P}$, then it updates as $\mathsf{Cl} \leftarrow \mathsf{Cl} + 1$ and resets $L_{\mathsf{adv}} \leftarrow \emptyset$.
– Upon receiving $(\mathsf{sid}, \textsc{Read\_Clock})$ from $X \in \mathbf{P} \cup \mathbf{F} \cup \{\mathcal{Z}, \mathcal{A}\}$, then it sends $(\mathsf{sid}, \textsc{Read\_Clock}, \mathsf{Cl})$ to X.

Fig. 1. The global clock functionality $\mathcal{G}_{\mathsf{clock}}(\mathbf{P}, \mathbf{F})$ interacting with the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$.

3.3 A UC definition of e-mail privacy

Let $\mathbf{Ad}$ be the set of all valid email addresses linking the set of clients $\mathbf{C} = \{C_1, \ldots, C_n\}$ and with their corresponding providers in $\mathbf{SP} = \{\mathsf{SP}_1, \ldots, \mathsf{SP}_N\}$, i.e. $\mathbf{Ad} := \cup_{i \in [N]} \mathbf{C}_i$. We denote by $\mathbf{P}$ the union $\mathbf{C} \cup \mathbf{SP} \cup \mathbf{MX}$.

The *history* of an email ecosystem execution that involves the entities in $\mathbf{C}, \mathbf{SP}$ and $\mathbf{MX}$ is a transcript of actions expressed as a list H, where each action entry of H is associated with a unique pointer ptr to this action. The leakage in each execution step, is expressed via a *leakage function* $\mathsf{Leak}(\cdot, \cdot)$ that, when given as input (i) a pointer ptr and (ii) an execution history sequence H, outputs some

leakage string z. Here, z could be $\perp$ indicating no leakage to the adversary. This leakage may depend on the entry indexed by ptr as well as on entries recorded previously (i.e. prior than ptr).

We require that during a time slot, the environment sends a message for every party, even when the party is idle (inactive) for this slot, so that the clock can be advanced as described in Fig. 1.

The ideal world execution. In the ideal world, the protocol execution is managed by the *email privacy functionality* $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}(\mathbf{P})$, parameterized by the message delivery delay bound Δ_{net} and the leakage function $\mathsf{Leak}(\cdot,\cdot)$, with access to $\mathcal{G}_{\mathsf{clock}}$. The functionality $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}(\mathbf{P})$ consists of the **Initialization**, **Execution**, and **Clock advancement** phases, that informally are run as follows:

– At the **Initialization** phase, all the SPs in **SP** and mix nodes in **MX** provide $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}(\mathbf{P})$ with an initialization notification via public delayed output. The functionality proceeds to the **Execution** phase when all SPs and mix nodes are initialized. Note that in the ideal world, the SPs and the mix nodes remain idle after **Initialization** (besides messages intended for $\mathcal{G}_{\mathsf{clock}}$), as privacy-preserving email routing is done by $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}(\mathbf{P})$. Their presence in the ideal setting is for consistency in terms of UC interface.

– At the **Execution** phase, $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}(\mathbf{P})$ manages the email traffic, as scheduled per time slot by the environment. During this phase, the clients may (dynamically) provide $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}(\mathbf{P})$ with registration, log in, log out, send or fetch requests. Upon receiving a request, $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}(\mathbf{P})$ updates the history by adding the request as a new entry associated with a unique pointer ptr, in a '$\mathsf{pending}$' mode. Then, it notifies the simulator Sim by attaching the corresponding leakage. The execution of a pending request which record is indexed by a pointer ptr is completed when $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}(\mathbf{P})$ receives an Allow_Exec message paired with ptr from Sim.

Within a time slot T, each client may perform only one action that also implies a time advancement request to $\mathcal{G}_{\mathsf{clock}}$. In order for the clock to advance all the other parties that performed no action (i.e., the SPs, the mix nodes and the clients that remained idle during T), send an explicit time advancement request to $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}(\mathbf{P})$. Besides, any party may submit clock reading requests arbitrarily. All the messages that are intended for $\mathcal{G}_{\mathsf{clock}}$ are forwarded to it by $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}(\mathbf{P})$.

– At the **Clock advancement** phase, all parties have already submitted time advancement requests during time slot T, so $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}(\mathbf{P})$ takes the necessary final steps before proceeding to $T+1$. In particular, $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}(\mathbf{P})$ completes the execution of all send and fetch requests that have been delayed for Δ_{net} steps (by Sim). This suggests that in the ideal-world, the delay in message delivery is upper bounded by Δ_{net}. Finally, $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}(\mathbf{P})$ informs Sim of the leakage derived from the aforementioned executions, advances its local time by 1 and reenters the **Execution** phase for time slot $T+1$.

Formally, the email privacy functionality $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}(\mathbf{P})$ is described as follows:

Initialization on status 'init'.

- $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}(\mathbf{P})$ sets its status to 'init'. It initializes the set of valid addresses $\mathbf{Ad}$, the set of active entities L_{act}, the set of clock-advanced entities L_{adv}, the *history* list H, and the set of leaked entries L_{leak} as empty.
- Upon receiving $(\mathsf{sid}, \textsc{Init})$ from a party $P \in \mathbf{SP} \cup \mathbf{MX}$, if $L_{\mathsf{act}} \subsetneq \mathbf{SP} \cup \mathbf{MX}$, then it sends the message $(\mathsf{sid}, \textsc{Init}, P)$ to Sim.
- Upon receiving $(\mathsf{sid}, \textsc{Allow\_Init}, P)$ from Sim, if $P \in (\mathbf{SP} \cup \mathbf{MX}) \setminus L_{\mathsf{act}}$, then it adds P to L_{act}. If $L_{\mathsf{act}} = \mathbf{SP} \cup \mathbf{MX}$, then it sends $(\mathsf{sid}, \mathsf{ready})$ to Sim.
- Upon receiving $(\mathsf{sid}, \textsc{Execute})$ from Sim, it sends $(\mathsf{sid}, \textsc{Read\_Clock})$ to $\mathcal{G}_{\mathsf{clock}}$.
- Upon receiving $(\mathsf{sid}, \textsc{Read\_Clock}, \mathsf{Cl})$ from $\mathcal{G}_{\mathsf{clock}}$, it sets its clock as Cl and its status to 'execute', and sends the message $(\mathsf{sid}, \mathsf{start}, \mathsf{Cl})$ to Sim.

Execution on status 'execute'.

Registration:

- Upon receiving $\big(\mathsf{sid}, \textsc{Register}, @\mathsf{SP}_i\big)$ from C_ℓ, if for every $j \in [N] : C_\ell@\mathsf{SP}_j \notin \mathbf{Ad}$ and $C_\ell \notin L_{\mathsf{adv}}$, then
  1. It sends the message $(\mathsf{sid}, \textsc{Advance\_Clock}, C_\ell)$ to $\mathcal{G}_{\mathsf{clock}}$.
  2. Upon receiving $(\mathsf{sid}, \textsc{Advance\_Ack}, C_\ell)$ from $\mathcal{G}_{\mathsf{clock}}$, it adds C_ℓ to L_{adv} and the entry $\big(\mathsf{ptr}, \big(\mathsf{sid}, \mathsf{Cl}, \textsc{Register}, C_\ell@\mathsf{SP}_i\big), \text{'}\mathsf{pending}\text{'}\big)$ to H.
  3. It sends the message $\big(\mathsf{sid}, \mathsf{ptr}, \mathsf{Leak}(\mathsf{ptr}, H)\big)$ to Sim.
  4. Upon receiving $\big(\mathsf{sid}, \textsc{Allow\_Exec}, \mathsf{ptr}\big)$ from Sim, if ptr refers to an entry of the form $\big(\mathsf{ptr}, \big(\mathsf{sid}, \mathsf{Cl}, \textsc{Register}, C_\ell@\mathsf{SP}_i\big), \text{'}\mathsf{pending}\text{'}\big)$, then
     (a) It adds $C_\ell@\mathsf{SP}_i$ to $\mathbf{Ad}$ and L_{act}, and initializes a list $\mathsf{Inbox}[C_\ell@\mathsf{SP}_i]$ as empty.
     (b) It updates the entry as $\big(\mathsf{ptr}, \big(\mathsf{sid}, \mathsf{Cl}', \textsc{Register}, C_\ell@\mathsf{SP}_i\big), \text{'}(\mathsf{registered}, \mathsf{Cl})\text{'}\big)$.
     (c) It sends the message $\big(\mathsf{sid}, \mathsf{ptr}, \mathsf{Leak}(\mathsf{ptr}, H)\big)$ to Sim.

Log in:

- Upon receiving $\big(\mathsf{sid}, \textsc{Active}, @\mathsf{SP}_i\big)$ from C_ℓ, if $C_\ell@\mathsf{SP}_i \in \mathbf{Ad}$ and $C_\ell \notin L_{\mathsf{adv}}$,
  1. It sends the message $(\mathsf{sid}, \textsc{Advance\_Clock}, C_\ell)$ to $\mathcal{G}_{\mathsf{clock}}$.
  2. Upon receiving $(\mathsf{sid}, \textsc{Advance\_Ack}, C_\ell)$ from $\mathcal{G}_{\mathsf{clock}}$, it adds C_ℓ to L_{adv} and the entry $\big(\mathsf{ptr}, \big(\mathsf{sid}, \mathsf{Cl}, \textsc{Active}, C_\ell@\mathsf{SP}_i\big), \text{'}\mathsf{pending}\text{'}\big)$ to H.
  3. It sends the message $\big(\mathsf{sid}, \mathsf{ptr}, \mathsf{Leak}(\mathsf{ptr}, H)\big)$ to Sim.
  4. Upon receiving $\big(\mathsf{sid}, \textsc{Allow\_Exec}, \mathsf{ptr}\big)$ from Sim, if ptr refers to an entry of the form $\big(\mathsf{ptr}, \big(\mathsf{sid}, \mathsf{Cl}', \textsc{Active}, C_\ell@\mathsf{SP}_i\big), \text{'}\mathsf{pending}\text{'}\big)$, then
     (a) If $C_\ell@\mathsf{SP}_i \notin L_{\mathsf{act}}$, then it adds $C_\ell@\mathsf{SP}_i$ to L_{act}.
     (b) It updates the entry as $\big(\mathsf{ptr}, \big(\mathsf{sid}, \mathsf{Cl}', \textsc{Active}, C_\ell@\mathsf{SP}_i\big), \text{'}(\mathsf{logged\ in}, \mathsf{Cl})\text{'}\big)$.
     (c) It sends the message $\big(\mathsf{sid}, \mathsf{ptr}, \mathsf{Leak}(\mathsf{ptr}, H)\big)$ to Sim.

Log out:

- Upon receiving $\big(\mathsf{sid}, \textsc{Inactive}, @\mathsf{SP}_i\big)$ from C_ℓ, if $C_\ell@\mathsf{SP}_i \in \mathbf{Ad}$ and $C_\ell \notin L_{\mathsf{adv}}$, then
  1. It sends the message $(\mathsf{sid}, \textsc{Advance\_Clock}, C_\ell)$ to $\mathcal{G}_{\mathsf{clock}}$.
  2. Upon receiving $(\mathsf{sid}, \textsc{Advance\_Ack}, C_\ell)$ from $\mathcal{G}_{\mathsf{clock}}$, it adds C_ℓ to L_{adv} and the entry $\big(\mathsf{ptr}, \big(\mathsf{sid}, \mathsf{Cl}, \textsc{Inactive}, C_\ell@\mathsf{SP}_i\big), , \text{'}\mathsf{pending}\text{'}\big)$ to H.
  3. It sends the message $\big(\mathsf{sid}, \mathsf{ptr}, \mathsf{Leak}(\mathsf{ptr}, H)\big)$ to Sim.

4. Upon receiving $(\mathsf{sid}, \textsc{Allow\_Exec}, \mathsf{ptr})$ from Sim, if ptr refers to an entry of the form $(\mathsf{ptr}, (\mathsf{sid}, \mathsf{Cl}', \textsc{Inactive}, C_\ell@\mathsf{SP}_i), \text{`pending'})$, then
   (a) If $C_\ell@\mathsf{SP}_i \in L_{\mathsf{act}}$, then it deletes $C_\ell@\mathsf{SP}_i$ from L_{act}.
   (b) It updates the entry as $(\mathsf{ptr}, (\mathsf{sid}, \mathsf{Cl}', \textsc{Inactive}, C_\ell@\mathsf{SP}_i), \text{`}(\mathsf{logged\ out}, \mathsf{Cl})\text{'})$.
   (c) It sends the message $(\mathsf{sid}, \mathsf{ptr}, \mathsf{Leak}(\mathsf{ptr}, H))$ to Sim.

Send:

– Upon receiving $(\mathsf{sid}, \textsc{Send}, \langle C_s@\mathsf{SP}_i, M, C_r@\mathsf{SP}_j\rangle)$ from C_s, if $C_s@\mathsf{SP}_i, C_r@\mathsf{SP}_j \in \mathbf{Ad}$ and $C_s \in L_{\mathsf{act}} \setminus L_{\mathsf{adv}}$, then
   1. It sends the message $(\mathsf{sid}, \textsc{Advance\_Clock}, C_s)$ to $\mathcal{G}_{\mathsf{clock}}$.
   2. Upon receiving $(\mathsf{sid}, \textsc{Advance\_Ack}, C_s)$ from $\mathcal{G}_{\mathsf{clock}}$, it adds C_s in L_{adv} and the entry $(\mathsf{ptr}, (\mathsf{sid}, \mathsf{Cl}, \textsc{Send}, \langle C_s@\mathsf{SP}_i, M, C_r@\mathsf{SP}_j\rangle), \text{`pending'})$ to H.
   3. It sends the message $(\mathsf{sid}, \mathsf{ptr}, \mathsf{Leak}(\mathsf{ptr}, H))$ to Sim.
   4. Upon receiving $(\mathsf{sid}, \textsc{Allow\_Exec}, \mathsf{ptr})$ from Sim, if ptr refers to an entry $(\mathsf{sid}, \mathsf{Cl}', \textsc{Send}, \langle C_s@\mathsf{SP}_i, M, C_r@\mathsf{SP}_j\rangle)$ with status 'pending', then
      (a) It adds $(\mathsf{sid}, \mathsf{Cl}', \textsc{Send}, \langle C_s@\mathsf{SP}_i, M, C_r@\mathsf{SP}_j\rangle)$ to $\mathsf{Inbox}[C_r@\mathsf{SP}_j]$.
      (b) It updates as $(\mathsf{ptr}, (\mathsf{sid}, \mathsf{Cl}', \textsc{Send}, \langle C_s@\mathsf{SP}_i, M, C_r@\mathsf{SP}_j\rangle), \text{`}(\mathsf{sent}, \mathsf{Cl})\text{'})$.
      (c) It sends the message $(\mathsf{sid}, \mathsf{ptr}, \mathsf{Leak}(\mathsf{ptr}, H))$ to Sim.

Fetch:

– Upon receiving $(\textsc{Fetch}, \mathsf{sid}, C_r@\mathsf{SP}_j)$ from C_r, if $C_r@\mathsf{SP}_j \in \mathbf{Ad}$ and $C_r \in L_{\mathsf{act}} \setminus L_{\mathsf{adv}}$, then
   1. It sends the message $(\mathsf{sid}, \textsc{Advance\_Clock}, C_r)$ to $\mathcal{G}_{\mathsf{clock}}$.
   2. Upon receiving $(\mathsf{sid}, \textsc{Advance\_Ack}, C_r)$ from $\mathcal{G}_{\mathsf{clock}}$, it adds C_s in L_{adv} and the entry $(\mathsf{ptr}, (\mathsf{sid}, \mathsf{Cl}, \textsc{Fetch}, C_r@\mathsf{SP}_j), \text{`pending'})$ to H.
   3. It sends the message $(\mathsf{sid}, \mathsf{ptr}, \mathsf{Leak}(\mathsf{ptr}, H))$ to Sim.
   4. Upon receiving $(\mathsf{sid}, \textsc{Allow\_Exec}, \mathsf{ptr})$ from Sim, if ptr refers to an entry of the form $(\mathsf{sid}, \mathsf{Cl}', \textsc{Fetch}, C_r@\mathsf{SP}_j)$ with status 'pending', then
      (a) It sends the message $(\mathsf{sid}, \mathsf{Inbox}[C_r@\mathsf{SP}_j])$ to C_r.
      (b) It updates the entry as $(\mathsf{ptr}, (\mathsf{sid}, \mathsf{Cl}', \textsc{Fetch}, C_r@\mathsf{SP}_j), \text{`}(\mathsf{fetched}, \mathsf{Cl})\text{'})$.
      (c) It resets $\mathsf{Inbox}[C_r@\mathsf{SP}_j]$ as empty.
      (d) It sends the message $(\mathsf{sid}, \mathsf{ptr}, \mathsf{Leak}(\mathsf{ptr}, H))$ to Sim.

Clock reading:

– Upon receiving $(\mathsf{sid}, \textsc{Read\_Clock})$ from a party $P \in \mathbf{P}$, then
   1. It sends the message $(\mathsf{sid}, \textsc{Read\_Clock})$ to $\mathcal{G}_{\mathsf{clock}}$.
   2. On receiving $(\mathsf{sid}, \textsc{Read\_Clock}, \mathsf{Cl})$ from $\mathcal{G}_{\mathsf{clock}}$ it adds $(\mathsf{ptr}, (\mathsf{sid}, \mathsf{Cl}, \textsc{Read\_Clock}, P))$ to H, sending $(\mathsf{sid}, \textsc{Read\_Clock}, \mathsf{Cl})$ to P.
   3. It sends the message $(\mathsf{sid}, \mathsf{ptr}, \mathsf{Leak}(\mathsf{ptr}, H))$ to Sim.

Clock advance:

– Upon receiving $(\mathsf{sid}, \textsc{Advance\_Clock})$ from a party $P \in \mathbf{P} \setminus L_{\mathsf{adv}}$, then
   1. It sends the message $(\mathsf{sid}, \textsc{Advance\_Clock}, P)$ to $\mathcal{G}_{\mathsf{clock}}$.
   2. Upon receiving $(\mathsf{sid}, \textsc{Advance\_Ack}, P)$ from $\mathcal{G}_{\mathsf{clock}}$, it adds P in L_{adv} and $(\mathsf{ptr}, (\mathsf{sid}, \mathsf{Cl}, \textsc{Advance\_Clock}, P))$ to H.
   3. It sends the message $(\mathsf{sid}, \mathsf{ptr}, \mathsf{Leak}(\mathsf{ptr}, H))$ to Sim.
   4. If $L_{\mathsf{adv}} = \mathbf{P}$, then it sets its status to 'advance' and proceeds to the **Clock advancement** phase below.

Clock advancement on status 'advance'.

- Upon setting its status to 'advance':
  1. For every history entry of the form $\big(\mathsf{sid}, \mathsf{Cl}', \textsc{Send}, \langle C_s@\mathsf{SP}_i, M, C_r@\mathsf{SP}_j\rangle\big)$ with status 'pending' such that $\mathsf{Cl} - \mathsf{Cl}' = \Delta_{\mathsf{net}}$, it adds this entry to $\mathsf{Inbox}[C_r@\mathsf{SP}_j]$ and updates the entry's status to '$(\mathsf{sent}, \mathsf{Cl})$'.
  2. For every history entry of the form $\big(\mathsf{sid}, \mathsf{Cl}', \textsc{Fetch}, C_r@\mathsf{SP}_j\big)$ with status 'pending' such that $\mathsf{Cl}-\mathsf{Cl}' = \Delta_{\mathsf{net}}$, it sends the message $(\mathsf{sid}, \mathsf{Inbox}[C_r@\mathsf{SP}_j])$ to C_r, resets the list $\mathsf{Inbox}[C_r@\mathsf{SP}_j]$ as input and updates the entry's status to '$(\mathsf{fetched}, \mathsf{Cl})$'.
  3. It sends the message $\big(\mathsf{sid}, \mathsf{ptr}, \mathsf{Leak}(\mathsf{ptr}, H)\big)$ to Sim.
  4. It finalizes execution for the current slot as follows:
     (a) It advances its time by $\mathsf{Cl} \leftarrow \mathsf{Cl} + 1$.
     (b) It adds $\big(\mathsf{ptr}, \big(\mathsf{sid}, \textsc{Clock\_Advanced}\big)\big)$ to H.
     (c) It reverts its status to '$\mathsf{execute}$' and resets L_{adv} to empty.
     (d) It sends the message $(\mathsf{sid}, \textsc{Clock\_Advanced})$ to Sim.

We denote by $\mathrm{EXEC}^{\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}, \Delta_{\mathsf{net}}}}_{\mathsf{Sim}, \mathcal{Z}, \mathcal{G}_{\mathsf{clock}}}[\mathbf{P}](\lambda)$, the output of the environment $\mathcal{Z}$ in an ideal-world execution of $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}, \Delta_{\mathsf{net}}}(\mathbf{P})$ under the presence of Sim.

The authenticated channel functionality $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$.

The functionality initializes a list of pending messages L_{pend} as empty.

- Upon receiving $(\mathsf{sid}, \textsc{Channel}, M, P')$ from $P \in \mathbf{P}$, then
  1. It sends the message $(\mathsf{sid}, \textsc{Read\_Clock})$ to $\mathcal{G}_{\mathsf{clock}}(\mathbf{P})$.
  2. Upon receiving $(\mathsf{sid}, \textsc{Read\_Clock}, \mathsf{Cl})$ to $\mathcal{G}_{\mathsf{clock}}(\mathbf{P})$, it picks a unique pointer ptr and stores the entry $\big(\mathsf{ptr}, (\mathsf{sid}, \mathsf{Cl}, \textsc{Channel}, P, M, P')\big)$ to L_{pend}.
  3. It sends the message $\big(\mathsf{ptr}, (\mathsf{sid}, \textsc{Channel}, P, M, P')\big)$ to $\mathcal{A}$.
- Upon receiving $(\mathsf{sid}, \textsc{Allow\_Channel}, \mathsf{ptr}')$ from $\mathcal{A}$, if there is an entry $\big(\mathsf{ptr}', (\mathsf{sid}, \mathsf{Cl}', \textsc{Channel}, P, M, P')\big)$ in L_{pend}, then it sends the message (sid, M, P) to P' and deletes $\big(\mathsf{ptr}', (\mathsf{sid}, \mathsf{Cl}', \textsc{Channel}, P, M, P')\big)$ from L_{pend}.
- Upon any activation from a party $P \in \mathbf{P}$ or $\mathcal{A}$ as above,
  1. It sends the message $(\mathsf{sid}, \textsc{Read\_Clock})$ to $\mathcal{G}_{\mathsf{clock}}(\mathbf{P})$.
  2. Upon receiving $(\mathsf{sid}, \textsc{Read\_Clock}, \mathsf{Cl})$ to $\mathcal{G}_{\mathsf{clock}}(\mathbf{P})$, it parses L_{pend}. For every entry $\big(\mathsf{ptr}', (\mathsf{sid}, \mathsf{Cl}', \textsc{Channel}, P, M, P')\big)$ s.t. $\mathsf{Cl} - \mathsf{Cl}' = \Delta_{\mathsf{net}}$, it sends the message (sid, M, P) to P' and deletes $\big(\mathsf{ptr}', (\mathsf{sid}, \mathsf{Cl}', \textsc{Channel}, P, M, P')\big)$ from L_{pend}.

Fig. 2. The authenticated channel functionality $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$ interacting with the adversary $\mathcal{A}$.

The $(\mathcal{G}_{\mathsf{clock}}, \mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}})$-hybrid world execution. In the real world email ecosystem $\mathbb{E}$, the clients, the SPs and the mix nodes interact according to the protocols'

guidelines and the environment's instructions. The message delivery is executed via the functionality $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$ described in Fig. 2 that captures the notion of an authenticated channel, upon which a maximum delivery delay Δ_{net} can be imposed. Clock advancement is done via calls to $\mathcal{G}_{\mathsf{clock}}$, which interacts with all entities and $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}$.

We denote by $\mathrm{EXEC}^{\mathbb{E}^{\mathcal{G}_{\mathsf{clock}}}, \mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}}_{\mathcal{A},\mathcal{Z},\mathcal{G}_{\mathsf{clock}}}[\mathbf{P}](\lambda)$ the output of the environment $\mathcal{Z}$ in an execution of $\mathbb{E}^{\mathcal{G}_{\mathsf{clock}}, \mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}}$ under the presence of $\mathcal{A}$.

The UC definition of a private email ecosystem is provided below.

Definition 1 (UC Email Privacy). *Let $\Delta_{\mathsf{net}}, \epsilon$ be non-negative values. Let $\mathbb{E}$ be an email ecosystem with client set $\mathbf{C} = C_1, \ldots, C_n$, service provider set $\mathbf{SP} = \mathsf{SP}_1, \ldots, \mathsf{SP}_N$ and mix node set $\mathbf{MX} = \mathsf{MX}_1, \ldots, \mathsf{MX}_m$. Let $\mathbf{P} := \mathbf{C} \cup \mathbf{SP} \cup \mathbf{MX}$. We say that $\mathbb{E}^{\mathcal{G}_{\mathsf{clock}}, \mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}}$* achieves statistical (resp. computational) ϵ-privacy with respect to leakage (Leak) and message delay Δ_{net}, *if for every unbounded (resp. PPT) global passive adversary $\mathcal{A}$, there is a PPT simulator Sim such that for every PPT environment $\mathcal{Z}$, it holds that*

$$\mathrm{EXEC}^{\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}}_{\mathsf{Sim},\mathcal{Z},\mathcal{G}_{\mathsf{clock}}}[\mathbf{P}](\lambda) \approx_{\epsilon} \mathrm{EXEC}^{\mathbb{E}^{\mathcal{G}_{\mathsf{clock}}}, \mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}}_{\mathcal{A},\mathcal{Z},\mathcal{G}_{\mathsf{clock}}}[\mathbf{P}](\lambda) \ .$$

4 Formalizing privacy notions via types of leakage functions

In [28], Pfitzmann and Hansen provide definitions for anonymity, unlinkability, unobservability and pseudonymity. Even though outside the context of a formal framework, the definitions in this seminal work have served as a reference point by researchers for the understanding of privacy notions. In this section, we formally express the said (yet not only these) notions by carefully specifying a corresponding leakage function.

Basic leakage sets. Below, we define some useful sets that will enable the succinct description of the various leakage functions that we will introduce. In our formalization, leakage will derive from the history entries that are in a '$\mathsf{pending}$' mode. This is due to technical reasons, as the ideal-world simulator Sim (cf. Section 3.3) must be aware of the actions to be taken by the email privacy functionality $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak},\Delta_{\mathsf{net}}}(\mathbf{P})$ *before* allowing their execution, so that it can simulate the real-world run in an indistinguishable manner. In the following, the symbol $*$ denotes a wildcard, and $\mathsf{ptr}' \leq \mathsf{ptr}$ denotes that entry indexed with pointer ptr' was added earlier than the entry with pointer ptr.

– The *active address set for H by pointer* ptr:

$$\mathsf{Act}_{\mathsf{ptr}}[H] =: \Big\{ C_\ell @ \mathsf{SP}_i \;\Big|\; \exists \mathsf{ptr}' \leq \mathsf{ptr} : \Big[\big[\big(\mathsf{ptr}', (\mathsf{sid}, *, \textsc{Active}, C_\ell @ \mathsf{SP}_i), \text{`}\mathsf{pending}\text{'}\big) \in H\big] \vee$$
$$\vee \big[\big(\mathsf{ptr}', (\mathsf{sid}, *, \textsc{Register}, C_\ell @ \mathsf{SP}_i), \text{`}\mathsf{pending}\text{'}\big) \in H\big]\Big] \wedge$$
$$\wedge \Big[\forall \mathsf{ptr}'' : \mathsf{ptr}' \leq \mathsf{ptr}'' \leq \mathsf{ptr} \Rightarrow \big(\mathsf{ptr}'', (\mathsf{sid}, *, \textsc{Inactive}, C_\ell @ \mathsf{SP}_i), \text{`}\mathsf{pending}\text{'}\big) \notin H\Big]\Big\} \ .$$

Note. To simplify the notation and terminology that follows, we consider as active all the addresses that are in a pending registration status.

– The *sender set for H by pointer* ptr:

$$\mathbf{S}_{\mathsf{ptr}}[H] := \left\{ C_s@\mathsf{SP}_i \;\middle|\; \exists \mathsf{ptr}' \leq \mathsf{ptr} : \left(\mathsf{ptr}', \left(\mathsf{sid}, *, \textsc{Send}, \langle C_s@\mathsf{SP}_i, *, * \rangle\right), \text{`}\mathsf{pending}\text{'}\right) \in H \right\} .$$

– The *sender multiset for H by pointer* ptr, denoted by $[\![\mathbf{S}_{\mathsf{ptr}}]\!][H]$, is defined analogously. The difference with $\mathbf{S}_{\mathsf{ptr}}[H]$ is that the cardinality of the pending Send messages provided by $C_s@\mathsf{SP}_i$ is attached.

– The *message set for H by pointer* ptr:

$$\mathbf{M}_{\mathsf{ptr}}[H] := \left\{ M \;\middle|\; \exists \mathsf{ptr}' \leq \mathsf{ptr} : \left(\mathsf{ptr}', \left(\mathsf{sid}, *, \textsc{Send}, \langle *, M, * \rangle\right), \text{`}\mathsf{pending}\text{'}\right) \in H \right\} .$$

– The *message-sender set for H by pointer* ptr:

$$\mathbf{MS}_{\mathsf{ptr}}[H] := \left\{ (M, C_s@\mathsf{SP}_i) \;\middle|\; \exists \mathsf{ptr}' \leq \mathsf{ptr} : \left(\mathsf{ptr}', \left(\mathsf{sid}, *, \textsc{Send}, \langle C_s@\mathsf{SP}_i, M, * \rangle\right), \text{`}\mathsf{pending}\text{'}\right) \in H \right\} .$$

– The *recipient set for H by pointer* ptr:

$$\mathbf{R}_{\mathsf{ptr}}[H] := \left\{ C_r@\mathsf{SP}_j \;\middle|\; \exists \mathsf{ptr}' \leq \mathsf{ptr} : \left(\mathsf{ptr}', \left(\mathsf{sid}, *, \textsc{Send}, \langle *, *, C_r@\mathsf{SP}_j \rangle\right), \text{`}\mathsf{pending}\text{'}\right) \in H \right\} .$$

– The *recipient multiset for H at time slot T*, denoted by $[\![\mathbf{R}_{\mathsf{ptr}}]\!][H]$, is defined analogously. The difference with $\mathbf{R}_{\mathsf{ptr}}[H]$ is that the cardinality of the pending Send messages intended for $C_r@\mathsf{SP}_j$ is attached.

– The *message-recipient set for H by pointer* ptr:

$$\mathbf{MR}_{\mathsf{ptr}}[H] := \left\{ (M, C_r@\mathsf{SP}_j) \;\middle|\; \exists \mathsf{ptr}' \leq \mathsf{ptr} : \left(\mathsf{ptr}', \left(\mathsf{sid}, *, \textsc{Send}, \langle *, M, C_r@\mathsf{SP}_j \rangle\right), \text{`}\mathsf{pending}\text{'}\right) \in H \right\} .$$

– The *set of fetching clients for H by pointer* ptr

$$\mathbf{F}_{\mathsf{ptr}}[H] := \left\{ C_r@\mathsf{SP}_j \;\middle|\; \exists \mathsf{ptr}' \leq \mathsf{ptr} : \left(\mathsf{ptr}, \left(\mathsf{sid}, *, \textsc{Fetch}, C_r@\mathsf{SP}_j\right), \text{`}\mathsf{pending}\text{'}\right) \right\} .$$

Unobservability. Unobservability is the state where "the messages are not discernible from random noise". Here, we focus on the case of *(complete) sender and receiver unobservability*, that we will refer to unobservability for brevity. In this case, the sender and recipient unobservability sets match the set of all online clients, and within these complete unobservability sets, it is neither noticeable if a client sends, nor if it receives as message. Hence, in our setting, unobservability is achieved if only the "client activity bit" is leaked. As a result, we can define the *unobservability leakage function* $\mathsf{Leak}_{\mathsf{unob}}$ as the active address set:

$$\mathsf{Leak}_{\mathsf{unob}}(\mathsf{ptr}, H) := \mathsf{Act}_{\mathsf{ptr}}[H] . \tag{1}$$

Remark 2 (Unobservability as a golden standard for email privacy). In our UC formalization of e-mail ecosystems, we consider a dynamic scenario where the clients register, go online/offline and make custom fetch requests, which is consistent with the real-world dynamics of email communication. It is easy to see that in such a setting the clients' online/offline status may be leaked to a global observer. E.g., the environment may provide send requests to offline clients and notify the global adversary that provided the said requests, so that the latter can check the activity of those clients. Hence, in our framework, unobservability as defined in Eq. (1), sets a "golden standard" for optimal privacy. In Section 5, we show that this golden standard is feasible in principle. Namely, we describe a theoretical construction with quadratic communication complexity and we prove it achieves unobservability. As a result, that construction sets one extreme point in the privacy vs. efficiency trade off for the client-server email infrastructure, the other being a simple and fast network with no security enhancements. Clearly, the challenge of every email construction is to balance the said trade off between these two extreme points.

We conclude our remark noting that a higher level privacy (e.g., no leakage at all) could be possible if we considered an alternative setting where the email addresses are a priori given, the clients are always online and mail delivery is via continuous push by the SPs. However, we believe that such a setting is restrictive for formally capturing what is an email ecosystem in general.

Anonymity. According to [28], *anonymity* "is the state of being not identifiable within a set of subjects, the anonymity set". In the email scenario, a sender (resp. recipient) should be anonymous within the set of potential senders (resp. recipients), i.e. the *sender* (resp. *recipient*) *anonymity set*. In addition, anonymity sets may change over time, which in our framework is done via global clock advancement and per slot. We recall from the discussion in Remark 2 that in our setting, the anonymity sets are restricted within the set of online users.

We define the predicate $\mathsf{End}(\cdot,\cdot)$ over the pointers and history transcripts to denote that a pointer ptr refers to the last history entry before the functionality enters the **Clock advancement** phase in order to finalize execution for the running time slot. By the above, we define the *anonymity leakage function*, $\mathsf{Leak}_{\mathsf{anon}}$, as follows:

$$\mathsf{Leak}_{\mathsf{anon}}(\mathsf{ptr}, H) := \begin{cases} \big(\mathbf{S}_{\mathsf{ptr}}[H], \mathbf{R}_{\mathsf{ptr}}[H], \mathsf{Act}_{\mathsf{ptr}}[H]\big), & \text{if } \mathsf{End}(\mathsf{ptr}, H) = 1 \\ \mathsf{Act}_{\mathsf{ptr}}[H], & \text{otherwise} \end{cases} \tag{2}$$

Unlinkability. Unlinkability of items of interest (e.g. subjects, messages, etc.) means that "the ability of the attacker to relate these items does not increase by observing the system". In [28] several anonymity variants are defined in terms of unlinkability. Below, we propose a formalization of the said notions[5].

[5] In the proceedings version of this work [8], the terms of sender-side/recipient-side unlinkability are used instead of sender/recipient anonymity, respectively. Here, we choose to be closer to the terminology of [28].

– *Sender anonymity:* a particular message can not be linked to any sender and to a particular sender, no message is linkable. We define the *sender anonymity leakage function* $\mathsf{Leak}_{\mathsf{s.anon}}$ as

$$\mathsf{Leak}_{\mathsf{s.anon}}(\mathsf{ptr}, H) := \begin{cases} \big(\mathbf{S}_{\mathsf{ptr}}[H], \mathbf{MR}_{\mathsf{ptr}}[H], \mathsf{Act}_{\mathsf{ptr}}[H]\big), & \text{if } \mathsf{End}(\mathsf{ptr}, H) = 1 \\ \mathsf{Act}_{\mathsf{ptr}}[H], & \text{otherwise} \end{cases} \tag{3}$$

Sender anonymity is a useful notion to capture the level of privacy desired in an e-voting process. Namely, all voters (senders) provide their votes (messages) in an encrypted form to a single known recipient server, playing the role of the ballot box. During tally, the ballot box opens so that the votes are counted, yet the link between the vote and the voter is should be broken so that privacy is preserved.

– *Recipient anonymity:* a particular message can not be linked to any recipient and to a particular recipient, no message is linkable. We define the *recipient anonymity leakage function* $\mathsf{Leak}_{\mathsf{r.anon}}$ as

$$\mathsf{Leak}_{\mathsf{r.anon}}(\mathsf{ptr}, H) := \begin{cases} \big(\mathbf{MS}_{\mathsf{ptr}}[H], \mathbf{R}_{\mathsf{ptr}}[H], \mathsf{Act}_{\mathsf{ptr}}[H]\big), & \text{if } \mathsf{End}(\mathsf{ptr}, H) = 1 \\ \mathsf{Act}_{\mathsf{ptr}}[H], & \text{otherwise} \end{cases} \tag{4}$$

As an interesting example for the direction of recipient-anonymity, suppose that an individual wishes to make several donations amongst a number of known charities. Recipient-anonymity ensures that even if the sender wishes to disclose the amounts, this is done without necessarily disclosing which the charity received which amount.

– *Relationship anonymity:* it is untraceable who communicates with whom. Namely, no linkability between any message sent and any message received and therefore the relationship between sender and recipient is not known. We capture this notion via the *relationship anonymity leakage function* $\mathsf{Leak}_{\mathsf{rel.anon}}$ below [6].

$$\mathsf{Leak}_{\mathsf{rel.anon}}(\mathsf{ptr}, H) := \begin{cases} \big(\mathbf{S}_{\mathsf{ptr}}[H], \mathbf{M}_{\mathsf{ptr}}[H], \mathbf{R}_{\mathsf{ptr}}[H], \mathsf{Act}_{\mathsf{ptr}}[H]\big), & \text{if } \mathsf{End}(\mathsf{ptr}, H) = 1 \\ \mathsf{Act}_{\mathsf{ptr}}[H], & \text{otherwise} \end{cases} \tag{5}$$

Pseudonymity. According to [28] "being pseudonymous is the state of using a pseudonym as ID". To capture pseudonymity, we may slightly abuse definition and consider leakage as a randomized function (or program). Namely, the functionality initially chooses a random permutation π over the set of clients $\mathbf{C}$, and the pseudonym of each client C_ℓ is $\pi(C_\ell) \in [n]$. We denote by $\pi[H]$ the

[6] The definition of $\mathsf{Leak}_{\mathsf{rel.anon}}$ aims to capture the spirit of relationship anonymity in [28], which allows that the message-sender or the message-receiver link may be established, but not the sender-receiver one. We point out that the "message" in this context refers to metadata rather than the actual plaintext. This is made clear in the reference to the classical MIX-net in [28, Footnote 41].

"pseudonymized history" w.r.t. to π, i.e. in every entry of H we replace C_ℓ by $\pi(C_\ell)$. We define the *pseudonymity leakage function* as follows:

$$\mathsf{Leak}_{\mathsf{pseudon}}(\mathsf{ptr}, H) := \pi[H], \quad \text{where } \pi \xleftarrow{\$} \left\{ f \;\middle|\; f : \mathbf{C} \longrightarrow [n] \right\}. \tag{6}$$

Besides anonymity, unlinkability, unobservability and pseudonymity defined in [28], other meaningful notions of privacy can be formally expressed in our framework. We present two such notions below.

Weak anonymity. We define *weak anonymity*, as the privacy notion where the number of messages that a client sends or receives and her fetching activity is leaked. In this weaker notion, the anonymity set for a sender (resp. recipient) consists of the subset of senders (resp. recipients) that are associated with the same number of pending messages. In addition, now the leakage for sender anonymity set is gradually released according to the protocol scheduling, whereas the recipient anonymity set still is leaked "per slot". The *weak anonymity leakage function*, $\mathsf{Leak}_{\mathsf{w.anon}}$, is defined via the sender and recipient multisets as follows:

$$\mathsf{Leak}_{\mathsf{w.anon}}(\mathsf{ptr}, H) := \begin{cases} \big([\![\mathbf{S}_{\mathsf{ptr}}]\!][H], [\![\mathbf{R}_{\mathsf{ptr}}]\!][H], \mathbf{F}_{\mathsf{ptr}}[H], \mathsf{Act}_{\mathsf{ptr}}[H]\big), & \text{if } \mathsf{End}(\mathsf{ptr}, H) = 1 \\ \big([\![\mathbf{S}_{\mathsf{ptr}}]\!][H], \mathbf{F}_{\mathsf{ptr}}[H], \mathsf{Act}_{\mathsf{ptr}}[H]\big), & \text{otherwise} \end{cases} \tag{7}$$

Remark 3. Even though not a very strong privacy notion, weak anonymity supports a reasonable level of privacy for email realizations that aim at a manageable overhead and practical use. Indeed, observe that if we can not tolerate to blow up the ecosystem's complexity by requiring some form of cover traffic (which is a plausible requirement in practical scenarios), then a global adversary monitoring the client-SP channel can easily infer the number of sent/received messages over this channel. Moreover, one may informally argue that in case the email users do not vary significantly in terms of their sending and fetching activity (or at least they can be grouped into large enough sets of similar activity), weak anonymity and standard anonymity are not far. In Section 6, we present an efficient weakly anonymous email construction based on parallel mixing [19, 20].

End-to-end encryption. The standard notion of *end-to-end encryption*, now applied in many internet applications (e.g., Signal, WhatsApp, Viber, Facebook Messenger, Skype), suggests context hiding of M in the communication of the end users (up to the message length $|M|$), in our case the sender and the recipient. Hence, we define the *end-to-end leakage function* $\mathsf{Leak}_{\mathsf{e2e}}$ as shown below.

$$\begin{aligned} \mathsf{Leak}_{\mathsf{e2e}} := \Big(\mathsf{Act}_{\mathsf{ptr}}[H], \Big\{ (C_s@\mathsf{SP}_i, |M|, C_r@\mathsf{SP}_j) \;\Big|\; \exists \mathsf{ptr}' \leq \mathsf{ptr} : \\ \big(\mathsf{ptr}', (\mathsf{sid}, *, \textsc{Send}, \langle C_s@\mathsf{SP}_i, M, C_r@\mathsf{SP}_j \rangle), \text{`}\mathsf{pending}\text{'}\big) \in H \Big\} \Big). \end{aligned} \tag{8}$$

Relation between privacy notions. Observe that the relation between two privacy notions can be deduced via their corresponding leakage functions. Namely,

if for every (ptr, H) a PPT adversary given the output of leakage function $\mathsf{Leak}_1(\mathsf{ptr}, H)$ can derive the output of some other leakage functions $\mathsf{Leak}_2(\mathsf{ptr}, H)$, then $\mathsf{Leak}_2(\cdot,\cdot)$ refers to a stronger notion of privacy than $\mathsf{Leak}_1(\cdot,\cdot)$. In Fig. 3, given the definitions of $\mathsf{Leak}_{\mathsf{unob}}, \mathsf{Leak}_{\mathsf{anon}}, \mathsf{Leak}_{\mathsf{s.unlink}}/\mathsf{Leak}_{\mathsf{r.unlink}}, \mathsf{Leak}_{\mathsf{w.anon}}, \mathsf{Leak}_{\mathsf{e2e}}$ above we relate the respective notions in an intuitively consistent way.

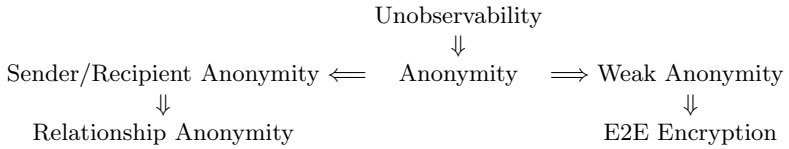

Fig. 3. Relations between privacy notions. By $A \Longrightarrow B$, we denote that notion A is stronger than notion B.

Remark 4. We observe that pseudonymity can not be compared to any of the notions in Fig. 3. Indeed, even for the stronger notion of unobservability, having the set of active addresses is not enough information to derive the pseudonyms. Conversely, having the entire email activity pseudonymized, is not enough information to derive the active clients' real identities. In addition, we can combine pseudonymity with some other privacy notion and result in a new 'pseudonymized' version of the latter (e.g. pseudonymous unobservability/anonymity/etc.). It is easy to see that the new notions can also be expressed via suitable (randomized) leakage functions, by applying a random permutation on the clients' identities and then define leakage as in the original corresponding leakage function, up to this permutation. E.g., for $\pi \xleftarrow{\$} \{f \mid f : \mathbf{C} \longrightarrow \mathbf{C}\}$, "pseudonymized unobservability" could be expressed via the leakage function

$$\mathsf{Leak}_{\mathsf{ps.unob}}(\mathsf{ptr}, H) := \left\{\pi(C_\ell)@\mathsf{SP}_i \mid C_\ell@\mathsf{SP}_i \in \mathsf{Act}_{\mathsf{ptr}}[H]\right\} .$$

Remark 5. As our E2E leakage does not cover fetch information, strictly speaking the implication from Weak anonymity to E2E encryption only holds if the fetch behavior is either known in advance (e.g. because of the system specification) or irrelevant. One could also opt to add the additional leakage to the E2E definition, but we believe there is little practical value in doing so.

5 An email ecosystem with optimal privacy

We present an email ecosystem, denoted by $\mathbb{E}_{\mathsf{comp}}$, that achieves privacy at an optimal level at the cost of high (quadratic) communication complexity. Specifically, in each time slot all SPs in $\mathbb{E}_{\mathsf{comp}}$ communicate with *complete connectivity* and always pad the right amount of dummy traffic, so that the activity of their registered clients is unobservable by a third party, leaking

nothing more than that they are online (logged in). In addition, end-to-end communication between the clients is done via encryption layers by utilizing a public key encryption scheme $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. The encryption layers are structured according to the network route

$$\textit{Sender} \longrightarrow \textit{Sender's SP} \longrightarrow \textit{Receiver's SP} \longrightarrow \textit{Receiver}$$

To support unobservability, the online clients who do not send an actual message during some round provide their SPs with a dummy ciphertext.

Even though certainly impractical, $\mathbb{E}_{\mathsf{comp}}$ sets a "golden standard" of privacy according to the discussion in Remark 2 that efficient constructions refer to in order to balance the privacy vs. efficiency trade off.

Description of $\mathbb{E}_{\mathsf{comp}}$. The email ecosystem $\mathbb{E}_{\mathsf{comp}}$ operates under a known delay bound Δ_{net}. Throughout the description of $\mathbb{E}_{\mathsf{comp}}$, we assume that the following simplifications: (a) all ciphertexts are of the same length. By $\mathsf{Enc}_{[P]}(M)$, we denote the encryption of M under P's public key, and (b) all computations require one time slot[7]:

The phases of $\mathbb{E}_{\mathsf{comp}}$ are as follows:

■ **Initialization**:

- On input $(\mathsf{sid}, \textsc{Init})$, a service provider SP_i that is not yet initialized, runs $\mathsf{KeyGen}(1^\lambda)$ to generate a private and a public key pair $(\mathsf{sk}_{\mathsf{SP}_i}, \mathsf{pk}_{\mathsf{SP}_i})$. Then, it initializes its list of setup entities, denoted by $L_{\mathsf{setup}}^{\mathsf{SP}_i}$, as the pair $(\mathsf{pk}_{\mathsf{SP}_i}, \mathsf{SP}_i)$, implying that at first SP_i is only aware of itself. In addition, SP_i initializes its list of valid addresses, denoted by $\mathbf{Ad}_{\mathsf{SP}_i}$, as empty. Finally, it broadcasts the message $(\mathsf{sid}, \textsc{Channel}, (\mathsf{setup}, \mathsf{pk}_{\mathsf{SP}_i}), \mathsf{SP}_j)$ to $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$ for every $j \in [N] \backslash \{i\}$, so that all other SPs receive its public key.
- Upon receiving $(\mathsf{sid}, (\mathsf{setup}, \mathsf{pk}_{\mathsf{SP}_j}, \mathsf{SP}_j)$ from $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$, SP_i adds $\big((\mathsf{pk}_{\mathsf{SP}_j}, \mathsf{SP}_j)\big)$ to $L_{\mathsf{setup}}^{\mathsf{SP}_i}$. When $L_{\mathsf{setup}}^{\mathsf{SP}_i}$ contains all SPs, the SP_i sets its status to '$\mathsf{execute}$', and only then it processes messages of the **Execution** phase described below.

■ **Execution**:

Registration:

- On input $(\mathsf{sid}, \textsc{Register}, @\mathsf{SP}_i)$, if C_ℓ is not registered to any SP and has not yet sent a message $(\mathsf{sid}, \textsc{Advance\_Clock}, C_\ell)$, then:

  1. C_ℓ sends the message $(\mathsf{sid}, \textsc{Advance\_Clock}, C_\ell)$ to $\mathcal{G}_{\mathsf{clock}}$.
  2. Upon receiving $(\mathsf{sid}, \textsc{Advance\_Ack}, C_\ell)$ from $\mathcal{G}_{\mathsf{clock}}$, C_ℓ runs $\mathsf{KeyGen}(1^\lambda)$ to generate a private and a public key pair $(\mathsf{sk}_\ell, \mathsf{pk}_\ell)$. It also initializes her list of setup entities, $L_{\mathsf{setup}}^{\ell}$ as the pair $(\mathsf{pk}_\ell, C_\ell)$, and her list of valid addresses, $\mathbf{Ad}_\ell$

[7] As it will become clear by the ecosystem's description, the above simplifications do not harm generality essentially. Namely,(a) can be reached via padding, while (b) leads to similar analysis as requiring a computational time upper bound.

as empty. Then, she sends the message $\big(\mathsf{sid}, \textsc{Channel}, (\mathsf{register}, \mathsf{pk}_\ell), \mathsf{SP}_i\big)$ to $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$.

3. Upon receiving $\big(\mathsf{sid}, (\mathsf{register}, \mathsf{pk}_\ell), C_\ell\big)$ from $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$, SP_i checks that $(C_\ell, \cdot) \notin L_{\mathsf{setup}}^{\mathsf{SP}_i}$ and that pk_ℓ is a valid public key, and if so, then it adds $(\mathsf{pk}_\ell, C_\ell)$ to $L_{\mathsf{setup}}^{\mathsf{SP}_i}$ and $C_\ell@\mathsf{SP}_i$ to $\mathbf{Ad}_{\mathsf{SP}_i}$. Next, it updates other SPs and its registered clients by broadcasting the message $\big(\mathsf{sid}, \textsc{Channel}, (\mathsf{setup}, \mathsf{pk}_{C_\ell}, C_\ell), P\big)$ to $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$ for every $P \in (\mathbf{SP} \setminus \{\mathsf{SP}_i\}) \cup \mathbf{C}_i$. It also sends the message $\big(\mathsf{sid}, \textsc{Channel}, (\mathsf{setup}, \{\mathsf{pk}_P, P\}_{P \in L_{\mathsf{setup}}^{\mathsf{SP}_i}}, \mathbf{Ad}_{\mathsf{SP}_i}), C_\ell\big)$, updating C_ℓ with all the valid public keys and addresses it knows so far. Finally, it initializes the inbox $\mathsf{Inbox}[C_\ell@\mathsf{SP}_i]$ of C_ℓ.

4. Upon receiving $\big(\mathsf{sid}, (\mathsf{setup}, \mathsf{pk}_\ell, C_\ell), \mathsf{SP}_i\big)$ from $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$, SP_j checks that $(C_\ell, \cdot) \notin L_{\mathsf{setup}}^{\mathsf{SP}_j}$ and that pk_ℓ is a valid public key, and if so, then it, then it adds $(\mathsf{pk}_\ell, C_\ell)$ to $L_{\mathsf{setup}}^{\mathsf{SP}_j}$ and $C_\ell@\mathsf{SP}_i$ to $\mathbf{Ad}_{\mathsf{SP}_j}$. It also adds it adds C_ℓ to its set of active users, denoted by $L_{\mathsf{act}}^{\mathsf{SP}_i}$ and initialized as empty. Next, it updates its registered clients by broadcasting the message $\big(\mathsf{sid}, \textsc{Channel}, (\mathsf{setup}, \mathsf{pk}_{C_\ell}, C_\ell), C\big)$ to $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$ for every $C \in \mathbf{C}_j$.

5. Upon receiving $\big(\mathsf{sid}, (\mathsf{setup}, \{\mathsf{pk}_P, P\}_{P \in L_{\mathsf{setup}}^{\mathsf{SP}_i}}, \mathbf{Ad}_{\mathsf{SP}_i}), \mathsf{SP}_i\big)$ from $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$, the client C_ℓ, newly registered to SP_i, checks that all public keys are valid. If the check is successful, then C_ℓ adds $\{\mathsf{pk}_P, P\}_{P \in L_{\mathsf{setup}}^{\mathsf{SP}_i}}$ to L_{setup}^{i} and sets $\mathbf{Ad}_\ell \leftarrow \mathbf{Ad}_{\mathsf{SP}_i}$. Thus, from this point, C_ℓ is aware of the public information of all SPs and all registered clients up to now. In addition, it sets its status as logged in to SP_i.

6. Upon receiving $\big(\mathsf{sid}, (\mathsf{setup}, \mathsf{pk}_{C_t}, \mathsf{SP}_j), \mathsf{SP}_i\big)$ from $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$, client C_ℓ (now already registered to SP_i) checks the validity of pk_{C_t}, and if so, then she adds (pk_t, C_t) to $L_{\mathsf{setup}}^{\ell}$ and $C_t@\mathsf{SP}_j$ to $\mathbf{Ad}_\ell$.

Log in:

– On input $\big(\mathsf{sid}, \textsc{Active}, @\mathsf{SP}_i\big)$, if C_ℓ is not logged in, $C_\ell@\mathsf{SP}_i$ is her valid address, and has not yet sent a message $(\mathsf{sid}, \textsc{Advance\_Clock}, C_\ell)$, then:

1. C_ℓ sends the message $(\mathsf{sid}, \textsc{Advance\_Clock}, C_\ell)$ to $\mathcal{G}_{\mathsf{clock}}$.

2. Upon receiving $(\mathsf{sid}, \textsc{Advance\_Ack}, C_\ell)$ from $\mathcal{G}_{\mathsf{clock}}$, C_ℓ "logs in" by sending $\big(\mathsf{sid}, \textsc{Channel}, \mathsf{Enc}_{[\mathsf{SP}_i]}(\textsc{Active}), \mathsf{SP}_i\big)$ to $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$.

3. Upon receiving $\big(\mathsf{sid}, \mathsf{Enc}_{[\mathsf{SP}_i]}(\textsc{Active}), C_\ell\big)$ from $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$, SP_i decrypts as $\big(\mathsf{sid}, \textsc{Active}, C_\ell\big)$ and checks that $C_\ell@\mathsf{SP}_i \in \mathbf{Ad}_{\mathsf{SP}_i}$. If so, then it adds C_ℓ to $L_{\mathsf{act}}^{\mathsf{SP}_i}$.

Log out:

– On input $\big(\mathsf{sid}, \textsc{Inactive}, @\mathsf{SP}_i\big)$, if C_ℓ is logged in, $C_\ell@\mathsf{SP}_i$ is her valid address, and has not yet sent a message $(\mathsf{sid}, \textsc{Advance\_Clock}, C_\ell)$, then:

1. C_ℓ sends the message $(\mathsf{sid}, \textsc{Advance\_Clock}, C_\ell)$ to $\mathcal{G}_{\mathsf{clock}}$.

2. Upon receiving $(\mathsf{sid}, \textsc{Advance\_Ack}, C_\ell)$ from $\mathcal{G}_{\mathsf{clock}}$, C_ℓ "logs out" by sending $\big(\mathsf{sid}, \textsc{Channel}, \mathsf{Enc}_{[\mathsf{SP}_i]}(\textsc{Inactive}), \mathsf{SP}_i\big)$ to $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$.

3. Upon receiving $\big(\mathsf{sid}, \mathsf{Enc}_{[\mathsf{SP}_i]}(\textsc{Inactive}), C_\ell\big)$ from $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$, SP_i decrypts as $\big(\mathsf{sid}, \textsc{Inactive}, C_\ell\big)$ and checks that $C_\ell@\mathsf{SP}_i \in \mathbf{Ad}_{\mathsf{SP}_i}$. If so, then it removes C_ℓ from $L_{\mathsf{act}}^{\mathsf{SP}_i}$.

Send:

– On input $\big(\mathsf{sid}, \textsc{Send}, \langle C_s@\mathsf{SP}_i, M, C_r@\mathsf{SP}_j\rangle\big)$, if C_s is logged in to SP_i and has not yet sent a message $(\mathsf{sid}, \textsc{Advance\_Clock}, C_s)$, then:

1. C_s sends the message $(\mathsf{sid}, \textsc{Advance\_Clock}, C_s)$ to $\mathcal{G}_{\mathsf{clock}}$.

2. Upon receiving $(\mathsf{sid}, \textsc{Advance\_Ack}, C_s)$ from $\mathcal{G}_{\mathsf{clock}}$, C_s encrypts the message M into layers and provides $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$ with the layered encryption

$$\big(\mathsf{sid}, \textsc{Channel}, \mathsf{Enc}_{[\mathsf{SP}_i]}\big(\mathsf{Enc}_{[\mathsf{SP}_j]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big)\big), \mathsf{SP}_i\big)$$

3. Upon receiving $\big(\mathsf{sid}, \mathsf{Enc}_{[\mathsf{SP}_i]}\big(\mathsf{Enc}_{[\mathsf{SP}_j]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big)\big), C_s\big)$ from $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$, SP_i checks that $C_s@\mathsf{SP}_i \in \mathbf{Ad}_{\mathsf{SP}_i}$. If so, then it decrypts the first layer with $\mathsf{sk}_{\mathsf{SP}_i}$ and adds $\big(\mathsf{sid}, C_s@\mathsf{SP}_i, \mathsf{Enc}_{[\mathsf{SP}_j]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big)\big)$ to its set of messages pending to be sent, denoted by $L_{\mathsf{send}}^{\mathsf{SP}_i}$ and initialized as empty.

Fetch:

– On input $\big(\mathsf{sid}, \textsc{Fetch}, C_r@\mathsf{SP}_j\big)$, if C_r is logged in to SP_j and has not yet sent a message $(\mathsf{sid}, \textsc{Advance\_Clock}, C_r)$:

1. C_r sends the message $(\mathsf{sid}, \textsc{Advance\_Clock}, C_\ell)$ to $\mathcal{G}_{\mathsf{clock}}$.

2. Upon receiving $(\mathsf{sid}, \textsc{Advance\_Ack}, C_r)$ from $\mathcal{G}_{\mathsf{clock}}$, C_r sends the message $\big(\mathsf{sid}, \textsc{Channel}, \mathsf{Enc}_{[\mathsf{SP}_j]}(\textsc{Fetch}), \mathsf{SP}_j\big)$ to $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$.

3. Upon receiving $\big(\mathsf{sid}, \mathsf{Enc}_{[\mathsf{SP}_j]}(\textsc{Fetch}), C_r\big)$ from $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$, C_r checks that $C_r@\mathsf{SP}_j \in \mathbf{Ad}_{\mathsf{SP}_j}$. If so, then she decrypts and adds $\mathsf{Inbox}[C_r@\mathsf{SP}_j]$ to her set of inboxes which messages are pending to be pushed, denoted by $L_{\mathsf{push}}^{\mathsf{SP}_j}$.

4. Upon receiving $(\mathsf{sid}, E_{r,1}, \ldots, E_{r,n}, \mathsf{SP}_j)$ from $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$ (see below), if C_r is registered to SP_j and has sent a $\big(\mathsf{sid}, \textsc{Fetch}, C_r@\mathsf{SP}_j\big)$ request, then she decrypts all ciphertexts and stores the ones that are not dummy, i.e. they correspond to actual mail messages with her as recipient. Otherwise, she discards $(\mathsf{sid}, E_{r,1}, \ldots, E_{r,n}, \mathsf{SP}_j)$.

Clock reading:

– On input $\big(\mathsf{sid}, \textsc{Read\_Clock}\big)$, the entity $P \in \mathbf{C} \cup \mathbf{SP}$ sends the message $\big(\mathsf{sid}, \textsc{Read\_Clock}\big)$ to $\mathcal{G}_{\mathsf{clock}}$. Upon receiving $\big(\mathsf{sid}, \textsc{Read\_Clock}, \mathsf{Cl}\big)$ from $\mathcal{G}_{\mathsf{clock}}$, P stores Cl as its local time.

Clock advance (for clients):

– On input $\big(\mathsf{sid}, \textsc{Advance\_Clock}\big)$, if the client C_ℓ is logged in to SP_i and has not yet sent a message $(\mathsf{sid}, \textsc{Advance\_Clock}, \mathsf{SP}_i)$, then she executes the following steps:

1. C_ℓ sends the message $(\mathsf{sid}, \textsc{Advance\_Clock}, C_\ell)$ to $\mathcal{G}_{\mathsf{clock}}$.

2. Upon receiving $(\mathsf{sid}, \textsc{Advance\_Ack}, C_\ell)$ from $\mathcal{G}_{\mathsf{clock}}$, then she sends a dummy message $\big(\mathsf{sid}, \mathsf{Enc}_{[\mathsf{SP}_i]}(\mathsf{null})\big)$ to SP_i via $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$ (in turn, SP_i will discard the received null upon decryption).

Clock advance (for SPs):

– On input $\big(\mathsf{sid}, \textsc{Advance\_Clock}\big)$, if SP_i has not yet sent a message $(\mathsf{sid}, \textsc{Advance\_Clock}, \mathsf{SP}_i)$, then it executes the following steps:

1. SP_i sends the message $(\mathsf{sid}, \textsc{Advance\_Clock}, \mathsf{SP}_i)$ to $\mathcal{G}_{\mathsf{clock}}$.

2. Upon receiving $(\mathsf{sid}, \textsc{Advance\_Ack}, \mathsf{SP}_i)$ from $\mathcal{G}_{\mathsf{clock}}$, for every address $C_s@\mathsf{SP}_i \in \mathbf{Ad}_{\mathsf{SP}_i}$:

- If there is a message $\big(\mathsf{sid}, C_s@\mathsf{SP}_i, \mathsf{Enc}_{[\mathsf{SP}_j]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big)\big)$ in $L^{\mathsf{SP}_i}_{\mathsf{send}}$, then SP_i broadcasts $\big(\mathsf{sid}, \mathsf{Enc}_{[\mathsf{SP}_j]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big)\big)$ to all SPs via $\mathcal{F}^{\Delta_{\mathsf{net}}}_{\mathsf{auth}}(\mathbf{P})$, and removes the message from $L^{\mathsf{SP}_i}_{\mathsf{send}}$.
- If there is no such message for $C_s@\mathsf{SP}_i$ but $C_s \in L^i_{\mathsf{act}}$, then SP_i broadcasts a dummy message $\big(\mathsf{sid}, \mathsf{Enc}_{[\mathsf{SP}_i]}\big(\mathsf{null}\big)\big)$ under its own key.

3. Upon receiving a message $\big(\mathsf{sid}, \tilde{E}, \mathsf{SP}_i\big)$ from $\mathcal{F}^{\Delta_{\mathsf{net}}}_{\mathsf{auth}}(\mathbf{P})$, SP_j checks whether $\tilde{E}$ is a ciphertext under its public key that decrypts as a pair of a valid address $C_r@\mathsf{SP}_j$ along with an (encrypted) message E. If so, then it adds E to $\mathsf{Inbox}[C_r@\mathsf{SP}_j]$.

4. When $L^{\mathsf{SP}_j}_{\mathsf{fin,k}}$ contains all SPs, then for every address $C_r@\mathsf{SP}_j$:

- If $\mathsf{Inbox}[C_r@\mathsf{SP}_j] \in L^{\mathsf{SP}_j}_{\mathsf{push}}$, then SP_j forwards all messages $E_{r,1}, \ldots E_{r,n_r}$ in $\mathsf{Inbox}[C_r@\mathsf{SP}_j]$ to C_r along with $n - n_r$ dummy ciphertexts under C_r's public key, empties $\mathsf{Inbox}[C_r@\mathsf{SP}_j]$ and removes it from $L^{\mathsf{SP}_j}_{\mathsf{push}}$.
- If $\mathsf{Inbox}[C_r@\mathsf{SP}_j] \notin L^{\mathsf{SP}_j}_{\mathsf{push}}$ but $C_r \in L^{\mathsf{SP}_j}_{\mathsf{act}}$, then SP_j forwards n dummy encryptions of 'null' to C_r, under her public key.

Thus, in any case, if C_r is active, then SP_j sends a message of the form $(\mathsf{sid}, E_{r,1}, \ldots, E_{r,n})$ to C_r via $\mathcal{F}^{\Delta_{\mathsf{net}}}_{\mathsf{auth}}(\mathbf{P})$.

Privacy of $\mathbb{E}_{\mathsf{comp}}$. To prove the privacy of $\mathbb{E}_{\mathsf{comp}}$, we require that the underlying public key encryption scheme $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies m-IND-CPA, as specified in Section 2.2. In the following theorem, we prove that $\mathbb{E}_{\mathsf{comp}}$ only leaks the "activity bit" of the clients formally expressed by the leakage function $\mathsf{Leak}_{\mathsf{unob}}(\cdot,\cdot)$ defined in Eq. (1).

Theorem 1. *Let $\mathbb{E}_{\mathsf{comp}}$ with clients $\mathbf{C} = \{C_1, \ldots, C_n\}$ and service providers $\mathbf{SP} = \mathsf{SP}_1, \ldots, \mathsf{SP}_N$ be implemented over the PKE scheme $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ that achieves m-IND-CPA security with error $\epsilon(\lambda)$. Then, $\mathbb{E}^{\mathcal{G}_{\mathsf{clock}}, \mathcal{F}^{\Delta_{\mathsf{net}}}_{\mathsf{auth}}}_{\mathsf{comp}}$ achieves computational $2(n+N)\epsilon(\lambda)$-privacy for message delay Δ_{net} with respect to the unobservability leakage function defined below*

$$\mathsf{Leak}_{\mathsf{unob}}(\mathsf{ptr}, H) := \mathsf{Act}_{\mathsf{ptr}}[H] \ .$$

Proof. Let $\mathcal{A}$ be a global passive PPT adversary against $\mathbb{E}^{\mathcal{G}_{\mathsf{clock}}, \mathcal{F}^{\Delta_{\mathsf{net}}}_{\mathsf{auth}}}_{\mathsf{comp}}$. We begin by constructing a simulator Sim for $\mathcal{A}$ as shown below.

Constructing a simulator for $\mathcal{A}$. The ideal adversary Sim for $\mathcal{A}$ that for any environment $\mathcal{Z}$, simulates an execution of $\mathbb{E}_{\mathsf{comp}}$ as follows:
Simulating interaction between $\mathcal{Z}$ and $\mathcal{A}$.

- Upon receiving a message $\big(\mathsf{sid}, M\big)$ from $\mathcal{Z}$, it forwards $\big(\mathsf{sid}, M\big)$ to $\mathcal{A}$ playing the role of a simulated environment.
- Upon receiving a message $\big(\mathsf{sid}, M\big)$ from $\mathcal{A}$ intended for the environment, it forwards $\big(\mathsf{sid}, M\big)$ to $\mathcal{Z}$.

Achieving synchronicity.

– Upon receiving any message from $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}_{\mathsf{unob}},\Delta_{\mathsf{net}}}(\mathbf{P})$, Sim sends the message $(\mathsf{sid}, \textsc{Read\_Clock})$ to $\mathcal{G}_{\mathsf{clock}}$. Upon receiving $(\mathsf{sid}, \textsc{Read\_Clock}, \mathsf{Cl})$ from $\mathcal{G}_{\mathsf{clock}}$, it stores Cl as the global time of the real-world simulation. This way, Sim simulates an execution where the simulated entities are synchronized with respective actual ones in the ideal-world.

Simulating real-world message delivery.

– Upon receiving a leakage message of the form $(\mathsf{sid}, (\mathsf{ptr}, M))$ (possibly $M = \perp$) from $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}_{\mathsf{unob}},\Delta_{\mathsf{net}}}(\mathbf{P})$, Sim knows that this message refers to some command (register/active/inactive/send/fetch) that in the real-world protocol is realized via communication between a client and her SP. Since in the simulation Sim also plays the role of $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$ in the eyes of $\mathcal{A}$, it must be consistent with the bounded delays (up to Δ_{net}) that $\mathcal{A}$ imposes on message communication. To achieve this consistency, Sim keeps record of the simulated message $\tilde{M}$ that sends to the simulated $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$ and is associated with ptr. Whenever the message delivery of $\tilde{M}$ is allowed, either by $\mathcal{A}$ or automatically when Δ_{net} delay has passed, Sim sends the message $(\mathsf{sid}, \textsc{Allow\_Exec}, \mathsf{ptr})$ to $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}_{\mathsf{unob}},\Delta_{\mathsf{net}}}(\mathbf{P})$.

Simulating Initialization.

– Upon receiving $(\mathsf{sid}, \textsc{Init}, \mathsf{SP}_i)$ from $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}_{\mathsf{unob}},\Delta_{\mathsf{net}}}(\mathbf{P})$, it runs $\mathsf{Gen}(1^\lambda)$ on behalf of SP_i to generate a pair of a private and a public key pair $(\mathsf{sk}_{\mathsf{SP}_i}, \mathsf{pk}_{\mathsf{SP}_i})$. Then, it broadcasts the message $(\mathsf{sid}, \textsc{Channel}, (\mathsf{setup}, \mathsf{pk}_{\mathsf{SP}_i}), \mathsf{SP}_j)$ to every $j \in [N] \setminus \{i\}$, also simulating the role of $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$. Observe that since $\mathcal{A}$ is global and passive, the execution will always initiate upon $\mathcal{Z}$'s request. Then, Sim sends the message $(\mathsf{sid}, \textsc{Allow\_Init}, \mathsf{SP}_i)$ to $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}_{\mathsf{unob}},\Delta_{\mathsf{net}}}(\mathbf{P})$.
– Upon receiving $(\mathsf{sid}, \mathsf{ready})$ from $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}_{\mathsf{unob}},\Delta_{\mathsf{net}}}(\mathbf{P})$, if all simulated SPs have initialized by generating and broadcasting their keys, then it sends $(\mathsf{sid}, \textsc{Execute})$ to $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}_{\mathsf{unob}},\Delta_{\mathsf{net}}}(\mathbf{P})$. Otherwise, it aborts simulation.

Simulating Execution.

Whenever the environment sends a register/active/inactive/send/fetch/clock advance command to a dummy party P that forwards it to $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}_{\mathsf{unob}},\Delta_{\mathsf{net}}}(\mathbf{P})$, Sim obtains (i) an $(\mathsf{sid}, \textsc{Advance\_Clock}, P)$ notification from $\mathcal{G}_{\mathsf{clock}}$, and (ii) the leakage of the form $(\mathsf{sid}, \mathsf{ptr}, \mathsf{Act}_{\mathsf{ptr}}[H])$ from $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}_{\mathsf{unob}},\Delta_{\mathsf{net}}}(\mathbf{P})$. Namely, Sim obtains the sequence of clock advances and the transcript of activations/deactivations. We describe how using this information, Sim simulates execution:

– Upon receiving $(\mathsf{sid}, \textsc{Advance\_Clock}, C_\ell)$ and $(\mathsf{sid}, \mathsf{ptr}, \mathsf{Act}_{\mathsf{ptr}}[H])$, then:
  • Playing the role of the global clock, Sim sends a simulated notification $(\mathsf{sid}, \textsc{Advance\_Clock}, C_\ell)$ to $\mathcal{A}$.

- If $C_\ell@\mathsf{SP}_i$ is in $\mathsf{Act}_{\mathsf{ptr}}[H]$ and $\big(\mathsf{sid}, \mathsf{ptr}, \mathsf{Act}_{\mathsf{ptr}}[H]\big)$ is the first entry that $C_\ell@\mathsf{SP}_i$ is activated, then Sim deduces that this refers to a registration command (Recall that for simplicity we included the pending registration commands in the set of active addresses). In this case, Sim runs the registration protocol between C_ℓ and SP_i exactly as in the description of $\mathbb{E}_{\mathsf{comp}}$, except that it replaces the ciphertext contents with 'null' messages. When $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}$ delivers the message, Sim sends the message $\big(\mathsf{sid}, \textsc{Allow\_Exec}, \mathsf{ptr}\big)$ to $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}_{\mathsf{unob}}, \Delta_{\mathsf{net}}}(\mathbf{P})$.
- If $C_\ell@\mathsf{SP}_i$ is in $\mathsf{Act}_{\mathsf{ptr}}[H]$ and is registered but not yet logged in, then Sim deduces that this refers to an active or a clock advance command. In either of these cases, Sim simulates execution by sending a dummy ciphertext $\big(\mathsf{sid}, \textsc{Channel}, \mathsf{Enc}_{[\mathsf{SP}_i]}(\mathsf{null}), \mathsf{SP}_i\big)$ to the simulated $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}$. When $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}$ delivers the message, Sim sends the message $\big(\mathsf{sid}, \textsc{Allow\_Exec}, \mathsf{ptr}\big)$ to $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}_{\mathsf{unob}}, \Delta_{\mathsf{net}}}(\mathbf{P})$.
- If $C_\ell@\mathsf{SP}_i$ is in $\mathsf{Act}_{\mathsf{ptr}}[H]$ and is registered and already logged in, then Sim deduces that this refers to either a inactive, send, fetch or a clock advance command. In either of these cases, Sim simulates execution by sending a dummy ciphertext $\big(\mathsf{sid}, \textsc{Channel}, \mathsf{Enc}_{[\mathsf{SP}_i]}(\mathsf{null}), \mathsf{SP}_i\big)$ as above.
- If $C_\ell@\mathsf{SP}_i$ is not in $\mathsf{Act}_{\mathsf{ptr}}[H]$, then Sim deduces that $C_\ell@\mathsf{SP}_i$ is inactive and takes no further action.

– Upon receiving $(\mathsf{sid}, \textsc{Advance\_Clock}, \mathsf{SP}_i)$ and $\big(\mathsf{sid}, \mathsf{ptr}, \mathsf{Act}_{\mathsf{ptr}}[H]\big)$:
- Playing the role of the global clock, Sim sends a simulated notification $(\mathsf{sid}, \textsc{Advance\_Clock}, \mathsf{SP}_i)$ to $\mathcal{A}$.
- For every address $C_s@\mathsf{SP}_i \in \mathbf{Ad}_{\mathsf{SP}_i}$, it broadcasts a dummy message $\big(\mathsf{sid}, \textsc{Channel}, \mathsf{Enc}_{[\mathsf{SP}_i]}(\mathsf{null}), \mathsf{SP}_i\big)$ to all other SPs. Then, it sends the message $\big(\mathsf{sid}, \textsc{Allow\_Exec}, \mathsf{ptr}\big)$ to $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}_{\mathsf{unob}}, \Delta_{\mathsf{net}}}(\mathbf{P})$.

Reducing privacy to m-IND-CPA security. We prove the privacy of $\mathbb{E}_{\mathsf{comp}}^{\mathcal{G}_{\mathsf{clock}}, \mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}}$ via a reduction to the m-IND-CPA security with error ϵ of the underlying public key encryption scheme $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, which is assumed in the theorem's statement. Our reduction works as follows: Let $\mathcal{A}$ be a real-world adversary and $\mathcal{Z}$ be an environment. First, we order the clients and servers as parties $P_1, \ldots, P_{n+N}$. Then, we construct a sequence of "hybrid" m-IND-CPA adversaries $\mathcal{B}_1, \ldots, \mathcal{B}_{n+N}$, where $\mathcal{B}_{j^*}$ executes the following steps:

1. It receives a public key pk from the m-IND-CPA challenger.
2. It generates the parties $P_1, \ldots, P_{n+N}$ and simulates an execution of $\mathbb{E}_{\mathsf{comp}}^{\mathcal{G}_{\mathsf{clock}}, \mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}}$ conducted by $\mathcal{Z}$ and under the presence of $\mathcal{A}$, also playing the role of $\mathcal{G}_{\mathsf{clock}}, \mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}$. The simulation differs from an actual execution as shown below:
   (a) Upon initialization of a party P_j: if $P_j \neq P_{j^*}$, then $\mathcal{B}_{j^*}$ honestly generates a fresh key pair $(\mathsf{sk}_j, \mathsf{pk}_j)$. If $P_j = P_{j^*}$, then it sets $\mathsf{pk}_{j^*} := \mathsf{pk}$.
   (b) When a party P_i must send an encrypted message M under the public key of P_j (note it may be the case that $P_i = P_j$) via $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}$:
      – If $j < j^*$, then $\mathcal{B}_{j^*}$ sends an encryption of M under pk_j.

- If $j = j^*$, then it sends a challenge pair $(M_0, M_1) := (\mathsf{null}, M)$ to the m-IND-CPA challenger. Upon receiving a ciphertext $\mathsf{Enc}_{[P_{j^*}]}(M_b)$, where b is the m-IND-CPA challenge bit, it sends $\mathsf{Enc}_{[P_{j^*}]}(M_b)$ to P_{j^*}.
- If $j > j^*$, then it sends an encryption of null under pk_j.

(c) Since $\mathcal{A}$ is passive, all parties are honest, thus $\mathcal{B}_{j^*}$ is completely aware of the plaintext-ciphertext correspondence. Therefore, when P_i encrypts M under P_j's public key to a ciphertext $\mathsf{Enc}_{[P_j]}(M)$, $\mathcal{B}_{j^*}$ proceeds as if P_j had indeed decrypted this ciphertext to M.

3. It returns the output of $\mathcal{Z}$.

Given the description of $\mathcal{B}_{j^*}$, $j^* = 1, \ldots, n+N$, we make the following observations:

- *The limit case* $j^* = 1$: if $b = 0$, then $\mathcal{B}_1$ replaces all real-world communication with encryptions of 'null', exactly as Sim does in its simulation. Thus, we have that
$$\Pr\left[\mathcal{B}_1 = 1 \mid b = 0\right] = \mathrm{EXEC}^{\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}_{\mathsf{unob}}, \Delta_{\mathsf{net}}}}_{\mathsf{Sim}, \mathcal{Z}, \mathcal{G}_{\mathsf{clock}}}[\mathbf{P}](\lambda)\ . \tag{9}$$
- *The hybrid step*: for every $1 \leq j^* < n+N$, the adversaries $\mathcal{B}_{j^*}$ and $\mathcal{B}_{j^*+1}$ have the same behavior regarding the parties P_j, where $j \neq j^*, j^*+1$. In addition, if the m-IND-CPA challenge bit b is 1, then $\mathcal{B}_{j^*}$ (i) respects the encryptions of P_j^* (hence, of every P_j, for $j \leq j^*$) and (ii) replaces with null any plaintext intended for P_j, for $j \geq j^*+1$. Observe that this is exactly the behavior of $\mathcal{B}_{j^*+1}$, if $b = 0$. Therefore, it holds that
$$\Pr\left[\mathcal{B}_{j^*} = 1 \mid b = 1\right] = \Pr\left[\mathcal{B}_{j^*+1} = 1 \mid b = 0\right]\ . \tag{10}$$
- *The limit case* $j^* = n+N$: if $b = 1$, then $\mathcal{B}_{n+N}$ executes real-world communication respecting the environments' instructions and inputs. Thus, we have that
$$\Pr\left[\mathcal{B}_{n+N} = 1 \mid b = 1\right] = \mathrm{EXEC}^{\mathbb{E}^{\mathcal{G}_{\mathsf{clock}}}, \mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}}_{\mathcal{A}, \mathcal{Z}, \mathcal{G}_{\mathsf{clock}}}[\mathbf{P}](\lambda)\ . \tag{11}$$

Consequently, by Eq. (9) and the m-IND-CPA security of PKE, we have that for every $j^* \in [n+N]$, it holds that

$$\begin{aligned}
&\Big|\Pr\left[\mathcal{B}_{j^*} = 1 \mid b = 1\right] - \Pr\left[\mathcal{B}_{j^*} = 1 \mid b = 0\right]\Big| = \\
=&\Big|\Pr\left[\mathcal{B}_{j^*} = 1 \mid b = 1\right] - \big(1 - \Pr\left[\mathcal{B}_{j^*} = 0 \mid b = 0\right]\big)\Big| \leq \\
\leq&\Big|2 \cdot \Pr\left[(\mathcal{B}_{j^*} = 1) \wedge (b = 1)\right] + 2 \cdot \Pr\left[(\mathcal{B}_{j^*} = 0) \wedge (b = 0)\right] - 1\Big| = \\
=&\Big|2 \cdot \Pr\left[\mathcal{B}_{j^*}(1^\lambda) \text{ breaks } \mathsf{PKE}\right] - 1\Big| \leq \Big|2 \cdot \big(1/2 + \epsilon(\lambda)\big) - 1\Big| = 2\epsilon(\lambda)\ .
\end{aligned} \tag{12}$$

Finally, by Eq. (9), (10),(11), and (12), we get that

$$\Big|\mathrm{EXEC}^{\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}_{\mathsf{unob}}, \Delta_{\mathsf{net}}}}_{\mathsf{Sim}, \mathcal{Z}, \mathcal{G}_{\mathsf{clock}}}[\mathbf{P}](\lambda) - \mathrm{EXEC}^{\mathbb{E}^{\mathcal{G}_{\mathsf{clock}}}, \mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}}_{\mathcal{A}, \mathcal{Z}, \mathcal{G}_{\mathsf{clock}}}[\mathbf{P}](\lambda)\Big| \leq 2(n+N)\epsilon(\lambda)$$

which completes the proof. □

6 A parallel mix email ecosystem with t strata

We will now describe a design to be used for routing messages between various clients, based on parallel mixing [19, 20]. A parallel mix is a design that borrows characteristics from stratified mixes i.e mixes where servers are grouped in sets called *strata*, and routing is restricted so that each stratum except the first only receives messages from the previous one and each stratum except the last only forwards messages to the next (the first and last strata operate as the entry and exit points respectively). In parallel mixing routing is determined by the servers themselves in the interest of symmetry and predictability in performance and security. All t strata consist of σ nodes each. We use $\mathsf{MX}_{i,j}$ to indicate the j-th server in stratum i, and let $\mathbf{MX} = \{\mathsf{MX}_{i,j} | i \leq t, j \leq \sigma\}$. We use $\mathbf{P} = \big(\mathbf{C} \cup \mathbf{SP} \cup \mathbf{MX}\big)$ to denote the set of all involved parties. We use a set of assumptions similar to those of section 5, specifically: (a) all communication is executed via $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{P})$ as described in Fig. 2; (b) all messages have the same size (i.e messages are padded ahead of time); (c) all computations complete within one unit slot; (d) each client is assigned to exactly one address.

As we assume a passive adversary and no corruptions, we are able to use a simple layering of encryptions instead of a more complex onion scheme. In practice one may wish to use a scheme such as Sphinx [15] or a variant thereof.

■ **Initialization**: Nodes of the same stratum share stratum-specific keying material. In practice, because of the long structure of the mixnet, and the large number of nodes involved, we might have that the same entities will be running multiple servers across different strata. We can thus regain some robustness by excluding some entities from each stratum so that each entity is absent from at least one stratum. Alternatively, we may use per-node keys and allow free routing, at the cost of slower (in terms of rounds) convergence to a random permutation .

- On input $\big(\mathsf{sid}, \textsc{Init}\big)$, a party $P \in \mathbf{P}$ that is not yet initialised, runs $\mathsf{Gen}(1^\lambda)$ to generate a pair of a private and a public key pair $(\mathsf{sk}_P, \mathsf{pk}_P)$. Then, it broadcasts the message $\big(\mathsf{sid}, (\mathsf{init}, \mathsf{pk}_P), P\big)$ to all clients and SPs by sending $\big(\mathsf{sid}, (\mathsf{init}, \mathsf{pk}_P), P'\big)$ to $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{G}[\mathbb{P}])$, for every $P' \in \mathbf{P} \setminus \{P\}$.
- When SP_i has received $\big(\mathsf{sid}, (\mathsf{init}, \mathsf{pk}_{\mathsf{SP}_j}, \mathsf{SP}_j)$ for every $i \in [N] \setminus \{j\}$, then begins the engagement in the email message exchange with its assigned clients and the other SPs.
- When $\mathsf{MX}_{i,1}$ has received $\big(\mathsf{sid}, (\mathsf{init}, \mathsf{pk}_S, S)$ for every $\mathsf{MX}_{i,j}, j > 1$, it runs $\mathsf{Gen}(1^\lambda)$ to generate stratum key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$. Then, it broadcasts the message $\big(\mathsf{sid}, (\mathsf{init}, \mathsf{pk}_i), \mathsf{MX}_{i,1}\big)$ to all parties P' outside stratum i by sending $\big(\mathsf{sid}, (\mathsf{init}, \mathsf{pk}_i), P'\big)$ to $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}(\mathbf{G}[\mathbf{P}])$. For parties P'' in stratum i it sends $\big(\mathsf{sid}, (\mathsf{init}, (\mathsf{pk}_i, \mathsf{sk}_i)), P''\big)$ instead.

■ **Execution**: Our mixnet operates in rounds. A round consists of t+2 subrounds, each consisting of $t_{sub} \geq \Delta_{\mathsf{net}} + 1$ timeslots. We assume timing information is publicly available. During each subround, messages are only sent during the first timeslot. The remaining timeslots exist to ensure that even delayed messages are

delivered before the next subround. To simplify notation we will introduce three functions on the clock value Cl:

Namely, we define (i) $round(\mathsf{Cl}) := \left\lfloor \frac{\mathsf{Cl}}{t_{sub}(t+2)} \right\rfloor$, (ii) $sub(\mathsf{Cl}) := \left\lfloor \frac{\mathsf{Cl}}{t_{sub}} \right\rfloor$, and (iii) $slot(\mathsf{Cl}) := \mathsf{Cl} \mod t_{sub}$. Essentially, at clock Cl we are in slot $slot(\mathsf{Cl})$ of subround $sub(\mathsf{Cl})$. We also assume that using the above functions use READ_CLOCK to determine the current value of Cl.

Registration is handled as in Section 5. Messages are routed through the mixnet as follows:

- Messages from clients are queued by their SPs until the round begins.
- Once a round begins, in sub-round 0, clients send their messages to the SPs. In sub-round 1, each SP uniformly randomly selects a server in the first stratum to receive each message.
- In the sub-round 2 (3), first-stratum (second) servers tally up their incoming messages and pad them to a multiple of σ. They shuffle them and send $\frac{1}{\sigma}$ of them to each 2nd-stratum (3rd) server. No padding is required afterwards.
- In sub-round i, where $4 \leq i \leq t+1$, the servers of stratum $i-1$ shuffle their received messages and send $\frac{1}{\sigma}$ of them to each server in stratum i+1.
- At the end of sub-round $t+2$, the SPs move messages from their input buffers to client inboxes. .

We will now formally describe our system. Note that some inputs will only have effect when given during particular sub-rounds or when given to certain parties (e.g. only Clients). As in the previous section, $\mathsf{Enc}_{[X]}(Y)$ denotes the encryption of Y under X's public key. For brevity, we use $\mathsf{Enc}_{[x,y]}(m)$ to denote $\mathsf{Enc}_{[x]}\big(\mathsf{Enc}_{[y]}(m)\big)$.

$C_s \in \mathbf{C}$ On input $\big(\mathsf{sid}, \textsc{Send}, \langle C_s@\mathsf{SP}_i, M, C_r@\mathsf{SP}_j\rangle\big)$, if C_s is not registered with an SP_i and $subround(\mathsf{Cl}) = 0$ and $slot(\mathsf{Cl}) = 0$, the client sets $reg = round(\mathsf{Cl})$ and runs the registration operation from Section 5.

$C_s \in \mathbf{C}$ On input $\big(\mathsf{sid}, \textsc{Send}, \langle C_s@\mathsf{SP}_i, M, C_r@\mathsf{SP}_j\rangle\big)$, if C_s is logged in to SP_i, she prepares the message $\big(\mathsf{sid}, \mathsf{Enc}_{[\mathsf{SP}_i]}\big(C_s@\mathsf{SP}_i, \mathsf{Enc}_{[\mathsf{SP}_j]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big)\big)\big)$ to be sent to SP_i. If, in addition the $sub(\mathsf{Cl})$ and $slot(\mathsf{Cl})$ are both 0 and $round(\mathsf{Cl}) > reg$, all prepared messages are sent to SP_i.

$C_r \in \mathbf{C}$ On input $\big(\mathsf{sid}, \textsc{Fetch}, C_r@\mathsf{SP}_j\big)$, if C_r is logged in to SP_j, it sends the message $\big(\mathsf{sid}, C_r@\mathsf{SP}_j, \mathsf{Enc}_{[\mathsf{SP}_j]}(\textsc{Fetch})\big)$ to SP_j which, if $C_r@\mathsf{SP}_j$ is a valid address, it decrypts and forwards all messages $E_{r,1}, \ldots E_{r,n_r}$ in $\mathsf{Inbox}[C_r@\mathsf{SP}_j]$ to C_r, and empties $\mathsf{Inbox}[C_r@\mathsf{SP}_j]$.

$C_r \in \mathbf{C}$ Upon receiving $(\mathsf{sid}, E_{r,1}, \ldots, E_{r,n})$ from SP_j and if C_r has sent a $\big(\mathsf{sid}, \textsc{Fetch}, C_r@\mathsf{SP}_j\big)$ request, C_r decrypts all ciphertexts and stores the ones that are not 0, i.e. they correspond to non-dummy mail messages.

$P \in \mathbf{P}$ On input $\big(\mathsf{sid}, \textsc{Read\_Clock}\big)$, the entity $P \in \mathbf{P}$ sends the message $\big(\mathsf{sid}, \textsc{Read\_Clock}\big)$ to $\mathcal{G}_{\mathsf{clock}}$. Upon receiving $\big(\mathsf{sid}, \textsc{Read\_Clock}, \mathsf{Cl}\big)$ from $\mathcal{G}_{\mathsf{clock}}$, P stores Cl as its local time and forwards the message $\big(\mathsf{sid}, \textsc{Read\_Clock}, \mathsf{Cl}\big)$ to the environment.

$\mathsf{SP}_i \in \mathbf{SP}$ On input $\big(\mathsf{sid}, \mathsf{Enc}_{[\mathsf{SP}_i]}\big(C_s@\mathsf{SP}_i, \mathsf{Enc}_{[\mathsf{SP}_j]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big)\big)\big)$, it checks that $C_s@\mathsf{SP}_i \in \mathbf{Ad}$ and if so, then it decrypts and adds $\big(\mathsf{sid}, C_s@\mathsf{SP}_i, \mathsf{Enc}_{[\mathsf{SP}_j]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big)\big)$ to its set of messages pending to be sent, denoted by L^i_{send}.

$\mathsf{SP}_j \in \mathbf{SP}$ Upon receiving a message $(\mathsf{sid}, \mathsf{Enc}_{[\mathsf{SP}_j]}(\cdot,\cdot))$ from some $\mathsf{MX}_{x,y}$, SP_j checks whether $x = t$, and if the content is a ciphertext under its public key that decrypts as a valid address $C_r@\mathsf{SP}_j$ along with a ciphertext E. If so, then it adds E to $\mathsf{B}[C_r@\mathsf{SP}_j]$.

$\mathsf{MX}_{1,j} \in \mathbf{S}$ On receiving $\big(\mathsf{sid}, \mathsf{Enc}_{[1,\ldots,t]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big), X\big)$, it checks that $X \in \mathsf{SP}$ and if so, it decrypts it and adds $\big(\mathsf{sid}, \mathsf{Enc}_{[2,\ldots,t]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big)$ to its set of messages pending to be sent, denoted by L^i_{send}.

$\mathsf{MX}_{k+1,j} \in \mathbf{S}$ On receiving $\big(\mathsf{sid}, \mathsf{Enc}_{[k,\ldots,t]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big), X\big)$, it checks that $X = \mathsf{MX}_{k,x}$ for some x and if so, it decrypts it and adds $\big(\mathsf{sid}, \mathsf{Enc}_{[k+2,\ldots,t]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big)$ to its set of messages pending to be sent, denoted by L^i_{send}. If $k = t-1$, it instead adds $\big(\mathsf{sid}, \mathsf{Enc}_{[SP_j]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big)$ to the list.

$P \in \mathbf{P}$ On input $(\mathsf{sid}, \textsc{Advance\_Clock})$, the entity $P \in \mathbf{P}$ sends the message $(\mathsf{sid}, \textsc{Advance\_Clock})$ to $\mathcal{G}_{\mathsf{clock}}$.

$\mathsf{SP}_i \in \mathbf{SP}$ On input $(\mathsf{sid}, \textsc{Advance\_Clock})$, If $sub(\mathsf{CI}) = 1$ and $slot(\mathsf{CI}) = 0$, for each message $\big(\mathsf{sid}, C_s@\mathsf{SP}_i, \mathsf{Enc}_{[\mathsf{SP}_j]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big)\big)$ in L^i_{send}, then SP_i sends $\big(\mathsf{sid}, \mathsf{SP}_i, \mathsf{Enc}_{[1,\ldots,t]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big)\big)$ to a randomly selected $\mathsf{MX}_{1,j}$ and removes the message from L^i_{send}. Finally, it sends the message $(\mathsf{sid}, \textsc{Advance\_Clock})$ to $\mathcal{G}_{\mathsf{clock}}$.

$\mathsf{MX}_{k,j} \in \mathbf{S}$ On input $(\mathsf{sid}, \textsc{Advance\_Clock})$, If $sub(\mathsf{CI}) \neq k+1$ or $slot(\mathsf{CI}) \neq 0$, send the message $(\mathsf{sid}, \textsc{Advance\_Clock})$ to $\mathcal{G}_{\mathsf{clock}}$ and return. Otherwise, if $k = 1$ or $k = 2$, $\mathsf{MX}_{k,j}$ pads the list L^i_{send} with $\big(\mathsf{sid}, \mathsf{Enc}_{[k+1,\ldots,t]}(0)\big)\big)$ so that its length is a multiple of σ. The list is then shuffled randomly. For each message $\big(\mathsf{sid}, \mathsf{Enc}_{[k+1,\ldots,t]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big)\big)$ in L^i_{send}, then $\mathsf{MX}_{k,j}$ sends $\big(\mathsf{sid}, \mathsf{MX}_{k,j}, \mathsf{Enc}_{[k+1,\ldots,t]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big)\big)$ to server $\mathsf{MX}_{k+1,j \mod \sigma}$, where j is the message's position on the list, and removes the message from L^i_{send}. Finally, it sends the message $(\mathsf{sid}, \textsc{Advance\_Clock})$ to $\mathcal{G}_{\mathsf{clock}}$.

$\mathsf{MX}_{t,j} \in \mathbf{S}$ On input $(\mathsf{sid}, \textsc{Advance\_Clock})$, If $sub(\mathsf{CI}) = t+1$ and $slot(\mathsf{CI}) = 0$, for each message $\big(\mathsf{sid}, \mathsf{Enc}_{[\mathsf{SP}_j]}\big(C_r@\mathsf{SP}_j, \mathsf{Enc}_{[C_r]}(M)\big)\big)$ in L^i_{send}, $\mathsf{MX}_{t,j}$ forwards it to SP_j. Finally it sends the message $(\mathsf{sid}, \textsc{Advance\_Clock})$ to $\mathcal{G}_{\mathsf{clock}}$.

$\mathsf{SP}_j \in \mathbf{SP}$ On input $(\mathsf{sid}, \textsc{Advance\_Clock})$, If $sub(\mathsf{CI}) = t+2$ and $slot(\mathsf{CI}) = 0$, it moves the contents of every buffer $\mathsf{B}[C_r@\mathsf{SP}_j]$ to the corresponding inbox $\mathsf{Inbox}[C_r@\mathsf{SP}_j]$. Finally it sends the message $(\mathsf{sid}, \textsc{Advance\_Clock})$ to $\mathcal{G}_{\mathsf{clock}}$.

Efficiency & Delivery times. The overhead of the padding is an $O\left(\frac{\sigma^2}{m}\right)$ multiplicative increase in the messages sent, where m is the number of messages sent, which we expect to be low for typical use cases. Disregarding padding messages, the cost to deliver a single email, is $3 + t$ messages compared to 3 in the insecure case (sender to SP_s to SP_r to receiver) or $1 + s \cdot n$ for the "golden standard" solution of Section 5. While in principle this is identical to a cascade

(i.e. single server per stratum) solution, in practice a parallel mix requires a larger t value. The load per mix server is $\frac{m}{\sigma}$ messages, compared to m in a cascade.

The encryption overhead depends on the specifics of the cryptosystem. While naive encryption might cause an exponential blow-up, solutions based on hybrid encryption, or onioning solutions such as Sphinx can reduce the overhead to a small linear factor. Delivery latency is also directly proportional to the length of the mixnet. We note that latency can be significantly reduced by pipelining (i.e. allowing messages to be sent at the end of every subround rather than at the end of the first round only), but we opt to describe the base version for clarity.

Privacy. Here, we will show that the system described above is secure under the weak anonymity definition and leakage function $\mathsf{Leak}_{\mathsf{w.anon}}(\mathsf{ptr}, H)$, defined in Eq. (7). For convenience, we will assume that one timeslot maps to one round.

Theorem 2. *Let* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a PKE scheme that achieves m-IND-CPA security with error* $\epsilon_E(\lambda)$*. Then, the parallel mix email ecosystem of Section 6 over* PKE *and* $\mathcal{G}_{\mathsf{clock}}, \mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}$*, using t strata of* σ *servers to deliver m messages achieves computational* $m^{1-\lfloor \frac{t-1}{2} \rfloor \frac{1}{4}} 4^{\lfloor \frac{t-1}{2} \rfloor \frac{1}{2}} \log m^{\lfloor \frac{t-1}{2} \rfloor \frac{1}{4}} + 2|\mathbf{P}|\epsilon_E$*-privacy for message delay* Δ_{net} *with respect to the weak anonymity leakage function defined below*

$$\mathsf{Leak}_{\mathsf{w.anon}}(\mathsf{ptr}, H) := \begin{cases} \big([\![\mathbf{S}_{\mathsf{ptr}}]\!][H], [\![\mathbf{R}_{\mathsf{ptr}}]\!][H], \mathbf{F}_{\mathsf{ptr}}[H], \mathsf{Act}_{\mathsf{ptr}}[H]\big), & \text{if } \mathsf{End}(\mathsf{ptr}, H) = 1 \\ \big([\![\mathbf{S}_{\mathsf{ptr}}]\!][H], \mathbf{F}_{\mathsf{ptr}}[H], \mathsf{Act}_{\mathsf{ptr}}[H]\big), & \text{otherwise} \end{cases}$$

Proof. We begin by describing the simulator, Sim. The handling of messages to $\mathcal{A}$, timing, message delays and initialization are identical to those in Theorem 1. We note that the adversary's decisions with regard to delays are irrelevant: all externally observable operations are timed to succeed even if all preceding messages are maximally delayed. In addition the simulator stores an internal tally of the sender multiset initialized as $S = \{\emptyset, \emptyset, \emptyset\}$, and pending send pointers P initialized to $\emptyset$.

We now describe how Sim handles the different kinds of leakage it may receive.

- Upon receiving $\big(\mathsf{sid}, \mathsf{ptr}, (\textsc{Register}, C_\ell @ \mathsf{SP}_i)\big)$ from $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}_{\mathsf{w.anon}}, \Delta_{\mathsf{net}}}(\mathbf{C}, \mathbf{SP}, \mathbf{Ad})$, Sim runs the registration protocol between C_ℓ and SP_i exactly as in the description of $\mathbb{E}_{\mathsf{comp}}$, also setting C_ℓ as active. Then, it sends the message $\big(\mathsf{sid}, \textsc{Allow\_Exec}, \mathsf{ptr}\big)$ to $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}_{\mathsf{w.anon}}, \Delta_{\mathsf{net}}}(\mathbf{C}, \mathbf{SP}, \mathbf{Ad})$.
- When Sim receives leakage $\big([\![\mathbf{S}_{\mathsf{ptr}}]\!][H], \mathbf{F}_{\mathsf{ptr}}[H], \mathsf{Act}_{\mathsf{ptr}}[H]\big)$, it compares it with its stored list S. If it differs in the first component, then a client $C_l = [\![\mathbf{S}_{\mathsf{ptr}}]\!][H] \setminus S_1$ is attempting to send a message. First, the simulator will check if the client is registered, and if so, sends a dummy message $\big(\mathsf{sid}, \textsc{Channel}, \mathsf{Enc}_{[\mathsf{SP}_i]}(0), \mathsf{SP}_i\big)$ to the simulated $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}$, and updates S_1, while also adding the handle ptr to P. Otherwise, it simply replies $\big(\mathsf{sid}, \textsc{Allow\_Exec}, \mathsf{ptr}\big)$.
  If the leakage differs in the second component, then a client $C_l = \mathbf{F}_{\mathsf{ptr}}[H] \setminus S_2$ is attempting to fetch her messages. First, the simulator will check if the client is

registered, and if so, sends a dummy message $M = \big(\mathsf{sid}, \textsc{Channel}, \mathsf{Enc}_{[\mathsf{SP}_i]}(0), \mathsf{SP}_i\big)$ to the simulated $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}$, and updates S_2, while also keeping record of M, ptr. Whenever the message delivery of M is allowed, either by $\mathcal{A}$ or automatically when Δ_{net} delay has passed, Sim allows execution of ptr by sending $\big(\mathsf{sid}, \textsc{Allow\_Exec}, \mathsf{ptr}\big)$. Otherwise, it simply replies $\big(\mathsf{sid}, \textsc{Allow\_Exec}, \mathsf{ptr}\big)$. If the leakage differs in the third component, then a client $C_l = \mathsf{Act}_{\mathsf{ptr}}[H] \triangle S_3$ is attempting to register, login or logout. This is handled as in the simulator of Theorem 1.

- When a client C_l registers at SP_j the simulator will receive $\big(\mathsf{sid}, \mathsf{ptr}, (\textsc{Register}, C_\ell @ \mathsf{SP}_i)\big)$. Sim records this, and simulates the rest of the registration protocol. Then, it sends $\big(\mathsf{sid}, \textsc{Allow\_Exec}, \mathsf{ptr}\big)$ to $\mathcal{F}_{\mathsf{priv}}^{\mathsf{Leak}_{\mathsf{w.anon}}, \Delta_{\mathsf{net}}}(\mathbf{C}, \mathbf{SP}, \mathbf{Ad})$.
- When the timeslot advances, Sim receives leakage $\big([\![\mathbf{S}_{\mathsf{ptr}}]\!][H], [\![\mathbf{R}_{\mathsf{ptr}}]\!][H], \mathbf{F}_{\mathsf{ptr}}[H], \mathsf{Act}_{\mathsf{ptr}}[H]\big)$. If the receiver multiset is non-empty, the simulator stores both multisets (potentially overriding their previous values). Note that this only happens during timeslot 0 of subround 0. If $slot(\mathsf{Cl}) = 0$ then the simulator needs to simulate the corresponding subround $sub(\mathsf{Cl})$.
  - For subround 0, the simulator looks at $[\![\mathbf{S}_{\mathsf{ptr}}]\!][H]$ to determine the number of messages originating from each SP. The messages, $\big(\mathsf{sid}, \mathsf{SP}_i, \mathsf{Enc}_{[1,\ldots,t]}(0)\big)$ are each sent to a uniformly randomly selected first stratum server via the simulated $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}$. The simulator then notes the messages stored in each server $\mathsf{MX}_{1,i}$.
  - For subrounds 1 to t-1, the simulator knows the messages stored in each server and is able to simply follow the protocol (including padding) by processing the messages and routing them through the simulated $\mathcal{F}_{\mathsf{auth}}^{\Delta_{\mathsf{net}}}$.
  - For subround t, the simulator replaces $|[\![\mathbf{R}_{\mathsf{ptr}}]\!][H]|$ randomly selected messages in MX_t with $[\![\mathbf{R}_{\mathsf{ptr}}]\!][H]$ and sends dummy messages to the corresponding SPs following the protocol. At the same time, it allows execution of all pending sends in P, and resets it to empty.

For the next part, we will use a series of $|\mathbf{P}| + 1$ hybrid games, H_0 to $H_{|\mathbf{P}|}$. H_0 represents a real execution of the protocol. H_i, for $i \in \{1, |\mathbf{P}|\}$ is identical to H_{i-1} but includes a challenger who embeds an m-IND-CPA challenge in the messages addressed to the i-th party, and encrypts only zeroes for parties $j > i$. For this, we order the parties so that the stratum order is reversed (i.e servers in the last stratum appear first). It also internally labels dummy ciphertexts with their corresponding message and uses the labels when the challenge and dummy messages are supposed to be decrypted. We note here, that the challenger in H_i cannot be used to instantiate a simulator as it requires full knowledge of the messages and metadata in addition to the specified leakage.

By the m-IND-CPA security of the encryption scheme, games H_i, H_{i-1} are computationally indistinguishable, as they only differ in the contents of ciphertexts addressed to party i.

The only difference between $H_{|\mathbf{P}|}$ and the simulation is that $H_{|\mathbf{P}|}$ is able to shuffle the messages that would pass through each server (using the labels and side input), whereas the simulation routes zeroed out messages throughout the

network, and decides randomly which server a received message will be delivered from.

To complete the proof, we use Theorem 3 (based on [21]) which shows that for large t the distribution of messages in game H_p is statistically indistinguishable from that of the simulation. Accounting for the computational security of Enc, we obtain that H_0 is computationally indistinguishable to the (efficient) simulator.

The adversary's advantage in distinguishing between games H_i and H_{i+1} is $2\epsilon_E$, so the total advantage is bounded by $2|\mathbf{P}|\epsilon_E$. The adversaries advantage in distinguishing H_p from the simulation is bounded by the statistical difference between the two distributions, which is bounded by $N^{1-\lfloor\frac{t-1}{2}\rfloor\frac{1}{4}}4^{\lfloor\frac{t-1}{2}\rfloor\frac{1}{2}}\log N^{\lfloor\frac{t-1}{2}\rfloor\frac{1}{4}}$. Thus the total advantage is bounded by $N^{1-\lfloor\frac{t-1}{2}\rfloor\frac{1}{4}}4^{\lfloor\frac{t-1}{2}\rfloor\frac{1}{2}}\log N^{\lfloor\frac{t-1}{2}\rfloor\frac{1}{4}} + 2|\mathbf{P}|\epsilon_E$. □

7 The Combinatorics of Parallel Mixing

Many of the works analysing parallel mixing investigate the probability distribution of a single message traversing the network. This is satisfactory for some definitions of anonymity but not for our modelling of a global adversary under universal composability. In our model, the environment determines the sender and receiver of each message, so it is not sufficient to argue that any one message is successfully shuffled (i.e has a uniformly random exit point from the network).

To illustrate, assume messages are represented by a deck of n playing cards, and further assume that our mixnet operates by simply "cutting" the deck once, in secret (i.e choosing $k \in \{0..n-1\}$, and placing the first k cards at the bottom of the deck in their original order). It is trivial to simulate drawing a single card from a deck shuffled this way, by sampling a random card. However, once a card has been drawn, subsequent draws are determined by the initial order. The environment knows the initial order because it set it, but the simulator does not, and the simulation fails.

Our approach will be to show that parallel mixing after a number of rounds produces a random permutation on the list of input messages, thus allowing the simulator to produce the list of output messages by sampling a random permutation of the recipients, independent of the senders (which is crucial as it does not know the relation between the two).

We will model parallel mixing as a generalisation of the square lattice shuffle of Håstad [21]. In a square lattice shuffle, $n = m^2$ cards are arranged in an $m \times m$ matrix, and shuffled as follows: in odd rounds each row is shuffled by an independently uniformly random sampled permutation. In even rounds, the same happens to columns. It is simple to check that t iterations of this process map directly to a t-stratum parallel mix with m servers per stratum, each with capacity m: we label odd strata as "rows" and even strata as "columns", where the i-th server corresponds to the i-th row (column). The mapping is then completed by noting the result of an odd round is that each row randomly contributes one of its elements to each column, and vice-versa for even rounds.

Thus Håstad's results are applicable to parallel mixing. A second observation is that because parties are assumed honest, we can assign multiple rows or columns to one party without invalidating the bounds. We thus reproduce Theorem 3.6 from [21] and explain how it applies in our construction.

Theorem 3 (Håstad [21], Theorem 3.6). *Let Π_t be the distribution defined by t iterrations of lattice shuffling on m objects. Then*

$$\Delta(\Pi_t, U_m) \leq O(m^{1-\lfloor \frac{t-1}{2} \rfloor \frac{1}{4}} \log m^{\lfloor \frac{t-1}{2} \rfloor \frac{1}{2}})$$

Closer examination of the proof, and assuming $m > 81$ enables us to dismiss the big-O and obtain:

$$\Delta(\Pi_t, U_m) \leq m^{1-\lfloor \frac{t-1}{2} \rfloor \frac{1}{4}} 1.5^{\lfloor \frac{t-1}{2} \rfloor} \log m^{\lfloor \frac{t-1}{2} \rfloor \frac{1}{2}}$$

This in turn implies

Corollary 1. *For $m > 10^6$, 31 rounds of lattice shuffling are statistically $\frac{1}{m}$ close to uniform.*

The theorem's proof also gives us insight in the effect of compromised servers in a stratum: as coupling takes place over 3 iterations (or 2 with the assumption that another honest iteration will follow), we must allow that a single compromised stratum essentially shortens our network by 3 strata at the worst case.

7.1 A brief discussion on convergence speed

The bounds stated above describe a parallel mix with many small servers. One would expect the situation to improve when examining fewer, larger servers. In that direction, we expect a generalization of Håstad's result to yield a tighter bound. That would be of value as there are few competing designs for random permutation networks suited to anonymous communication [26].

The core of Håstad's analysis is about the probability of "coupling" two permutations that start out differing by a single transposition, after 2 rounds of shuffling. A first observation is that with "large" servers, the probability that the transposition lies in one server (and thus the coupling is immediate) becomes significant, improving convergence. A second, is that the probability of a missed coupling is inversely proportional to the number of elements per server which again implies improved convergence. We believe that a bound of $m^{1-\lfloor \frac{t-1}{2} \rfloor \frac{1}{2}} 1.5^{\lfloor \frac{t-1}{2} \rfloor} \log m^{\lfloor \frac{t-1}{2} \rfloor \frac{\sigma-1}{\sqrt{\sigma}} \lfloor \frac{t-1}{2} \rfloor}$ is possible, which would approximately halve the rounds required for the bound to reach $\frac{1}{m}$, when σ is small, e.g. 17 rounds for $\sigma = 4$, $m > 300.000$. However, we consider the specifics outside the scope of this work, and leave the question of statistical bounds for parallel mixing open for further research.

References

1. Alexopoulos, N., Kiayias, A., Talviste, R., Zacharias, T.: MCMix: Anonymous messaging via secure multiparty computation. In: USENIX (2017)
2. Angel, S., Setty, S.: Unobservable communication over fully untrusted infrastructure. In: OSDI (2016)
3. Backes, M., Kate, A., Manoharan, P., Meiser, S., Mohammadi, E.: Anoa: A framework for analyzing anonymous communication protocols. In: CSF (2013)
4. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: A composable treatment. In: CRYPTO (2017)
5. Camenisch, J., Lysyanskaya, A.: A formal treatment of onion routing. In: CRYPTO. pp. 169–187. Springer (2005)
6. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Foundations of Computer Science. IEEE (2001)
7. Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: TCC (2007)
8. Chaidos, P., Fourtounelli, O., Kiayias, A., Zacharias, T.: A universally composable framework for the privacy of email ecosystems. In: ASIACRYPT (2018)
9. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of cryptology **1**(1), 65–75 (1988)
10. Chaum, D., Das, D., Javani, F., Kate, A., Krasnova, A., de Ruiter, J., Sherman, A.T.: cmix: Mixing with minimal real-time asymmetric cryptographic operations. In: ACNS. pp. 557–578 (2017)
11. Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM **24**(2), 84–90 (1981)
12. Corrigan-Gibbs, H., Boneh, D., Mazières, D.: Riposte: An anonymous messaging system handling millions of users. In: Security and Privacy (2015)
13. Corrigan-Gibbs, H., Ford, B.: Dissent: accountable anonymous group messaging. In: CCS. pp. 340–350 (2010)
14. Danezis, G., Dingledine, R., Mathewson, N.: Mixminion: Design of a type III anonymous remailer protocol. In: Security and Privacy. pp. 2–15 (2003)
15. Danezis, G., Goldberg, I.: Sphinx: A compact and provably secure mix format. In: Security and Privacy (2009)
16. Díaz, C., Seys, S., Claessens, J., Preneel, B.: Towards measuring anonymity. In: PETS (2002)
17. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. Tech. rep., DTIC Document (2004)
18. Dwork, C.: Differential privacy. In: Automata, Languages and Programming. pp. 1–12 (2006)
19. Golle, P., Juels, A.: Parallel mixing. In: CCS. pp. 220–226. ACM (2004)
20. Goodrich, M.T., Mitzenmacher, M.: Anonymous card shuffling and its applications to parallel mixnets. In: Automata, Languages, and Programming. pp. 549–560. Springer (2012)
21. Håstad, J.: The square lattice shuffle. Random Structures & Algorithms **29**(4), 466–474 (2006)
22. Johnson, A., Wacek, C., Jansen, R., Sherr, M., Syverson, P.: Users get routed: Traffic correlation on tor by realistic adversaries. In: CCS. pp. 337–348 (2013)
23. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: TCC (2013)

24. Kesdogan, D., Egner, J., Büschkes, R.: Stop-and-go-mixes providing probabilistic anonymity in an open system. In: International Workshop on Information Hiding. pp. 83–98. Springer (1998)
25. Kotzanikolaou, P., Chatzisofroniou, G., Burmester, M.: Broadcast anonymous routing (BAR): scalable real-time anonymous communication. Int. J. Inf. Sec. **16**(3), 313–326 (2017)
26. Kwon, A., Corrigan-Gibbs, H., Devadas, S., Ford, B.: Atom: Horizontally scaling strong anonymity. In: SOSP. ACM (2017)
27. Kwon, A., Lazar, D., Devadas, S., Ford, B.: Riffle: An efficient communication system with strong anonymity. PoPETS **2016**(2), 115–134 (2015)
28. Pfitzmann, A., Hansen, M.: A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management. Version v0.34 http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf (August 2010)
29. Pfitzmann, A., Köhntopp, M.: Anonymity, unobservability, and pseudonymity—a proposal for terminology. In: Designing privacy enhancing technologies. Springer (2001)
30. Piotrowska, A., Hayes, J., Elahi, T., Danezis, G., Meiser, S.: The loopix anonymity system. In: USENIX (2017)
31. Samarati, P., Sweeney, L.: Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. In: Security and Privacy (1998)
32. Serjantov, A., Danezis, G.: Towards an information theoretic metric for anonymity. In: Privacy Enhancing Technologies. pp. 41–53. Springer (2002)
33. Shmatikov, V., Wang, M.: Timing analysis in low-latency mix networks: Attacks and defenses. In: ESORICS. pp. 18–33 (2006)
34. Syverson, P.F., Goldschlag, D.M., Reed, M.G.: Anonymous connections and onion routing. In: Security and Privacy. pp. 44–54 (1997)
35. Syverson, P.F., Tsudik, G., Reed, M.G., Landwehr, C.E.: Towards an analysis of onion routing security. In: Workshop on Design Issues in Anonymity and Unobservability. pp. 96–114 (2000)
36. Van Den Hooff, J., Lazar, D., Zaharia, M., Zeldovich, N.: Vuvuzela: Scalable private messaging resistant to traffic analysis. In: SOSP. pp. 137–152 (2015)
37. Wikström, D.: A universally composable mix-net. In: TCC. pp. 317–335. Springer (2004)