CHORUS: Secret Recovery with Ephemeral Client Committees

Deevashwer Rathee
UC Berkeley

Emma Dauterman
Stanford University

Allison Li
Okta

Raluca Ada Popa
UC Berkeley

***Abstract*—End-to-end encrypted applications protect user data by ensuring that user secrets are only available on client devices. However, if a user loses all of their devices, they need a way to recover their data using only a short password. To realize a password-based secret recovery system resilient to brute-force attacks, prior works relied on secure hardware or a few non-colluding servers.**

In this work, we take a conceptually different approach that distributes trust across the many clients already in the system, while using the server only as an orchestrator without relying on it for privacy. To achieve this, we design and implement Chorus, a secret recovery system that employs ephemeral committees, each consisting of approximately a thousand clients, to provide strong privacy with high scalability. Committees change frequently in Chorus, typically on the order of a few minutes, to severely limit an attacker's ability to compromise clients on a committee. We design Chorus for unreliable, resource-constrained clients and show that the per-client overhead decreases as more clients join the system.

Assuming each user performs recovery once a year, the expected per-client overhead in Chorus is under 30 s of computation on a mobile device and 13.2 MB of communication, both incurred only once every four months in a configuration with 100M clients, up to 50M of which may be offline and at most 10M may be compromised. To achieve this performance, we contribute two key techniques: (i) a password-based secret recovery scheme that confines expensive committee interactions to infrequent, latency-tolerant operations, and (ii) a non-interactive verifiable secret-sharing scheme that reduces client overhead by two orders of magnitude by delegating computation to the server.

1. Introduction

The privacy of end-to-end encrypted applications [1], [2], [3], [4], [5], [7], [8] fundamentally rests on the privacy of users' secrets. In such systems, data is encrypted locally on the user's device and can only be decrypted using an on-device secret. Even if an attacker compromises the application's servers or the network, they cannot access users' private information. This strong privacy guarantee has driven the widespread adoption of end-to-end encrypted messaging applications such as Signal [54].

While requiring client devices to manage user secrets provides strong privacy, it also creates new challenges. If users lose all their devices, they should be able to access their data. To address this, many applications provide a mechanism to back up user secrets to the application provider without revealing them. For convenience, these backups are typically automated [17], [55], [81], [148], since manual processes [129] (e.g., writing a key on paper) often deter users from performing backups at all [151].

Automated backups encrypt user secrets under a key that the user can recover even after losing all devices. Since this key must ultimately be derived from something the user can remember, and users are unlikely to recall a cryptographically strong key, many systems instead rely on short, human-memorable passwords or PINs [88]. The challenge then lies in using a low-entropy password to protect a secret from a compromised application provider. Simply deriving a key from the password and storing the ciphertext on the server is insufficient, since a compromised server can brute-force the password to recover the secret. We therefore need a mechanism that provides strong protections using only a cryptographically weak secret.

To protect user backups from brute-force attacks, prior work limited the number of password guesses by relying primarily on secure hardware [17], [55], [60], [81], [143], [148] or a few non-colluding servers, at least one of which is honest [51], [98], [103], [111]. While secure hardware raises the bar for the attacker, it is not immune to compromise [38], [39], [118], [135], [145], [146], and splitting secrets across a few non-colluding servers means that an attacker only needs to compromise or subpoena [140], [149] a small set of servers to extract many users' secrets.

In this work, we take a conceptually different approach to secret backup: we distribute trust across the *many clients* already in the system and use the server solely as an orchestrator, without relying on it for privacy. The shift of trust from a few servers to numerous clients echoes broader trends in federated learning [22], [92], [94], [109], [112] and federated analytics [131], [132], [133], and aligns with the general principle of decentralized trust [107].

Guided by this principle, we design and build Chorus[1], a secret recovery system that defends against an attacker who can compromise the application server and at most a fraction (e.g., 10% of 100M) of clients simultaneously. In Chorus, each user's secret is split across a committee of roughly one thousand clients. To reduce targeted attacks on committee members, committees rotate quickly, typically every few minutes, which severely limits an attacker's ability to compromise clients on the current committee. Frequent rotation also balances load: as more clients join the system,

1. Chorus stands for **C**lients-**HO**sted **R**ecovery of **U**ser **S**ecrets.

the expected per-client overhead decreases, making Chorus well-suited for large-scale deployments.

Our design of Chorus is inspired by the theoretical line of work on evolving-committee proactive secret sharing (ECPSS) [27]. In an ECPSS scheme, a small, randomly chosen committee maintains secrets on behalf of a much larger group and periodically transfers these secrets to a new committee. While this committee-based approach is promising for distributing trust across the many clients in Chorus, existing ECPSS and secret recovery schemes are ill-suited for our setting: ECPSS was developed for blockchains and its private-state transfers are expensive for unreliable, resource-constrained clients (§3.2.1), and existing secret recovery schemes assume frequent interactions with a server committee that the client committees cannot sustain (§3.1).

To bridge these gaps, we co-design both schemes and introduce two new techniques. First, we propose a *server-aided* non-interactive verifiable secret sharing scheme that securely offloads computation to the server, reducing client overhead during ECPSS state transfer by two orders of magnitude (§3.2.1). Second, we design a secret recovery scheme with *non-interactive backups* that confines costly committee interactions to just the rare, catastrophic recovery events (§3.1). When composed, these schemes achieve low client overhead and minimal client interaction: both schemes operate without intra-committee communication, backups are non-interactive, and recovery completes in a single round. Orthogonally, we also propose a new committee selection technique (§3.2.2) that reduces trust assumptions relative to prior work [92]. Although developed for Chorus, all three techniques are of independent interest.

We implement and evaluate Chorus. Assuming each user performs recovery once a year, the expected per-client overhead in Chorus is under 30 s of computation on a mobile device and 13.2 MB of communication, both incurred only once every four months in a configuration with 100M clients, up to 50M of which may be offline and at most 10M compromised (§7.4.2). The most expensive client operation is recovery, which typically incurs a latency of a few minutes and takes 16s of compute on a mobile device (§7.4.1); recovery occurs infrequently and only when the user loses all their devices. We estimate that running a Chorus server costs roughly $27.6K/month for 100M users (§7.5), which is only 2.37% percent of Signal's operating costs [150] for tens of millions of users [55].

Limitations. Chorus inherits the limitations of systems that distribute trust across clients [31], [56], [69], [109], [112], [119], [131], [132], [133], [138]: it assumes a public key infrastructure (PKI), and its privacy guarantees rely on the PKI's effectiveness in limiting fake or Sybil clients (§2.1). Chorus does not defend against an attacker capable of exploiting a single vulnerability to compromise a large fraction of clients simultaneously. To reduce such risks, clients should run on diverse device types, and the client software should be open source, making large-scale compromise more difficult. Finally, Chorus can be combined with complementary defenses, such as trusted hardware or non-colluding servers, for a defense-in-depth deployment.

2. System Overview

2.1. Entities

Clients. Clients run on mobile devices or laptops, and they have limited bandwidth, compute, and storage. They can join the protocol at any point, leave without warning, lose state, or go offline for long periods of time. A client can play two roles in the system: (1) backing up or recovering data on behalf of a user, or (2) participating in the protocol to manage the keys to users' secrets.

Server. The server providing the backup service to users has large amounts of compute, storage, and bandwidth.

Transparency log. Our construction makes black-box use of a non-forking transparency log [108], [113], and the functionality we require is described in Figure 12. The server and clients in our construction could also participate in maintaining state for the transparency log. If an application provider already has transparency log infrastructure for distributing public keys [53], [77], [105], they could re-use some of this infrastructure for the Chorus transparency log.

Public key infrastructure (PKI). Chorus also makes black-box use of a PKI that registers clients' signature verification keys (see Figure 11). User backups are protected as long as at most a configurable fraction of registered clients are compromised, and the PKI determines which clients are registered. For instance, the PKI can leverage device uniqueness checks on iPhone [15] to enforce one registration per device. We discuss other PKI instantiations in §9.

2.2. High-level Workflow

In Chorus, clients encrypt their backups under a master public key pk and upload the ciphertexts to the application server. At any time, a single committee of clients holds shares of the corresponding master secret key msk, which is used to decrypt backups and enforce guess-limiting on recovery requests for all users. To balance load and limit targeting of any client over time, the current committee periodically re-shares msk to a newly formed committee each epoch. Each committee is selected uniformly at random from the client population through an unbiasable, verifiable lottery (§5.3). Client committees also participate in the initial distributed key generation (DKG) protocol (Appendix A.2), which establishes pk and the corresponding msk shares. Figure 1 illustrates the backup and recovery workflow in more detail.

2.3. Communication Model

The clients and the server are connected through a star topology with the server in the center, as is standard for end-to-end encrypted applications [1], [2], [3], [4], [5], [7], [8]. Thus, the server acts as a router for all the clients, and it

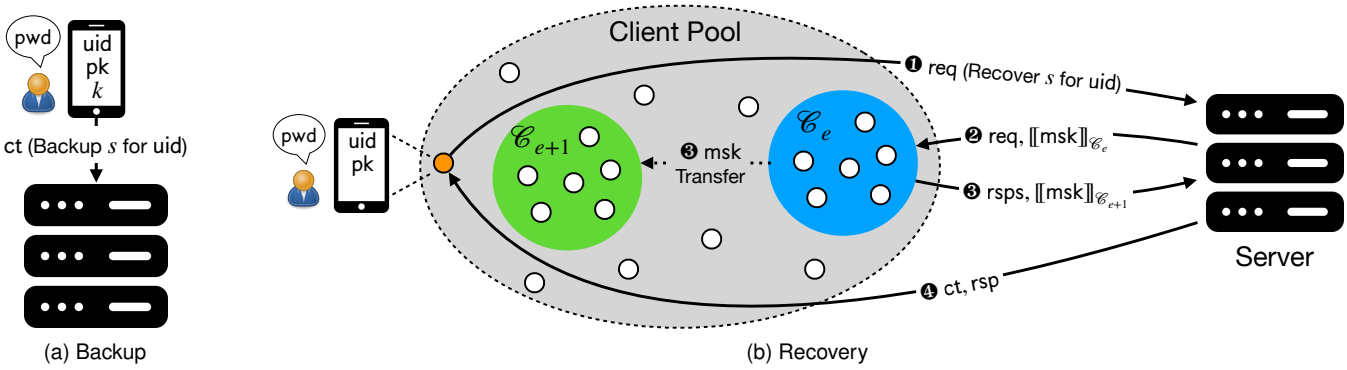


Figure 1. High-level workflow of backup (Figure 1a) and recovery (Figure 1b) in Chorus. Chorus has a master secret key msk and a corresponding public key pk. It operates over epochs, and during an epoch e, a client committee $\mathcal{C}_e$ holds shares of msk and uses them to answer recovery requests. Given pk and a user-chosen password pwd, client $\mathcal{P}_{\text{uid}}$ can back up a secret s by simply uploading a backup ciphertext ct to the server. Recovery in epoch e proceeds as follows: ❶ Given pwd from the user, client $\mathcal{P}_{\text{uid}}$ samples a random blind r and creates a recovery request rqst that hides pwd and sends it to the server. ❷ The server sends rqst and encrypted shares $[\![\text{msk}]\!]_{\mathcal{C}_e}$ of msk to committee $\mathcal{C}_e$ that it received from $\mathcal{C}_{e-1}$ in the previous epoch. ❸ $\mathcal{C}_e$ generates the recovery responses resps and also transfers msk to the next committee $\mathcal{C}_{e+1}$ by sending encrypted shares $[\![\text{msk}]\!]_{\mathcal{C}_{e+1}}$ to the server. ❹ Finally, the server processes resps and sends the backup ciphertext ct and the final recovery response resp to $\mathcal{P}_{\text{uid}}$. Given resp and ct, $\mathcal{P}_{\text{uid}}$ can use the blind r to recover the secret s. For the sake of simplicity, we only show one recovery request per epoch. Chorus also relies on a transparency log and a PKI for security (§2.1), and we also omit them for simplicity.

can arbitrarily drop and schedule their messages. To prevent long-term compromise, clients on a committee timeout after δ time and erase their shares (see §2.4). The application developer should configure δ so that enough clients have a chance to send their messages before they timeout. The performance of Chorus (and the duration of an epoch) is independent of δ and only depends on the actual message delivery delays and the computation time. As such, δ can be set conservatively to balance privacy and availability without affecting performance: a smaller δ makes it harder to corrupt committee members before they delete their secrets, while a larger δ makes the network assumption more viable.

2.4. Threat Model and Security Properties

Below, we summarize Chorus's threat model and high-level security guarantees. §6 formalizes these guarantees and gives the corresponding proof intuition.

Privacy. Chorus hides user secrets from an active, mobile adversary that may corrupt the application server and at most a configurable fraction κ of the application clients. Corrupted parties may behave arbitrarily and collude; in particular, a corrupted server may drop, delay, or tamper with honest clients' messages. The adversary is mobile, capable of compromising any honest client over time provided the total corruption fraction stays below κ. We limit the adversary's speed such that corrupting a client requires at least δ time and grants full access to its private state, where δ is a configurable system parameter that balances privacy and availability (§2.3). Specifically, δ defines the time-to-live (TTL) after which a client that has announced its committee nomination erases its encryption keys [14], [16] to defend against the mobile adversary. This client later re-joins the protocol with freshly sampled encryption keys (§5.2).

An attacker that corrupts the server can also make a small, configurable number of password attempts—if the user chooses a strong password, this mechanism provides strong protection. To prevent malicious clients from doing the same, Chorus, like prior works, relies on the honest server to employ alternative authentication factors (see "Authentication Server" in SVR3 [55]). The server may also offer a grace period for clients to counter spoofed recovery attempts by proving knowledge of their secret key.

Availability. Chorus ensures availability (i.e., allows clients to recover their secrets) as long as the application server is honest and at most a configurable fraction ϕ of clients are offline at any moment ($\phi \geq \kappa$ as compromised clients can choose to not respond). Note that relying on the server for availability does not introduce an additional trust assumption, as end-to-end encrypted applications typically operate in a star topology and already make this assumption (§2.3). Chorus seamlessly incorporates client membership changes as the PKI registers new clients or revokes existing ones.

Parameter selection. Chorus allows an application developer to configure the maximum fraction of clients an attacker can compromise without violating privacy, κ, and the maximum fraction of clients that can go offline while preserving availability, ϕ. We analyze the relationship of these parameters to committee size n, threshold t, and the total number of clients N in Appendix A.1. In our evaluation (§7), we assume $\kappa = 10\%$ and $\phi = 50\%$. We also consider the case where $\phi = 75\%$ and $\kappa = 1\%$, in line with prior federated learning works [112], [137] that argue that 1% is reasonable or even impractically high [137] for large-scale production deployments. We leave it to the application developer to choose an optimal δ for their application as this parameter does not affect performance §2.3.

2.5. Performance Requirements

As clients are likely running on mobile devices, Chorus needs to account for clients that are both unreliable and resource-constrained. To make the paradigm of secret recovery through client committees practical, we require our design to satisfy the following properties:

RQ 1 (Non-interactive Committee): To minimize the time a client needs to be online, we require that the clients on the committee only need to fetch some data, perform some computation, and send a message (they do not have to stay online and wait for other clients on the committee).

RQ 2 (Constant State): The committee state size should be constant and independent of the number of users as state transfers are frequent and members are resource-constrained.

RQ 3 (Single-Round Recovery): Recovering a backup should only require interacting once with the committee, so that it takes roughly the duration of an epoch (a few minutes typically)—we consider this acceptable as recovery is only performed in the catastrophic event where a user loses access to all of their devices.

RQ 4 (Non-interactive Backup): While recoveries are infrequent, backups happen regularly, so backing up should be non-interactive (i.e., the client simply uploads a ciphertext to the server and does not interact with the committee).

RQ 5 (Lightweight Client): The protocol should minimize client compute and communication, thereby also reducing mobile device battery consumption.

3. Technical Overview

Chorus is a password-based secret recovery system that allows users to create password-protected backups and later recover the underlying secret using only a password. Since passwords have low entropy, the system must enforce guess-limited recovery to prevent brute-force attacks. This enforcement is handled by a key manager entity, which maintains the state needed for recovery. To avoid relying on a single trusted entity, the key manager is "thresholdized" using Shamir secret sharing [136] to distribute the recovery state among a committee of n parties, remaining secure as long as fewer than threshold $t \leq n$ members are compromised.

Our goal is to distribute trust in the key manager across millions of application clients. To achieve this efficiently, we draw inspiration from the evolving committee proactive secret sharing (ECPSS) paradigm [27], used in blockchains to distribute trust across a large population at the cost of thresholding over a much smaller committee. In this model, random subsets of the population take turns managing the secret state, spreading trust across the entire population over time. We apply this paradigm in the client–server setting, where committees are sampled from the client pool.

Overall, Chorus is a combination of two components: (1) a password-based secret recovery scheme with a threshold key manager (§3.1), and (2) an ECPSS scheme in the client-server setting (§3.2). We co-design both components such that their composition, where the role of the threshold key manager is performed by the dynamic client committees, meets our performance goals (§2.5). We now explain our design for these components in more detail and discuss how we organize the rest of the paper in §3.3.

3.1. Secret Recovery Scheme

Secret recovery schemes with a threshold key manager have a common blueprint (see §8 for the full discussion): the user knows a low-entropy password pwd, the key manager committee holds t-out-of-n Shamir shares of a high-entropy master secret key msk, and the backup key k is derived from pwd, the user's unique identifier uid, and msk to prevent brute-force attacks. The key derivation is done via a multi-party computation (MPC) [79], [152] between the client and the key-manager committee, which hides pwd and msk but not uid. The client invokes this MPC during both backup and recovery to compute k, while the committee uses uid to guess-limit recovery.

This approach, however, has an inherent limitation: it requires committee interaction during backup, violating our non-interactive backup requirement (RQ 4). While acceptable for small, static committees, interactive backups become infeasible for frequently changing committees with thousands of lightweight clients and millions of backups. For instance, with 100M users and client committees capable of handling 1K backups every two minutes, initial backups alone would take roughly 138 days.

To overcome this limitation, we propose a new secret recovery scheme that supports non-interactive backups while preserving other performance goals (§2.5). Unlike the above blueprint, our approach uses an asymmetric primitive allowing clients to encrypt secrets locally using the key manager's public key pk. To guess-limit recovery, the encryption and recovery must still bind to uid, pwd, and msk like the blueprint. We achieve this asymmetric binding through threshold anonymous identity-based encryption (IBE) [33]. In IBE, given a public key pk, anyone can encrypt to an identity id, and only an id-specific decryption key d_{id} derived from msk can decrypt. By setting $\mathsf{id} = \mathsf{uid} \| \mathsf{pwd}$, we ensure that d_{id} is a function of msk, pwd, and uid, thereby tying encryption and recovery to the same credentials. Meanwhile, the anonymity property of IBE guarantees that ciphertexts reveal nothing about the user's password.

The IBE scheme does not hide $\mathsf{id} = \mathsf{uid} \| \mathsf{pwd}$ during the decryption key extraction, which reveals user passwords to the threshold key manager during recovery. We therefore want an extraction protocol that hides pwd in id, but the key manager should learn uid so that it can guess-limit. We can leverage existing techniques [32] to build a blind extraction protocol that meets our efficiency requirements, but this protocol is fully blind (hides id entirely). To make the request partially blind and reveal only uid, we use a zkSNARK [30] and show that the proof generation cost is reasonable given that recovery is a rare operation (§7.4.1).

Ultimately, the key manager's private state consists of a single field element, msk, which it uses to facilitate and

guess-limit recoveries. It also has a public state that tracks the number of recovery attempts per uid. The key manager stores only a short commitment to this state, while the application server maintains the full record. When forwarding recovery requests, the server updates the state commitment and includes a proof showing that the update is valid with respect to the previous commitment and the new requests. The next section describes how dynamic client committees securely manage the private state (transferring the public state commitment is straightforward through signatures).

3.2. Evolving Committee Proactive Secret Sharing

In this section, we describe the design of our evolving committee proactive secret sharing (ECPSS) scheme, which allows dynamic client committees to manage a private state. A (t, n, N)-ECPSS scheme consists of (i) *committee selection*: sampling n-sized committees out of an N-sized population, and (ii) *committee handover*: transferring the private state between committees while maintaining the invariant that each committee holds a t-out-of-n Shamir secret sharing of the private state. Recall that the private state of the threshold key manager from §3.1 is msk and the committee members hold t-out-of-n Shamir shares of msk. Thus, given our ECPSS scheme, it is straightforward to compose the two schemes such that the client committee currently holding the state plays the role of the key manager and the committee members follow the secret recovery protocol exactly. Furthermore, we design both the secret recovery scheme and the ECPSS scheme such that intra-committee interaction is not required. Thus, their combination satisfies the non-interactive committee requirement (RQ 1). Overall, the server sends a single batch of recovery requests to the current committee, which processes that batch while simultaneously transferring the state to the next committee. We now discuss the design of committee handover and selection, the two core procedures of our ECPSS scheme.

3.2.1. Committee Handover. The handover is straightforward to construct with Shamir secret-sharing: each sender committee member re-shares its own share of the private state to the receiver committee, and the receiving committee members combine $O(n)$ shares they receive to get their own Shamir share of the private state. While this is enough for correctness, the protocol is not robust as compromised committee members can send malformed shares. To address this issue, prior works use a non-interactive verifiable secret sharing (NIVSS) scheme [46], [73], [96]: the verifiability allows honest parties to detect malicious behavior, and the non-interactivity preserves the one-shot communication pattern, avoiding intra-committee interaction. The challenge here, however, is that the existing NIVSS schemes impose impractical overheads on the clients and the server in our setting, violating the lightweight client requirement (RQ 5). Concretely, for 1000-client committees and 2-minute epochs, using existing tools, the server would need to send 655 GB of data every epoch (costing roughly $\$1.2$M/month), and the client would use 2.5% of its battery, compute for 318 seconds, and download 601 MB (§7.3).

To this end, we observe that delegating availability guarantees to the server and relying on the clients only for privacy can significantly improve both client and server costs. Importantly, this delegation does not degrade the end-to-end robustness guarantee: unlike a fully decentralized setting like blockchain, Chorus operates in a setting where any VSS scheme among clients would inherently rely on the server for message delivery (§2.3). Given this insight, we propose a hybrid non-interactive verifiable secret sharing scheme (§5.4) that combines an expensive and publicly verifiable NIVSS scheme [96] with a weaker and more efficient scheme that only provides security with abort [52]. Our hybrid scheme allows the server to verify malicious proofs and relay only the well-formed (security with abort) proofs to the clients; a misbehaving server only impacts availability. This partitioning of work ultimately reduces the server communication by $100\times$, the client communication by $111\times$, and the client's runtime by $147\times$ for $n \approx 1000$ (§7.3).

3.2.2. Committee Selection. This procedure samples a random subset of n clients from a population of size N in a publicly verifiable manner. State-of-the-art committee selection schemes [75], [92] rely on verifiable random functions (VRFs) to achieve unbiasable and publicly verifiable selection. To prevent bias, the VRF keys must be independent of the global input seed. Lotto [92], the state-of-the-art in the client-server setting, ensures this property by assuming that a PKI samples the VRF keys for all clients honestly and independently of the seed. This, however, changes the standard PKI operation and introduces additional trust assumptions. Ideally, clients should generate their own keys while the PKI only verifies identities during registration.

Instead of restricting how clients sample their VRF keys, we focus on the seed-selection mechanism to ensure the independence of the seed and the client's keys. Our key idea is to make the seed a function of the VRF keys of all participating clients such that any attempt to manipulate the VRF key to bias the committee selection results in unpredictable and random changes to the seed. Since clients cannot perform this $O(N)$ computation on their own, we set up this computation such that it can be efficiently verified with $O(\log N)$ complexity. With that, the $O(N)$ seed selection computation can be delegated to the server; a malformed seed only affects availability. Concretely, we propose a simple Merkle-tree-based construction that satisfies these properties in the random oracle model [25].

3.3. Organization

We decompose Chorus into two modular components: (1) a password-based secret recovery scheme that assumes access to an ideal IBE key-manager functionality, and (2) a key-manager scheme that securely realizes this functionality using our client ECPSS construction. We model the IBE key manager as an ideal functionality rather than directly

instantiating a threshold IBE scheme to (i) simplify the security proof of the recovery scheme, (ii) enable reuse of the recovery scheme with alternative realizations (e.g., trusted hardware), and (iii) make our client-committee realization applicable to other threshold-IBE applications [48].

We present the secret recovery scheme with an idealized key manager in §4 and our client ECPSS scheme in §5. We abstract away and defer the mechanics of thresholdizing the idealized IBE key manager across a committee to Appendix A.2, as the details are standard and not central to our contributions. We give high-level formalization and proof intuition in §6, evaluate Chorus in §7, discuss related work in §8 and practical considerations in §9, and finally, conclude in §10. Formal definitions and security proofs for the secret recovery scheme and the key-manager scheme are deferred to Appendix A.6 and Appendix A.7, respectively.

4. Chorus Secret Recovery Scheme

In this section, we describe our password-based secret recovery scheme. For modularity (§3.3), we assume access to an idealized IBE key manager, $\mathcal{F}^{\mathsf{KM}}$, which manages decryption keys for user backups. Our scheme leverages the Boneh–Franklin IBE (BF-IBE) scheme [33] to minimize both client state and interaction, and is the first to satisfy our performance goals (§2.5) when $\mathcal{F}^{\mathsf{KM}}$ is instantiated with dynamic client committees (§5). We begin with background on BF-IBE (§4.1) and then present our construction.

4.1. Background: Boneh-Franklin IBE

An IBE scheme is a kind of public-key encryption scheme that allows encrypting a message to an arbitrary public identity or string id, and then decrypting it using an id-specific decryption key. In this section, we give background on BF-IBE scheme [33] components that are relevant to our construction. BF-IBE is defined over pairing-friendly groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ of prime order q that admit a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Let $g \in \mathbb{G}_1$ be the generator for $\mathbb{G}_1$, $H_{ID} : \{0,1\}^* \to \mathbb{G}_2$ a hash function (modeled as a random oracle), and $\mathcal{M}$ the message space. The scheme's public parameters are $\mathsf{pp} = \langle \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, \hat{e}, g, H_{ID}, \mathcal{M}, \ldots \rangle$ (ommitting irrelevant parameters). BF-IBE consists of the algorithms $\mathsf{IBE} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Extract}, \mathsf{Decrypt})$:

- $(\mathsf{pk}, \mathsf{msk}) \leftarrow_{\$} \mathsf{KeyGen}(\mathsf{pp})$: Sample msk randomly from $\mathbb{F}_q$, set $\mathsf{pk} \leftarrow g^{\mathsf{msk}} \in \mathbb{G}_1$, and output $(\mathsf{pk}, \mathsf{msk})$.
- $\mathsf{ct}_{\mathsf{id}} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathsf{id}, \mathsf{msg})$: Given pk, encrypt $\mathsf{msg} \in \mathcal{M}$ to an arbitrary identity $\mathsf{id} \in \{0,1\}^*$ to get a ciphertext $\mathsf{ct}_{\mathsf{id}}$.
- $d_{\mathsf{id}} \leftarrow \mathsf{Extract}(\mathsf{msk}, \mathsf{id})$: Compute the decryption key for id as follows: $d_{\mathsf{id}} \leftarrow H_{ID}(\mathsf{id})^{\mathsf{msk}} \in \mathbb{G}_2$.
- $\mathsf{msg} \leftarrow \mathsf{Decrypt}(d_{\mathsf{id}}, \mathsf{ct}_{\mathsf{id}})$: Given d_{id}, decrypt $\mathsf{ct}_{\mathsf{id}}$ to retrieve the message $\mathsf{msg} \in \mathcal{M}$.

A BF-IBE variant [125] provides *anon-IND-ID-CCA* privacy (see Appendix A.4 for details) —informally, an IBE ciphertext for id does not reveal any information about id or the underlying message to a chosen ciphertext adversary, even given access to the decryption keys for all other identities. Traditionally, a single private key generator runs both KeyGen and $\mathsf{Extract}$, but as these operations are simple in BF-IBE, this scheme is easy to thresholdize [95] such that the private key generator role is split across a committee.

4.2. Construction

4.2.1. KeyGen. In our construction, the idealized key manager $\mathcal{F}^{\mathsf{KM}}$ plays the role of the BF-IBE private key generator (Appendix A.2 shows how to split $\mathcal{F}^{\mathsf{KM}}$ across client committees). At the start of the protocol, server $\mathcal{S}$ instructs $\mathcal{F}^{\mathsf{KM}}$ to run the BF-IBE keygen algorithm, keep msk private, and return pk to $\mathcal{S}$. After this point, any party can retrieve pk from $\mathcal{F}^{\mathsf{KM}}$ if $\mathcal{S}$ allows it, and the users can perform backups.

4.2.2. Backup. To back up a secret message $\mathsf{msg} \in \{0,1\}^*$, a client $\mathcal{P}_{\mathsf{uid}}$ with unique identity $\mathsf{uid} \in \mathbb{U}$ (identity space) and password $\mathsf{pwd} \in \mathbb{P}$ (password space) does the following:

- Retrieve the public key pk from $\mathcal{F}^{\mathsf{KM}}$. Sample a random authenticated encryption key $k \in \mathcal{M}$ and encrypt k to identity $\mathsf{id} = \mathsf{uid} \| \mathsf{pwd}$ to get $\mathsf{ct}_{\mathsf{ibe}}$.
- Encrypt msg with k to get $\mathsf{ct}_{\mathsf{ae}}$ and output backup as $\mathsf{bkp}_{\mathsf{uid}} = (\mathsf{ct}_{\mathsf{ibe}}, \mathsf{ct}_{\mathsf{ae}})$.

$\mathcal{P}_{\mathsf{uid}}$ then uploads the backup ciphertext to $\mathcal{S}$. Since the IBE scheme is anonymous, the backup does not reveal anything about id and, therefore, the password. Note that this scheme satisfies the *non-interactive backup* requirement RQ 4.

4.2.3. Recovery. The high-level idea of the recovery is straightforward: given password pwd, client $\mathcal{P}_{\mathsf{uid}}$ can request $\mathcal{F}^{\mathsf{KM}}$ for the decryption key for $\mathsf{id} = \mathsf{uid} \| \mathsf{pwd}$, get $\mathsf{bkp}_{\mathsf{uid}}$ from $\mathcal{S}$, and then decrypt the backup to recover its secret msg. Guess-limiting to a configurable L guesses is also straightforward; $\mathcal{F}^{\mathsf{KM}}$ can keep track of recovery requests for each uid, and refuse requests after L attempts.

While this approach suffices for an idealized key manager, we require two additional properties from the recovery protocol to support client committees:

- The recovery request cannot directly reveal $\mathsf{id} = \mathsf{uid} | \mathsf{pwd}$ to the key manager, as this would expose the password to the committee. We therefore need a mechanism to *blind* the recovery request so that it hides the password, yet remains *partially-blinded* to allow guess-limiting based on uid. This should not come at the expense of additional interaction as we want to preserve *non-interactive committee* (RQ 1) and *single-round recovery* (RQ 3).
- The committee cannot maintain per-user recovery attempts, as this would grow linearly with the number of users and violate the *constant committee state* requirement (RQ 2). We therefore need a mechanism to securely offload the storage of recovery attempts.

Partially blinded recovery requests. To make the BF-IBE extraction partially blinding, we leverage its well-known connection to BLS signatures [34]. In BF-IBE, the decryption key for an identity id is computed as $d_{\mathsf{id}} = H_{ID}(\mathsf{id})^{\mathsf{msk}}$, which is algebraically identical to a BLS signature on id

under signing key msk. This equivalence means that the BF-IBE extraction can be viewed as a "signing" operation on id, and we can adapt techniques from blind BLS signatures [32] so that the identity remains hidden during extraction. Although a partially blind BLS scheme [155] more closely matches our goal, it requires interaction among signers when thresholdized, violating our *non-interactive committee* requirement (RQ 1). We therefore begin with the fully blind variant and then extend it to achieve partial blindness without introducing additional interaction.

To generate a fully blinded recovery request for $\mathsf{id} = \mathsf{uid}\|\mathsf{pwd}$, the client samples a random blinding factor $r \in \mathbb{F}_q$, computes $\mathsf{rqst} \leftarrow (H_{ID}(\mathsf{id}))^r$, which hides id, and sends it to $\mathcal{F}^{\mathsf{KM}}$ instead of id. Given rqst, $\mathcal{F}^{\mathsf{KM}}$ responds with $\mathsf{resp} \leftarrow \mathsf{rqst}^{\mathsf{msk}}$, and the client can unblind the response to get the decryption key $d_{\mathsf{id}} \leftarrow \mathsf{resp}^{-r}$. This process mirrors vanilla BF-IBE key extraction, allowing each committee member to respond *non-interactively* with a single message when $\mathcal{F}^{\mathsf{KM}}$ is instantiated with client committees.

To make these requests *partially* blinding, we augment them with a zkSNARK [121]. A zkSNARK allows a prover to non-interactively produce a short proof that a computation on secret inputs was performed correctly without revealing the inputs themselves. The client now generates a proof π^{rqst} that shows that rqst is tied to a specific uid without revealing the password pwd or the blind r. Specifically, the user proves the following relation with public inputs uid and rqst: $\mathcal{R}_{\mathsf{blind-extract}} = \{(\mathsf{pwd}, r) \mid \mathsf{rqst} = H_{ID}(\mathsf{uid}\|\mathsf{pwd})^r\}$

This proof can be generated on a mobile phone within 16 seconds (§7.4). Note that proving H_{ID} was correctly computed within zkSNARK requires white-box access to a random oracle, and we argue security in the standard model using the random oracle heuristic like the prior works on recursive proof composition [37], [40], [41], [102].

Constant committee state. To keep the committee's state size independent of the number of users, we build on authenticated dictionaries [120], [141]. The server $\mathcal{S}$ maintains the full dictionary of recovery attempts, while $\mathcal{F}^{\mathsf{KM}}$ stores only a short commitment to it. We instantiate this using the Sparse Merkle Tree with open addressing from VeRSA [144]. Each time the server requests recovery on a user's behalf, it also provides a new state commitment $\mathsf{st}_{\mathsf{new}}$ and a proof that shows that the new state correctly accounts for the recovery request. If $\mathsf{st}_{\mathsf{new}}$ does not violate the guess limit, $\mathcal{F}^{\mathsf{KM}}$ updates its public state to $\mathsf{st}_{\mathsf{new}}$ and responds to the request. As a result, $\mathcal{F}^{\mathsf{KM}}$ stores only msk and a constant-sized st, making its storage independent of the total number of users.

Final recovery predicate. We now describe the predicate ψ_L for guess limit L that $\mathcal{F}^{\mathsf{KM}}$ checks before answering recovery queries. ψ_L takes as input $(\mathsf{rqst}, \mathsf{st}_{\mathsf{new}}, \mathsf{st}_{\mathsf{old}}, \pi^{\mathsf{st}})$ and validates the following:

- Parse $\pi^{\mathsf{st}} = (\mathsf{uid}, \pi^{\mathsf{rqst}}, \pi^{\mathsf{mt}})$ and verify zkSNARK proof π^{rqst} is valid on input $(\mathsf{uid}, \mathsf{rqst})$.
- Verify Merkle proof π^{mt} to check that $\mathsf{st}_{\mathsf{new}}$ only increments uid's counter in $\mathsf{st}_{\mathsf{old}}$, and the updated counter $\leq L$.

Security. We formally describe the $\mathcal{F}^{\mathsf{KM}}$ functionality in Figure 2 and our secret recovery scheme SR_L in the $\mathcal{F}^{\mathsf{KM}}$-hybrid model in Figure 3. We defer the definitions and the proofs to Appendix A.6.

Ideal functionality $\mathcal{F}^{\mathsf{KM}}$ for Key Manager

This functionality is initialized with parameters of the IBE scheme pp and a guess-limiting predicate ψ_L (see §4.2.3) with initial state st_0. $\mathcal{S}$ denotes the server, $\mathcal{A}$ the adversary, and $\mathcal{P}$ could be any party. Corrupted $\mathcal{S}$ can send $\langle\mathsf{Terminate}\rangle$ at any point after which functionality does not answer any recovery request.

- **KeyGen**: On the first input $\langle\mathsf{KeyGen}\rangle$ from $\mathcal{S}$, set $\mathsf{st} \leftarrow \mathsf{st}_0$ and run $(\mathsf{msk}, \mathsf{pk}) \leftarrow \mathsf{IBE.KeyGen}(\mathsf{pp})$. Send $\langle\mathsf{KeyGen}, \mathsf{pk}\rangle$ to $\mathcal{A}$ and wait for its response. Then send $\mathsf{pk} \in \mathbb{G}_1$ to $\mathcal{S}$.
- **Get Public Key**: On input $\langle\mathsf{PublicKey}\rangle$ from any party $\mathcal{P}$ after the setup phase, send retriever's identity to $\mathcal{A}$ and wait for its response. If $\mathcal{S}$ is corrupted and $\mathcal{A}$ sends $\perp$, return $\perp$ to $\mathcal{P}$. Else, return pk to $\mathcal{P}$.
- **Recovery**: On input $\langle\mathsf{Recover}, \mathsf{rqst}, \mathsf{st}_{\mathsf{new}}, \pi^{\mathsf{st}}\rangle$ from $\mathcal{S}$ after setup, if $\psi_L(\mathsf{rqst}, \mathsf{st}_{\mathsf{new}}, \mathsf{st}, \pi^{\mathsf{st}}) = 1$, update $\mathsf{st} \leftarrow \mathsf{st}_{\mathsf{new}}$ and set $\mathsf{resp} \leftarrow \mathsf{rqst}^{\mathsf{msk}} \in \mathbb{G}_2$. Send $\langle\mathsf{Recover}, \mathsf{rqst}, \mathsf{st}_{\mathsf{new}}, \pi^{\mathsf{st}}, \mathsf{resp}\rangle$ to $\mathcal{A}$ and wait for its response. Then send resp to $\mathcal{S}$.

Figure 2. Ideal functionality for Key Manager.

5. Chorus Evolving Committee Proactive Secret Sharing Scheme

We now describe our evolving committee proactive secret sharing (ECPSS) scheme in the client-server setting. In this section, we focus on our new techniques and defer the complete ECPSS description to Appendix A.5. In particular, we describe the committee selection in §5.3 and the committee handover in §5.4. Before we discuss our techniques, we provide background on ECPSS in §5.1 and an overview of client ECPSS in §5.2.

5.1. Background: ECPSS

An Evolving Committee Proactive Secret Sharing (ECPSS) scheme [27] is a t-out-of-n secret sharing scheme where the committee holding the secret changes over time. The changing committees serve two purposes: (i) it *divides the workload* of a committee member across all the parties in the system, and (ii) it protects against the threat of a *mobile adversary* [123]. ECPSS is parameterized by the security parameter λ, a universe of N parties, committee size $n \ll N$, threshold $t < n$, and has the following procedures:

- $\mathsf{CommitteeSelection}()$: selects a committee $\mathcal{C}$ of (expected) size $n \ll N$ such that $< t$ members are compromised and $\geq t$ members are honest and online.
- $\mathsf{Share}(s, \mathcal{C})$: secret share s to committee $\mathcal{C}$.
- $\mathsf{Reconstruct}(\mathcal{C})$: given t or more shares from committee $\mathcal{C}$ members, outputs the secret s (or $\perp$ in case of failure).

Our Secret Recovery Scheme SR_L in $\mathcal{F}^{\mathsf{KM}}$-hybrid

The scheme is parameterized by password guess limit L, and it relies on authenticated encryption AE, the BF-IBE scheme IBE (§4.1), the hash function H_{ID} from public parameters pp of BF-IBE, a zkSNARK [30], and $\mathcal{F}^{\mathsf{KM}}$ ideal functionality (initialized with pp and ψ_L from §4.2.3). Let $\mathcal{S}$ denote the server, and $\mathcal{P}_{\mathsf{uid}}$ denote the client with identifier uid.

KeyGen():

- $\mathcal{S}$ sends $\langle \mathsf{KeyGen} \rangle$ to $\mathcal{F}^{\mathsf{KM}}$ and gets pk back.

Backup(uid, pwd, msg):

- After the setup is complete, $\mathcal{P}_{\mathsf{uid}}$ sends $\langle \mathsf{PublicKey} \rangle$ to $\mathcal{F}^{\mathsf{KM}}$ and gets public key pk of the IBE scheme.
- $\mathcal{P}_{\mathsf{uid}}$ samples $k \leftarrow \mathsf{AE.Gen}(1^\lambda)$, encrypts k using IBE: $\mathsf{ct}_{\mathsf{ibe}} \leftarrow \mathsf{IBE.Encrypt}(\mathsf{pk}, \mathsf{uid} \| \mathsf{pwd}, k)$, and encrypts msg using key k: $\mathsf{ct}_{\mathsf{ae}} \leftarrow \mathsf{AE.Encrypt}(k, \mathsf{msg})$.
- $\mathcal{P}_{\mathsf{uid}}$ sends $\mathsf{bkp}_{\mathsf{uid}} = (\mathsf{ct}_{\mathsf{ibe}}, \mathsf{ct}_{\mathsf{ae}})$ to $\mathcal{S}$, which stores it.

Recover(uid, pwd):

- $\mathcal{P}_{\mathsf{uid}}$ samples blind $r \in \mathbb{F}_q$ and creates a (blinded) IBE extraction request: $\mathsf{rqst} \leftarrow H_{ID}(\mathsf{uid} \| \mathsf{pwd})^r$. Additionally, it generates a zkSNARK proof π^{rqst} for relation $\mathcal{R}_{\mathsf{blind-extract}}$ (§4.2.3). $\mathcal{P}_{\mathsf{uid}}$ sends $(\mathsf{rqst}, \pi^{\mathsf{rqst}})$ to $\mathcal{S}$.
- If $\mathcal{P}_{\mathsf{uid}}$ has already made L recovery requests, reject the request and return $\perp$. Otherwise, let $\mathsf{st}_{\mathsf{new}}$ denote the new log state after including the recovery request for uid and π^{mt} the corresponding update proof. Define state transition proof $\pi^{\mathsf{st}} = (\mathsf{uid}, \pi^{\mathsf{rqst}}, \pi^{\mathsf{mt}})$.
- $\mathcal{S}$ sends $\langle \mathsf{Recover}, \mathsf{rqst}, \mathsf{st}_{\mathsf{new}}, \pi^{\mathsf{st}} \rangle$ to $\mathcal{F}^{\mathsf{KM}}$, and gets response resp in return. $\mathcal{S}$ forwards resp to $\mathcal{P}_{\mathsf{uid}}$ along with the corresponding backup $\mathsf{bkp}_{\mathsf{uid}}$.
- $\mathcal{P}_{\mathsf{uid}}$ parses $\mathsf{bkp}_{\mathsf{uid}} = (\mathsf{ct}_{\mathsf{ibe}}, \mathsf{ct}_{\mathsf{ae}})$, unblinds the response to get decryption key $d \leftarrow \mathsf{resp}^{-r}$, decrypts $\mathsf{ct}_{\mathsf{ibe}}$ to get the AE key $k \leftarrow \mathsf{IBE.Decrypt}(d, \mathsf{ct}_{\mathsf{ibe}})$, and finally decrypts $\mathsf{ct}_{\mathsf{ae}}$ to get the message $\mathsf{msg} \leftarrow \mathsf{AE.Decrypt}(k, \mathsf{ct}_{\mathsf{ae}})$.

Figure 3. Our Secret Recovery Scheme.

- $\mathsf{Handover}(\mathcal{C}_{\mathsf{prev}}, \mathcal{C}_{\mathsf{curr}}, \mathcal{C}_{\mathsf{next}})$: committee $\mathcal{C}_{\mathsf{curr}}$ members reconstruct their shares of s from the handover generated by $\mathcal{C}_{\mathsf{prev}}$, and re-share them to committee $\mathcal{C}_{\mathsf{next}}$.

An ECPSS scheme is *scalable* if committee selection and handover costs only depend on n and λ and are independent of total parties N.

5.2. Overview of Client ECPSS

In this section, we give a high-level overview of client ECPSS, which adapts existing ECPSS techniques to the client-server setting.

Client key setup and rotation. Each client in the system has a unique index uid and three keypairs: a signature keypair $(\mathsf{pk}^{\mathsf{sig}}, \mathsf{sk}^{\mathsf{sig}})$, a verifiable random function (VRF) [78] keypair $(\mathsf{pk}^{\mathsf{vrf}}, \mathsf{sk}^{\mathsf{vrf}})$, and an ephemeral encryption keypair $(\mathsf{pk}^{\mathsf{enc}}, \mathsf{sk}^{\mathsf{enc}})$. Before joining the ECPSS protocol, a client registers its signature verification key $\mathsf{pk}^{\mathsf{sig}}$ with the PKI (Figure 11). It then generates fresh VRF and encryption keypairs, signs their corresponding public keys $(\mathsf{pk}^{\mathsf{vrf}}, \mathsf{pk}^{\mathsf{enc}})$ with $\mathsf{sk}^{\mathsf{sig}}$, and sends the attested tuple $\mathsf{ckeys} = (\mathsf{pk}^{\mathsf{sig}}, \mathsf{pk}^{\mathsf{vrf}}, \mathsf{pk}^{\mathsf{enc}})$ to the server. The signature and VRF keys are long-lived, while the encryption keys are ephemeral (see selection and handover below). To rotate its encryption key, the client samples a fresh encryption keypair, attests it with its signing key, and informs the server of the new encryption key.

Committee selection. Committee selection uses the clients' VRF keys to select an n-sized committee uniformly at random from the population of N clients in a verifiable manner (§5.3). In particular, clients maintain local epoch counters, which are loosely synchronized with the server's epoch counter. In epoch e, clients use $\mathsf{sk}^{\mathsf{vrf}}$ to locally check whether they are selected for committee $\mathcal{C}_e$ and reveal participation to the server along with a membership proof if selected. Once a client announces its membership, it starts a timer to rotate its encryption key after δ time unless a completed handover does so first. The server processes the membership proofs at the end of the epoch and publishes $\mathcal{C}_e$ to the transparency log, allowing any client to verify the committee and retrieve the signature and encryption public keys $(\mathsf{pk}^{\mathsf{sig}}, \mathsf{pk}^{\mathsf{enc}})$ of its members.

Committee handover. Committee handover lets $\mathcal{C}_{\mathsf{curr}}$-members receive the secret state from $\mathcal{C}_{\mathsf{prev}}$ and transfer it to $\mathcal{C}_{\mathsf{next}}$ without interacting with other members. A $\mathcal{C}_{\mathsf{curr}}$ member first uses $\mathcal{C}_{\mathsf{prev}}$'s signature keys $\mathsf{pk}^{\mathsf{sig}}$ to ensure the received handover came from $\mathcal{C}_{\mathsf{prev}}$ and then verifies that it is well-formed and consistent with the prior handover. It then reconstructs its current share and re-shares it to $\mathcal{C}_{\mathsf{next}}$ using $\mathcal{C}_{\mathsf{next}}$'s encryption public keys $\mathsf{pk}^{\mathsf{enc}}$. The server processes these handovers, weeds out malformed ones, and publishes the resulting handover to the transparency log. Handover is pipelined across epochs. In epoch e, committee $\mathcal{C}_{e-2}$ processes the state it received from $\mathcal{C}_{e-3}$ and re-shares it to committee $\mathcal{C}_{e-1}$. As $\mathcal{C}_{e-1}$ was already committed in epoch $e-1$, the senders know the recipients and their public keys before transfer begins. After completing handover, or upon timeout if transfer does not complete, committee members rotate their encryption keys and erase their secret shares.

Long-term security. ECPSS provides long-term security by proactively refreshing shares every epoch, so compromised shares become useless after refresh. The ECPSS-managed secret state is the key-manager secret msk, which is initially set up via a distributed key generation (DKG) protocol (Appendix A.2). To rotate msk, the system can run a fresh DKG and have users re-back up under the new public key.

5.3. Committee Selection

Background: VRF-based committee selection. Prior work uses cryptographic sortition [75], [92] to select committees: in each epoch, parties learn a seed and use a verifiable

random function (VRF) [78] to locally check whether they are selected and produce a publicly verifiable membership proof. If the VRF keys are independent of the seed, this mechanism provides the following (informal) properties:

- **Unpredictability**: Committee membership of parties cannot be inferred from the seed unless they reveal it.
- **Fairness**: The seed and the VRF ensure that the n-sized committee is chosen randomly from the N-party universe.

In blockchain settings [75], seed independence is achieved by having the entire network agree on a fresh random seed each epoch, which is impractical for our setting with hundreds of millions of clients. Lotto [92] instead focuses on the client-server setting and proposes a solution that ensures seed independence without global coordination by assuming that the PKI honestly samples each client's VRF keys during registration. Although Lotto is practical for our setting, it introduces additional trust assumptions on the PKI and changes the way they operate.

Our scheme. We build on Lotto and preserve its efficiency, but eliminate its requirement that the PKI generate VRF keys. Instead, we let clients sample their own keys, and the PKI only performs verification during registration. To enable this, we introduce a new seed-selection scheme that imposes no restrictions on how clients sample their VRF keys.

Key-anchored seed selection. We want the seed selection to satisfy the following properties: (i) the VRF keys of all the clients are independent of the seed, (ii) the seed itself is uniformly random, and (iii) the clients can efficiently verify the selection. Given these properties, an unbiasable VRF [76] guarantees a verifiable sortition that is indistinguishable from a random lottery. Our approach is to define the seed as a function of all clients' VRF keys such that any attempts to manipulate a VRF key to bias the sortition would result in a random change to the seed. Although computing this function requires $O(N)$ work, the server can perform the computation and distribute the seed, and the clients can verify it in sublinear time.

This function has a simple instantiation: the server creates a Merkle tree-based authenticated dictionary [144] that maps each client's unique identity uid to its VRF public key $\mathsf{pk}^{\mathsf{vrf}}$, and then hashes the root rt of the Merkle tree along with the epoch counter e to get $\mathsf{seed}_e = H_{\mathsf{seed}}(\mathsf{rt}, e)$. The server then sends rt to all participating clients, and when publishing a committee's data to the transparency log, it also includes rt and the Merkle proofs for the selected clients' VRF keys. This construction satisfies the three required properties. First, seed_e is anchored to the VRF keys, forcing the keys to be chosen independently. Second, modeling H_{seed} as a random oracle makes seed_e uniformly random. Third, verification is efficient: clients only need to check that each committee member's Merkle proof is consistent with the same root rt from which seed_e was derived, and that the VRF keys are distinct. A content distribution network (CDN) can distribute the root rt initially and after periodic (e.g., weekly) updates to the client list to manage churn. We prove security of the seed selection in the random oracle model (Lemma 11), where we also model the Merkle tree hash function as a random oracle.

Parameter Selection. We defer the ECPSS parameter analysis to Appendix A.1; Table 10 summarizes the parameters.

5.4. Committee Handover

Background: NIVSS-based committee handover. In a (t, n, N)-ECPSS scheme, each committee maintains a (t, n) Shamir sharing [136] of the private state, meaning the members' shares lie on a degree-t polynomial. During handover, each member of the current commmittee $\mathcal{C}_{\mathsf{curr}}$ re-shares its share to the next committee $\mathcal{C}_{\mathsf{next}}$; if at least t members do so correctly, $\mathcal{C}_{\mathsf{next}}$ can combine the received shares to obtain a valid (t, n) sharing of the same state. Since malicious clients may send malformed dealings, the handover requires a mechanism to detect and reject invalid ones.

Prior work [27] achieves this while preserving non-interactive committees (RQ 1) by using a non-interactive verifiable secret sharing (NIVSS) scheme [46], [73], [96] with public verifiability, which we denote as pVSS. With a pVSS scheme, each $\mathcal{C}_{\mathsf{curr}}$ member independently produces a (t, n) dealing of its share as follows: it re-shares the value it holds using (t, n) Shamir secret sharing, encrypts the new shares under the public keys of $\mathcal{C}_{\mathsf{next}}$ members, and attaches a proof of correct sharing showing that these ciphertexts are well-formed and correspond to a degree-t polynomial. This proof also includes a public commitment to the dealer's share, which links successive handovers and ensures that the state transferred to $\mathcal{C}_{\mathsf{next}}$ is the one received from $\mathcal{C}_{\mathsf{prev}}$. Finally, the dealer publishes the proof and ciphertexts to a broadcast channel (a transparency log in our setting). Overall, pVSS offers the following (informal) properties:

- **Privacy**: Shares of honest receivers are not revealed.
- **Uniqueness**: Each dealing commits to a unique degree-t polynomial, allowing honest receivers to verify that their shares are valid w.r.t. the same polynomial and that the shared secret (dealer's share) is consistent with the polynomials received from $\mathcal{C}_{\mathsf{prev}}$.
- **Public verifiability**: Without learning the shares, anyone can verify that a dealing's ciphertexts are well-formed and their plaintexts constitute a valid (t, n) sharing.

Privacy and uniqueness are critical for protecting user secrets, while public verifiability ensures availability by allowing honest receivers to independently reject invalid dealings.

Impractical overheads of pVSS. Handover between n-sized committees $\mathcal{C}_{\mathsf{curr}}$ and $\mathcal{C}_{\mathsf{next}}$ requires each $\mathcal{C}_{\mathsf{curr}}$ member to generate one pVSS dealing, while every $\mathcal{C}_{\mathsf{next}}$ member must verify $O(n)$ such dealings before reconstructing its share. Since each dealing costs $O(n)$ communication and computation to generate and verify, $\mathcal{C}_{\mathsf{curr}}$ members incur $O(n)$ overhead, but the $\mathcal{C}_{\mathsf{next}}$ members face $O(n^2)$ overhead. Concretely, for $n \approx 1000$ and state-of-the-art pVSS [96], the dealer spends 5.5 s of compute and uploads ~ 550 KB of data, while the receiver downloads 601 MB of data and spends 318 s of compute on a mobile phone (§7.3).

If we apply this ECPSS scheme to our setting, the server needs to send ~ 655 GB of data to the clients to relay these dealings every epoch, which would cost the server ~ 1.2M$ per month assuming two-minute epochs on average.

Our scheme saVSS**.** To this end, we propose saVSS, a server-aided NIVSS scheme that reduces handover overhead by offloading computation to the server. Our key insight is that we can significantly reduce both client and server overhead by making the clients responsible for only protecting user secrets, while delegating availability-related computation (and guarantees) to the server. By doing so, we take advantage of our setting where we already have a powerful server that is hosting the application and orchestrating the communication among millions of clients. Thus, delegating availability to the server does not add a new trust assumption to the system; it simply makes an existing one explicit. Given this insight, we only need (1) a weaker notion of NIVSS for clients that only protects privacy, and (2) a server mechanism to preserve availability against malicious clients.

Minimizing client overhead through a weaker notion of NIVSS. Recall from the pVSS background that clients only need privacy and uniqueness to protect user secrets. We observe that both of these properties are supported by the weaker notion of NIVSS with abort [52] (henceforth called aVSS). In aVSS, instead of sending ciphertexts and a proof of their well-formedness, the dealer broadcasts a commitment to the sharing polynomial and sends the individual shares to the receivers through point-to-point channels. A corrupted dealer can cause an abort by sending an incorrect share that does not match the commitment, but this still suffices for our purposes: honest parties receive their shares through a secure channel (privacy), and the dealer commits a unique degree-t polynomial to the log (uniqueness).

Using aVSS for clients drastically reduces their verification cost. For $n \approx 1000$, the state-of-the-art pVSS scheme [96] requires $36\times$ more communication and over $690\times$ more verification time than the best-known aVSS construction [52], even though their asymptotic complexity is roughly the same. The gap arises because pVSS requires receivers to verify a global statement over all $O(n)$ encrypted shares, forcing the proof to operate over ciphertexts and incur substantial encryption overhead. In contrast, each receiver in aVSS only validates its *own* share against the commitment; receivers decrypt their shares before verification, so the proof avoids the encryption overhead entirely.

Preserving availability through a hybrid NIVSS scheme. aVSS ensures privacy, but we also need to defend against malicious clients who want to deny availability by sending malformed dealings. To allow the server to detect invalid dealings with low client overhead, we propose a hybrid NIVSS scheme called server-aided NIVSS (henceforth called saVSS) where the dealer generates both pVSS and aVSS dealings. The server verifies both dealings to make sure they are well-formed, and the clients only verify the aVSS dealings. For this to work though, the clients also have to prove that the dealings are consistent and correspond to the same sharing. Fortunately, we can get this for free

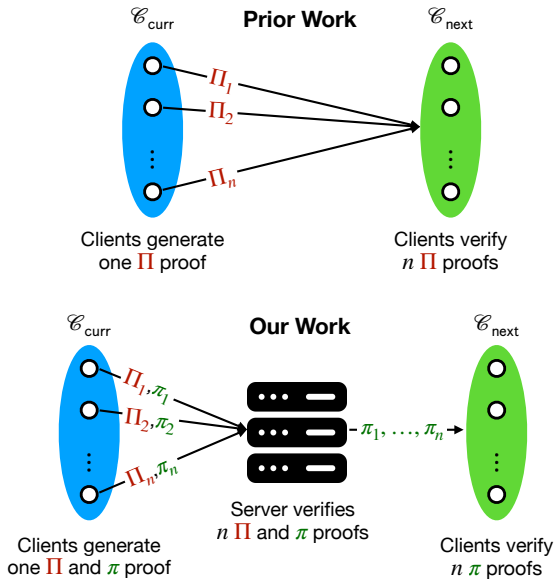


Figure 4. ECPSS handover between n-sized committees $\mathcal{C}_{\text{curr}}$ and $\mathcal{C}_{\text{next}}$ with our server-aided NIVSS vs pVSS. Π denotes a pVSS proof, which is much more expensive to generate and verify than the aVSS proof π.

because pVSS schemes support outputting a commitment to the sharing polynomial, and generating both dealings for the same commitment ensures consistency between them. Finally, instead of the dealer sending shares via point-to-point channels, the server sends the corresponding ciphertexts from the pVSS dealings in saVSS.

With saVSS, $\mathcal{C}_{\text{next}}$ incurs the $O(n^2)$ receiver overhead of aVSS and an $O(n)$ cost of decrypting pVSS ciphertexts. The $\mathcal{C}_{\text{curr}}$ overhead is higher because we generate both pVSS and aVSS dealings, but this is acceptable as dealing is not a bottleneck, and the added overheads are small (§7.3). The server now needs to verify pVSS dealings instead of the clients—this tradeoff is desirable as the server has more compute resources. Overall, saVSS improves the client handover time and communication by $147\times$ and $111\times$ (§7.3). The server's dollar cost is also improved by $100\times$. We compare saVSS and pVSS qualitatively in Figure 4 and describe saVSS formally in Figure 13 assuming access to a transparency log $\mathcal{F}^{\text{tlog}}$ (Figure 12).

6. Formalization and Security Intuition

We prove security modularly. First, we show that the client-committee-based key-manager protocol KM (Appendix A.2) securely realizes the idealized key manager $\mathcal{F}^{\text{KM}}$ (Figure 2). Then, assuming $\mathcal{F}^{\text{KM}}$, we prove privacy and availability of the secret-recovery scheme §4.2.3. Standard composition [110] gives end-to-end security for Chorus.

Adversary modeling. We consider an active, mobile adversary that may corrupt the server and at most a κ fraction of clients (§2.4). Client corruptions take effect after δ time and reveal the full local state. The adversary has full control over all corrupted parties. Through the corrupted server, it

can arbitrarily delay or drop messages. It can also control the network, provided the network assumption (§2.3) holds for availability: messages exchanged during epochs e to $e+2$ among the online honest parties (the server, ideal functionality, and $1-\phi$ fraction of honest clients) are delivered within δ time from the start of epoch e. The adversary specifies honest-party inputs and observes outputs.

We state informal theorem statements and the proof intuitions below. The formal key-manager security statement and full proof appear in Appendix A.7. The formal secret-recovery privacy and availability statements and proofs appear in Appendix A.6.1 and Appendix A.6.2, respectively.

Theorem 1 (informal key-manager security)**.** *Assuming access to the transparency-log functionality $\mathcal{F}^{\mathsf{tlog}}$ (Figure 12) and the PKI functionality $\mathcal{F}^{\mathsf{PKI}}$ (Figure 11), the key-manager protocol KM (Appendix A.2) securely realizes the ideal key manager $\mathcal{F}^{\mathsf{KM}}$ (Figure 2).*

Proof Intuition. We first argue that committee selection establishes unbiasable committees with $< t$ corrupted clients and $\geq t$ online honest clients, and that all honest clients agree on these committees. Then we show that these committees exactly emulate the ideal key-manager functionality.

Unbiasable committees. The authenticated dictionary mapping uid (from PKI) to a VRF public key enforces that each registered client contributes at most one VRF key to the sortition. We then reduce committee-selection fairness to unbiasable-VRF security [76] (see Lemma 11), which only requires that the global seed is uniform, independent of the selected VRF keys, and that the selected VRF keys are distinct. These conditions hold because our seed-selection mechanism enforces both uniformity (from the random oracle) and independence (from the Merkle tree over the VRF keys dictionary), while honest clients accept a committee only if the selected VRF keys are distinct. Even if a malicious server tries multiple dictionary configurations to get different seeds, Appendix A.1 bounds the probability of selecting $\geq t$ corrupted members by 2^{-128}. This holds against a mobile adversary because corruptions made after selection take effect after time δ, when committee members have already deleted their secret state, while corruptions made before selection are already accounted for in the global κ bound. Appendix A.1 also bounds the probability of selecting $< t$ online honest clients by 2^{-80}. Finally, the non-forking property of the transparency log ensures that all honest clients agree on the committee for each epoch.

Emulating the ideal key manager. We show that these committees emulate the ideal key-manager functionality. The ideal key manager simply runs key generation, outputs pk, keeps msk private, and outputs $\mathsf{rqst}^{\mathsf{msk}}$ only if the recovery predicate is satisfied and the server behaves honestly.

First, we show that the committee DKG, handover, and recovery responses leak the same information about the secret state msk as the ideal functionality. To show this, the proof relies on aVSS and follows a standard threshold-cryptography simulation argument [52]. The aVSS proof lets the simulator extract the underlying shares used by the adversary. Given pk, the recovery responses, and the adversary-chosen shares, the distribution of honest parties' msk-related responses in DKG, handover, and recovery can be simulated identically without knowing msk.

Next, we argue that the committees enforce the same guess-limiting predicate as the key manager. Every use of msk requires $\geq t$ shares, and since each committee has $< t$ corrupted members, at least one honest committee member response is always needed. That honest member produces a recovery response only if the predicate is satisfied. The public guess-limiting state is committed to the transparency log (Appendix A.2), and its non-forking property ensures that all honest participants see the same state.

Finally, if the server behaves honestly, the committees provide the same availability guarantee as the ideal key manager. An honest server can discard malformed responses that would otherwise make an honest committee member abort; it can do so without learning any honest client shares because pVSS is publicly verifiable. Because each committee has $\geq t$ online honest members and the network assumption holds, the committees make progress and process requests.

Theorem 2 (informal secret-recovery privacy)**.** *Assuming access to the ideal key manager $\mathcal{F}^{\mathsf{KM}}$ (Figure 2), the secret-recovery scheme (§4) reveals the backed-up secret only if the adversary guesses the password within the allowed number of password guesses.*

Proof Intuition. Once we have the idealized key manager, the privacy of the secret recovery scheme is straightforward. The anon-IND-ID-CCA property of the IBE scheme (§4.1) ensures that the backup ciphertext does not reveal the password, and the only way to learn the secret underlying the backup ciphertext is to extract the decryption key for that password and uid from the ideal key manager. The ideal key manager will not provide such keys beyond the allowed guess limit, and the zkSNARK prevents malformed requests that try to recover for one uid while claiming another in order to bypass that guess limit.

Theorem 3 (informal secret-recovery availability)**.** *Assuming access to the ideal key manager $\mathcal{F}^{\mathsf{KM}}$ (Figure 2) and an honest server, an honest user who submits the correct password within the allowed number of password guesses recovers the backed-up secret even if the adversary controls all other secret recovery clients in the system.*

Proof Intuition. Correctness follows from the correctness of the IBE scheme and the availability of the key manager when the server is honest.

7. Evaluation

In this section, we answer the following questions:

- (§7.3) How does our server-aided NIVSS scheme saVSS compare against the prior NIVSS schemes that do not leverage server computation?
- (§7.4) What are the overheads experienced by a Chorus client as the total clients in the system scale?
- (§7.5) What are the costs incurred by the Chorus server as the total clients in the system scale?

- (§7.6) How does Chorus compare to a secret recovery system like Signal's SVR3 in terms of dollar cost [55]?

We also benchmark the one-time costs to setup the public key of our secret recovery scheme. The costs are similar to a typical epoch in Chorus, and we defer the results to Appendix A.3.1.

7.1. Experimental Setup

We benchmark the server on a `c2d-highcpu-112` GCP instance (AMD EPYC 7B13 processor, 112 vCPUs, 224 GB RAM), and the client on an Android phone (Qualcomm Snapdragon 8 Gen 3 processor, 8 cores, 8 GB RAM, 3880 mAh battery). To simulate a global area network, we locate the server in South Asia and the phone on the US west coast. Over 10 tries, the average bandwidth between the two was 75 Mbps and the average round trip time was 280 ms. All experiments were multi-threaded and the runtimes/battery usage are averages taken over 10 runs.

We fix the protocol parameters in the evaluation to committee size $n = 1090$ and threshold $t = 300$ to target a setting where at most $\phi = 50\%$ of the clients can be offline and the attacker can compromise at most $\kappa = 10\%$ (Table 10). We also consider a setting where $\phi = 75\%$ clients can be offline and $\kappa = 1\%$ can be compromised $(n = 1214, t = 121)$; this setting is cheaper for both the server and the clients, and we defer its evaluation to Appendix A.3.2. Chorus operates asynchronously, and the exact epoch length depends on the network delays. Assuming no delays, epochs last ≈ 1 minute (see Table 7 and Table 9); for this evaluation, we conservatively set the average epoch length to 2 minutes to estimate Chorus's costs. We vary the total clients N in the system from $N = 10^6$ to $N = 10^8$ to show that Chorus scales well with a large number of clients. To estimate the recovery requests in an epoch, we assume that each client will make one recovery request per year on average since it is a rare event. The server's costs in Chorus are probabilistic and depend on the number of clients on the committee that come online and are compromised. We consider the worst-case costs throughout our experiments, i.e., the server has to communicate with all n clients to receive t valid responses. Note that even in the worst-case, the server only needs to process $2t - 1$ responses to get t valid responses because there are $< t$ compromised clients on each committee.

7.2. Implementation

We implemented Chorus with 6.6K lines of Rust code and our code is available at https://github.com/deevashwer/chorus. For the class group implementation for cgVSS, we wrote Rust bindings to the C++ library `BICYCL` [35]. We instantiated BF-IBE over the pairing-friendly `bls12-377` elliptic curve [36]. We instantiated the zkSNARK in the secret recovery scheme with Groth16 [83] over the `bw6-761` curve [89] since this curve allows efficiently proving constraints on the `bls12-377` curve. We rely on `arkworks` [18] for the curves and Groth16 implementation. Groth16 requires a circuit-specific trusted setup, and we assume that the system developer facilitates this setup through a separate setup ceremony [116] that involves crowd-sourced contributions and provides security if at least one contributor is honest. We use `ed25519` [93] for signatures, `aes-gcm` [117] for authenticated encryption, `sha256` [87] for the random oracle, and the `merkle-tree-avd` implementation from VeRSA [144] for the authenticated dictionary. Finally, we use the unbiasable [76] `ecvrf` as the VRF scheme.

To evaluate server overhead, we co-locate the server and all clients on a single machine. This setup allows us to measure server runtime and the total communication between the server and the clients. Note that this approach does not account for the server overhead of managing network connections with many clients.

To evaluate client overheads, we implement and benchmark the Chorus client on the Android phone, and the key-manager client interacts with the server over the network.

We do not implement existing components that integrate with Chorus, including the transparency log, the PKI, and the zkSNARK setup ceremony.

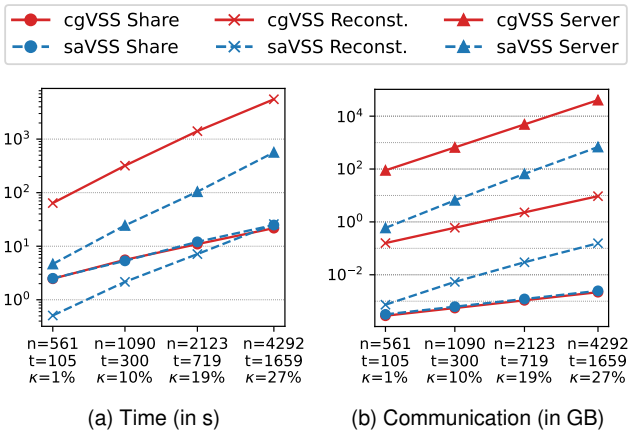


Figure 5. Our NIVSS scheme saVSS (§5.4) reduces both latency and communication overheads by two orders of magnitude compared to the state-of-the-art publicly-verifiable NIVSS scheme cgVSS [96] for an ECPSS handover between committees of size n and threshold t. κ represents the corruption fraction these parameters target and the fail fraction ϕ is set to 50%. The graphs use a log-log scale. The cgVSS server time is not pictured because the cgVSS server simply acts as a relay.

7.3. Our NIVSS Scheme vs State-of-the-art

In Figure 5, we compare our server-aided NIVSS scheme saVSS (§5.4) with the state-of-the-art publicly-verifiable NIVSS scheme cgVSS [96] in performing an ECPSS handover (§5.1) for various committee sizes n and thresholds t.

The reconstruction step during handover in cgVSS is the bottleneck for the client as it scales quadratically with n (§5.4). For $n = 1090$, cgVSS requires 318 seconds of compute (or 2.5% of battery) and 601.1 MB of communication. In saVSS, we move this overhead to the server (see Figure 5a), which can handle the communication and perform the same computation within 24 seconds.

saVSS improves the reconstruction runtime for the client by two orders of magnitude, $125\times$ for $n = 561$ and $214\times$ for $n = 4292$. We see a larger improvement for larger n because, unlike prior work, the bottleneck in saVSS is not quadratic in n (§5.4). saVSS also improves reconstruction communication for the client by $214\times$ for $n = 561$ and $60\times$ for $n = 4292$. The communication improvements decrease with increasing n because saVSS cost scales with t while cgVSS cost scales with n.

The server communication in cgVSS is also very high. For $n = 1090$, the server does 655 GB of communication per 2-minute epoch, which alone would cost the server roughly $1.2M/month. saVSS improves this communication by two orders of magnitude because the server's communication that scales with $O(n^3)$ is also much smaller (Figure 4). As such, the server communication improvements are similar to that of client's, ranging from $59\times$ to $150\times$.

Finally, Figure 5 shows that saVSS only adds minimal latency and communication overhead over cgVSS.

7.4. Client Costs

In this section, we analyze the costs incurred by a Chorus client, both as a secret-recovery client, and a key-manager client on an ECPSS committee.

7.4.1. Secret-recovery client. Table 6 summarizes the overheads for a secret-recovery client (independent of ECPSS parameters).

Backup. Backup is very cheap in Chorus: given the public key, it only takes 8 ms to generate a 196 B backup ciphertext for a 32 B secret. (To back up a larger secret, the client can back up a symmetric encryption key and then use that key to encrypt the secret.) The client uploads its backup ciphertext to the server.

Recovery. A recovery is more expensive than a backup, but is still feasible on a mobile device considering it is only needed during a rare, catastrophic event where a user loses access to all of their devices. Generating a recovery request takes about 16 s on a phone with multi-threading and involves 205.1 MB of communication. Almost all of this communication comes from downloading the public parameters for the Groth16 zkSNARK; excluding this download, the total communication is only 624 B. Since recovery is an infrequent operation, clients can also perform it over Wi-Fi. Approximately 96% of the zkSNARK cost comes from the high arithmetization cost of the map-to-curve function over SHA-256. Recent work [86] reduces this cost by two orders of magnitude by eliminating hash functions, which we estimate could lower the overall recovery time and communication by at least an order of magnitude.

Assuming an epoch length of 2 minutes, the recovery latency would be between 2-4 minutes after generating the request, depending on how soon the client committee responds and whether the client's request has to wait to get batched in the next epoch.

TABLE 6. SECRET-RECOVERY CLIENT COSTS.

Operation	Time	Comm.	Battery
Backup	8 ms	196 B	0.07 mAh (0.002%)
Recovery	15.8 s	205.1 MB	9.7 mAh (0.25%)

TABLE 7. THE OVERHEADS FOR A CLIENT SELECTED FOR A COMMITTEE GROW SLOWLY AS THE TOTAL NUMBER OF CLIENTS N INCREASES, BUT THE EXPECTED FREQUENCY WITH WHICH A CLIENT MUST PAY THIS OVERHEAD DROPS QUICKLY FOR LARGER N. WE USE A COMMITTEE SIZE $n = 1090$ AND THRESHOLD $t = 300$.

	$N = 10^6$	$N = 10^7$	$N = 10^8$
Frequency	1d 6h	12d 17h	4 mo 7 d
Battery	13.2 mAh 0.34%	13.3 mAh 0.34%	14.7 mAh 0.38%
Time	24.2 s	24.4 s	27.5 s
Comm.	12.57 MB	12.63 MB	13.20 MB

7.4.2. Key-manager client. Table 7 and Figure 8 report the cost incurred by a key-manager client on a committee, assuming each of the N clients in the system performs 1 recovery per year.

The client cost has two components: (1) the handover cost, which depends only on n and t, and (2) the cost of processing recovery requests. The cost of processing a single recovery request is independent of the total users in the system N, but the number of expected recovery requests grows with N. Figure 8 shows that the handover cost dominates the total cost in our system. Even for $N = 10^8$ where we estimate 381 recovery requests per epoch, handover accounts for $> 88\%$ of the total cost. As a result, the client overhead remains nearly the same even if we scale the system from $N = 10^6$ to $N = 10^8$. Even for $N = 5 \cdot 10^8$ (1903 requests per epoch), we expect the client overhead will only grow by $1.5\times$. Concretely, the entire client computation (including network) takes at most 28 seconds, consumes at most 0.38% battery, and requires at most 13.2 MB of communication to support up to 100M clients (Table 7).

This concrete cost is also incurred very infrequently by a Chorus client, especially for large N. For a committee of size n, the probability of getting selected on a committee each epoch is $\frac{n}{N}$, which decreases linearly with N for a fixed n. Hence, the expected on-committee frequency for a client is once every $\frac{N}{n}$ epochs, which is just once every 4 months for $N = 10^8$ (Table 7). Even for a million clients, the frequency is just once every 30 hours.

As we increase N, the overheads for a client selected on the committee increase slightly, but the probability of a client doing this work drops linearly; the expected per-client overhead in Chorus decreases as we scale to more clients.

7.5. Server Costs

We analyze the costs of deploying a Chorus server.

Backup. Recall that a backup ciphertext is 196 B, excluding the additional costs for storing a larger secret (§7.4.1). Assuming that the server is only storing 32 B secrets, for

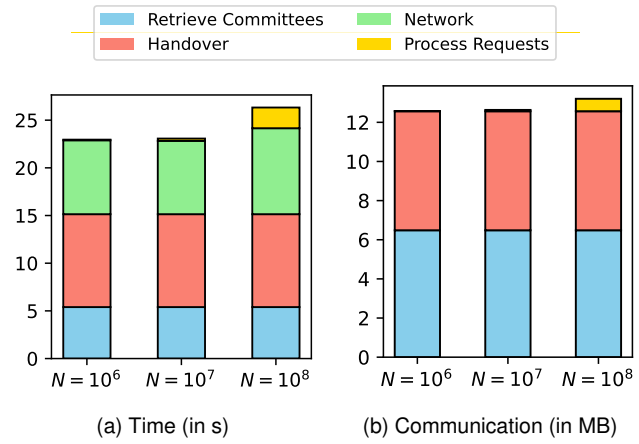


Figure 8. The majority of the costs incurred by a key-manager client are independent of the number of clients N for $N \leq 10^8$. We assume that each client makes a recovery request once a year.

$N = 10^8$, the server has 19.6 GB of ciphertexts, which costs < 1 dollar to store in an S3 bucket.

Recovery. The server's cost in facilitating recovery follows the same trend as the key-manager client (§7.4.2): (1) a handover cost that depends only on n and t, and (2) the cost of processing recovery requests, which grows with the number of requests. We report the server's runtime and communication for the fixed and recovery costs for $N = 10^6$ to $N = 10^8$ clients in Table 9.

Like the client, the server's cost is also dominated by the fixed handover cost. Concretely, the server performs under 39 seconds of computation and 14.4 GB of communication per epoch to support up to $N = 10^8$ clients. Since the server's interaction with the clients is asynchronous within an epoch (i.e., it can process each client's response independently), the server does not need to perform 39 seconds of computation after the last response is received. Instead, it can pipeline the processing with the client responses, and the final response will only take milliseconds to process. The server's communication is also just 14.4 GB, which takes under 5 seconds with a 25 Gbps bandwidth connection.

Dollar cost. We estimate the dollar cost for the server to support 100M clients. The `c2d-highcpu-112` instance costs $1930.52 per month, and given the egress communication cost of $0.085 per GB [150] (ingress is free), our dollar-cost for communication is $25.1K/month. The protocol state is stored by the clients, and the storage cost for backup is negligible. We also estimate the CDN cost of distributing a 32 B seed across 100M clients. Since the seed is small, the server only pays for the number of requests made. Assuming the seed is changed once every week, 100M clients would make around 500M requests every month, which costs roughly $600/month with Amazon CloudFront [13]. Summing up, the total dollar cost (excluding the transparency log and the PKI) comes out to be $27.6K/month.

TABLE 9. THE SERVER PER-EPOCH COSTS ARE DOMINATED BY A FIXED COST THAT IS INDEPENDENT OF THE NUMBER OF CLIENTS N, AND THE RECOVERY COST GROWS LINEARLY WITH N (ASSUMING EACH CLIENT PERFORMS ONE RECOVERY PER YEAR).

	Fixed Cost	**— Recovery Cost —**		
		$N = 10^6$	$N = 10^7$	$N = 10^8$
Time	26.8 s	0.5 s	1.6 s	12.0 s
Comm.	13.7 GB	7.4 MB	70.9 MB	0.69 GB

7.6. Comparison with Signal's SVR3

We compare the dollar cost of Chorus with SVR3 [55], a secret recovery system from Signal. We will focus our comparison on server-side costs (Chorus clients costs are higher than SVR3 client costs). Our comparison of server-side costs is not a fully apples-to-apples comparison: (1) Chorus provides cryptographic guarantees that are fundamentally different from SVR3 which relies on secure hardware, and (2) the reported costs for SVR3 include all aspects of the system, while our costs do not account for fault tolerance and exclude some orthogonal components like the transparency log and the PKI.

SVR3 reports a dollar cost of $2.1K/month for a staging deployment with 10M users, which grows to $37.6K/month for a production deployment with 500M users. In contrast, the costs of Chorus grow slowly, costing $27.6K/month for 100M users. Even with 500M users, we expect the cost to only grow to roughly $34.8K/month, due to the added server communication from recovery requests and the CDN requests. While these costs do not account for replication, server communication accounts for 85% of our cost and that does not need to be replicated. Moreover, these costs assume a worst-case scenario where the server needs to send data to all n clients on each committee to receive t valid responses; if half of the committee is online on average, then the server communication cost will also be halved.

8. Related Work

Secret recovery. Password-based primitives [11], [20], [43], [57], [61], [63], [90], [91], [104] typically underlie password-based secret recovery systems [5], [51], [55], [111], [143] and require interactive backups due to their reliance on symmetric primitives such as oblivious pseudorandom functions (OPRFs) [45]. Safetypin [60] relies on puncturable encryption and has non-interactive backups, but it was not designed for dynamic committees, and it is unclear how to transfer its private state efficiently. Most of the prior work [17], [55], [60], [81], [143], [148] on password-based recovery relies on secure hardware (see SVR3 [55] for detailed survey). Other works [51], [98], [103], [111] instead distribute trust across a few servers. Some schemes do not rely on passwords for recovery by having the user print and store a 128-bit cryptographic key for the backup [100], [106], [124] or shard it among peers or guardians [42], [49], [97], [127]. These works place manual burden on the user

during backup which can lead them to skip the backup process entirely [151]. Adei et al. [10] propose a secure hardware-based recovery scheme that requires the clients to stay online to refute illegitimate recovery attempts. Prior work has shown that end-to-end encrypted backups can be vulnerable to injection attacks where the backup length may reveal metadata [62], and so care must be taken when serializing backups.

Client-committee–based systems. Like Chorus, several works on federated learning [22], [92], [94], [109], [112] and federated analytics [131], [132], [133] also distribute trust across client committees. Both of our ECPSS techniques can benefit these systems: our committee-selection mechanism reduces PKI trust assumptions compared to the state-of-the-art Lotto [92], and our server-aided NIVSS scheme removes the need for intra-committee interaction during state transfer.

Decentralized storage. Secret recovery works allow users to recover their secrets even after losing all their cryptographic keys. A separate line of work focuses on storing user secrets on a blockchain [27], [48], [82], [128] or a decentralized committee [101], [114], [134], and assumes that clients have some cryptographically strong key to authenticate themselves while retrieving the secret.

Verifiable secret sharing. Chorus relies on verifiable secret sharing (VSS) to transfer state from one committee to the next. We specifically focus on non-interactive VSS (NIVSS) schemes for field elements [46], [73], [84], [96] in this work. Prior works have considered other VSS variants that are more efficient, namely, NIVSS for group elements [47], [58] and interactive VSS [9], [12], [28], [29], [59], [65], [71], [126], [139], [153]. The former is not suitable for Chorus as it does not compose with threshold BF-IBE, and the latter introduces intra-committee interaction which increases latency and limits client availability. Our server-aided NIVSS scheme (§5.4) avoids this interaction and our client and server handover communication costs roughly match those of NIVSS with abort [52], which are smaller than the state-of-the-art interactive VSS scheme [59].

Proactive secret sharing. A proactive secret sharing (PSS) scheme [123] allows a committee to maintain a shared secret in the presence of a mobile adversary. Dynamic PSS [82], [115], [147], [154] considers a setting where the committees change every epoch, and assumes an external mechanism to select committees. In this work, we focus on evolving committee PSS (ECPSS) [27] that incorporates committee selection as part of the protocol. Unlike Chorus, recent ECPSS works [27], [44], [74] focus on anonymous committees to defend against a strongly adaptive adversary in the "you only speak once" (YOSO) model [72]. This adversary model is challenging to achieve in our setting because the server would have to communicate with millions of clients each epoch to maintain the anonymity of the committee.

9. Discussion

PKI instantiations. The PKI can be instantiated in different ways: it could be a third-party certificate authority (CA), either centralized, such as a government agency, or based on distributed trust [114]; a blockchain-based PKI [130]; a key-transparency log [53]; or even the application server itself (assuming compromise after setup time) [112].

Sybil resistance in practice. Chorus provides privacy against an attacker that can control $< \kappa\%$ of registered clients. With $\kappa = 10\%$ and $N = 100$M, an attacker would need to obtain on the order of 10M valid client identities to violate privacy. Therefore, the registration mechanism should make it challenging (and ideally infeasible) to acquire fake identities at that scale. PKI providers have several options to do this, including checking government-issued mobile driver's license (mDL) [122], privacy-preserving proof-of-personhood initiatives such as World ID [6], privacy-preserving credentials over legacy credentials like passports [114], [130], device uniqueness checks such as Apple's DeviceCheck [15], web-of-trust [68], [100], or the traditional identity-proofing process [142].

Defense in depth. Our approach can be layered with other secret-recovery mechanisms for defense in depth. For instance, msk can be split into two shares, with one handled by client committees and the other by the server: an attacker that compromises more than κ fraction of clients cannot recover user secrets. We can also layer Chorus with approaches based on non-colluding servers [51], [98], [103], [111] or secure hardware [17], [55], [60], [81], [143], [148].

10. Conclusion

In this paper, we presented Chorus, a secret recovery system that distributes trust across ephemeral committees of a thousand clients each. Chorus minimizes client overhead by offloading computation to the server and further reduces per-client costs as the system scales to more clients.

Now, we discuss some avenues for future work. First, preserving committee anonymity would substantially strengthen Chorus's security. Since the committees are non-interactive in Chorus, it will map naturally to the YOSO model [44], [72] and defend against strongly adaptive adversaries. Second, although Chorus recovery costs are reasonable considering the rarity of recoveries, more efficient map-to-curve circuits [86] would further reduce client-side recovery costs. Finally, our server-aided NIVSS scheme efficiently distributes trust among clients and may prove useful in other applications that rely on the server for availability.

Acknowledgements

We thank Henry Corrigan-Gibbs for his support on this project, as well as the students in the Sky Lab Security Group for their valuable feedback. This work was supported in part by NSF Award CNS-2054869, NSF CAREER Award 1943347, and by generous gifts from Apple, Capital One, Facebook, Google, Keypasco, Mozilla, Accenture, AMD, Anyscale, Broadcom, Cisco, IBM, Intel, Intesa Sanpaolo, Lambda, Lightspeed, Mibura, Microsoft, NVIDIA, Samsung SDS, and SAP.

Ethics considerations

None

LLM usage considerations

LLMs were used for editorial purposes in this manuscript, and all outputs were inspected by the authors to ensure accuracy and originality.

References

[1] "Lastpass," https://www.lastpass.com.

[2] "Protonmail," https://proton.me/mail.

[3] "Signal," https://signal.org.

[4] "Telegram," https://telegram.org.

[5] "Whatsapp," https://www.whatsapp.com.

[6] "World id," https://world.org/world-id.

[7] "Zoom," https://zoom.us.

[8] 1Password. [Online]. Available: https://1password.com

[9] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, and G. Stern, "Bingo: Adaptively secure packed asynchronous verifiable secret sharing and asynchronous distributed key generation," *IACR Cryptol. ePrint Arch.*, 2022.

[10] D. Adei, C. Orsini, A. Scafuro, and T. Verber, "How to recover a cryptographic secret from the cloud," https://eprint.iacr.org/2023/1308, 2023.

[11] S. Agrawal, P. Miao, P. Mohassel, and P. Mukherjee, "PASTA: password-based threshold authentication," in *ACM CCS*, 2018.

[12] N. Alhaddad, M. Varia, and H. Zhang, "High-threshold AVSS with optimal communication complexity," in *FC*, 2021.

[13] Amazon, "Amazon CloudFront," https://calculator.aws/#/createCalculator/CloudFront.

[14] Android, "Android keystore system," https://developer.android.com/privacy-and-security/keystore.

[15] Apple, "Devicecheck," https://developer.apple.com/documentation/devicecheck.

[16] ——, "Protecting keys with the secure enclave," https://developer.apple.com/documentation/security/protecting-keys-with-the-secure-enclave.

[17] ——, "icloud keychain security overview," 2021, https://support.apple.com/guide/security/icloud-keychain-security-overview-sec1c89c6f3b/.

[18] arkworks rs, "arkworks-rs," https://github.com/arkworks-rs, 2024.

[19] M. Backendal, H. Davis, F. Günther, M. Haller, and K. G. Paterson, "A formal treatment of end-to-end encrypted cloud storage," in *CRYPTO*, 2024.

[20] A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu, "Password-protected secret sharing," in *ACM CCS*, 2011.

[21] L. Ballard, M. Green, B. de Medeiros, and F. Monrose, "Correlation-resistant storage via keyword-searchable encryption," *IACR ePrint*, 2005.

[22] J. Bell-Clark, A. Gascón, B. Li, M. Raykova, and P. Schoppmann, "Willow: Secure aggregation with one-shot clients," in *CRYPTO*, 2025.

[23] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," *J. Cryptol.*, 2008.

[24] M. Bellare, T. Ristenpart, and S. Tessaro, "Multi-instance security and its application to password-based cryptography," in *CRYPTO*, 2012.

[25] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *ACM CCS*, 1993.

[26] M. Bellare and L. Shea, "Flexible password-based encryption: Securing cloud storage and provably resisting partitioning-oracle attacks," in *CT-RSA*, 2023.

[27] F. Benhamouda, C. Gentry, S. Gorbunov, S. Halevi, H. Krawczyk, C. Lin, T. Rabin, and L. Reyzin, "Can a public blockchain keep a secret?" in *TCC*, 2020.

[28] F. Benhamouda, S. Halevi, H. Krawczyk, A. Miao, and T. Rabin, "Threshold cryptography as a service (in the multiserver and YOSO models)," in *ACM CCS*, 2022.

[29] A. Bhat, N. Shrestha, Z. Luo, A. Kate, and K. Nayak, "Randpiper - reconfiguration-friendly random beacons with quadratic communication," in *ACM CCS*, 2021.

[30] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth, "Succinct non-interactive arguments via linear interactive proofs," *J. Cryptol.*, 2022.

[31] A. Bittau, Ú. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnés, and B. Seefeld, "Prochlo: Strong privacy for analytics in the crowd," in *ACM SOSP*, 2017.

[32] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *Public Key Cryptography*, 2003.

[33] D. Boneh and M. K. Franklin, "Identity-based encryption from the weil pairing," in *CRYPTO*, 2001.

[34] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *ASIACRYPT*, 2001.

[35] C. Bouvier, G. Castagnos, L. Imbert, and F. Laguillaumie, "I want to ride my BICYCL : BICYCL implements cryptography in class groups," *J. Cryptol.*, 2023.

[36] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, "ZEXE: enabling decentralized private computation," in *IEEE S&P*, 2020.

[37] S. Bowe, J. Grigg, and D. Hopwood, "Halo: Recursive proof composition without a trusted setup," *IACR ePrint*, 2019.

[38] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution," in *USENIX Security*, 2018.

[39] J. V. Bulck, D. Moghimi, M. Schwarz, M. Lipp, M. Minkin, D. Genkin, Y. Yarom, B. Sunar, D. Gruss, and F. Piessens, "LVI: hijacking transient execution through microarchitectural load value injection," in *IEEE S&P*, 2020.

[40] B. Bünz, A. Chiesa, W. Lin, P. Mishra, and N. Spooner, "Proof-carrying data without succinct arguments," in *CRYPTO*, 2021.

[41] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner, "Recursive proof composition from accumulation schemes," in *TCC*, 2020.

[42] V. Buterin, "Why we need wide adoption of social recovery wallets," https://vitalik.eth.limo/general/2021/01/11/recovery.html, 2021.

[43] J. Camenisch, A. Lehmann, A. Lysyanskaya, and G. Neven, "Memento: How to reconstruct your secrets from a single password in a hostile environment," in *CRYPTO*, 2014.

[44] R. Canetti, S. Kolby, D. Ravi, E. Soria-Vazquez, and S. Yakoubov, "Taming adaptivity in YOSO protocols: The modular way," in *TCC*, 2023.

[45] S. Casacuberta, J. Hesse, and A. Lehmann, "Sok: Oblivious pseudorandom functions," in *IEEE EuroS&P*, 2022.

[46] I. Cascudo and B. David, "Publicly verifiable secret sharing over class groups and applications to DKG and YOSO," in *EUROCRYPT*, 2024.

[47] I. Cascudo, B. David, L. Garms, and A. Konring, "YOLO YOSO: fast and simple encryption and secret sharing in the YOSO model," in *ASIACRYPT*, 2022.

[48] A. Cerulli, A. Connolly, G. Neven, F. Preiss, and V. Shoup, "vetkeys: How a blockchain can keep many secrets," *IACR ePrint*, 2023.

[49] M. Chase, H. Davis, E. Ghosh, and K. Laine, "Acsesor: A new framework for auditable custodial secret storage and recovery," *IACR ePrint*, 2022.

[50] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *CRYPTO*, 1992.

[51] L. Chen, Y. Li, Q. Tang, and M. Yung, "End-to-same-end encryption: Modularly augmenting an app with an efficient, portable, and blind cloud storage," in *USENIX Security*, 2022.

[52] Y. Chen and Y. Lindell, "Feldman's verifiable secret sharing for a dishonest majority," *IACR Commun. Cryptol.*, 2024.

[53] Cloudflare, "Key Transparency Auditor," https://developers.cloudflare.com/key-transparency/, 2024.

[54] CNN, "Why messaging app signal is surging in popularity right now," 2021, https://www.cnn.com/2021/01/12/tech/signal-growth-whatsapp-confusion/.

[55] G. Connell, V. Fang, R. Schmidt, E. Dauterman, and R. A. Popa, "Secret key recovery in a global-scale end-to-end encryption system," in *OSDI*, 2024.

[56] G. Danezis, C. Díaz, C. Troncoso, and B. Laurie, "Drac: An architecture for anonymous low-volume communications," in *PETS*, 2010.

[57] P. Das, J. Hesse, and A. Lehmann, "DPaSE: Distributed Password-Authenticated Symmetric-Key Encryption, or How to Get Many Keys from One Password," in *ACM AsiaCCS*, 2022.

[58] S. Das, B. Pinkas, A. Tomescu, and Z. Xiang, "Distributed randomness using weighted vufs," in *EUROCRYPT*, 2025.

[59] S. Das, Z. Xiang, A. Tomescu, A. Spiegelman, B. Pinkas, and L. Ren, "Verifiable secret sharing simplified," in *IEEE S&P*, 2025.

[60] E. Dauterman, H. Corrigan-Gibbs, and D. Mazières, "SafetyPin: Encrypted Backups with Human-Memorable Secrets," in *OSDI*, 2020.

[61] A. Everspaugh, R. Chatterjee, S. Scott, A. Juels, and T. Ristenpart, "The pythia PRF service," in *USENIX Security*, 2015.

[62] A. Fábrega, C. O. Pérez, A. Namavari, B. Nassi, R. Agarwal, and T. Ristenpart, "Injection attacks against end-to-end encrypted applications," in *IEEE S&P*, 2024.

[63] S. H. Faller, T. Handirk, J. Hesse, M. Horváth, and A. Lehmann, "Password-protected key retrieval with(out) HSM protection," in *ACM CCS*, 2024.

[64] S. Faust, M. Kohlweiss, G. A. Marson, and D. Venturi, "On the non-malleability of the fiat-shamir transform," in *INDOCRYPT*, 2012.

[65] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *FOCS*, 1987.

[66] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO*, 1986.

[67] M. Fischlin, "Communication-efficient non-interactive proofs of knowledge with online extractors," in *CRYPTO*, 2005.

[68] T. L. Foundation, "Pgp web of trust: Delegated trust and keyservers," https://www.linuxfoundation.org/blog/blog/pgp-web-of-trust-delegated-trust-and-keyservers.

[69] M. J. Freedman and R. Morris, "Tarzan: a peer-to-peer anonymizing network layer," in *ACM CCS*, 2002.

[70] G. Fuchsbauer, A. Plouviez, and Y. Seurin, "Blind schnorr signatures and signed elgamal encryption in the algebraic group model," in *EUROCRYPT*, 2020.

[71] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *J. Cryptol.*, 2007.

[72] C. Gentry, S. Halevi, H. Krawczyk, B. Magri, J. B. Nielsen, T. Rabin, and S. Yakoubov, "YOSO: You Only Speak Once," in *CRYPTO*, 2021.

[73] C. Gentry, S. Halevi, and V. Lyubashevsky, "Practical non-interactive publicly verifiable secret sharing with thousands of parties," in *EUROCRYPT*, 2022.

[74] C. Gentry, S. Halevi, B. Magri, J. B. Nielsen, and S. Yakoubov, "Random-index PIR and applications," in *TCC*, 2021.

[75] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *ACM SOSP*, 2017.

[76] E. Giunta and A. Stewart, "Unbiasable verifiable random functions," in *EUROCRYPT*, 2024.

[77] T. Göbel and D. Huigens, "Proton key transparency whitepaper," 2024.

[78] S. Goldberg, L. Reyzin, D. Papadopoulos, and J. Včelák, "Verifiable Random Functions (VRFs)," RFC 9381, 2023.

[79] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *ACM STOC*, 1987.

[80] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM J. Comput.*, 1988.

[81] Google, "Google cloud key vault service," 2018, https://developer.android.com/about/versions/pie/security/ckv-whitepaper.

[82] V. Goyal, A. Kothapalli, E. Masserova, B. Parno, and Y. Song, "Storing and retrieving secrets on a blockchain," in *PKC*, 2022.

[83] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT*, 2016.

[84] ——, "Non-interactive distributed key generation and key resharing," *IACR ePrint*, 2021.

[85] J. Groth and M. Maller, "Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks," in *CRYPTO*, 2017.

[86] J. Groth, H. Malvai, A. Miller, and Y.-N. Zhang, "Constraint-friendly map-to-elliptic-curve-group relations and their applications," in *ASIACRYPT*, 2025.

[87] T. Hansen and D. E. E. 3rd, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)," RFC 6234.

[88] F. Hao and P. C. van Oorschot, "Sok: Password-authenticated key exchange - theory, practice, standardization and real-world lessons," in *ACM AsiaCCS*, 2022.

[89] Y. E. Housni and A. Guillevic, "Optimized and secure pairing-friendly elliptic curves suitable for one layer proof composition," in *CANS*, 2020.

[90] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu, "TOPPSS: cost-minimal password-protected secret sharing based on threshold OPRF," in *ACNS*, 2017.

[91] S. Jarecki, H. Krawczyk, and J. Xu, "OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks," in *EUROCRYPT*, 2018.

[92] Z. Jiang, P. Ye, S. He, W. Wang, R. Chen, and B. Li, "Lotto: Secure participant selection against adversarial servers in federated learning," in *USENIX Security*, 2024.

[93] S. Josefsson and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)," RFC 8032, 2017.

[94] H. Karthikeyan and A. Polychroniadou, "sfopa: One-shot private aggregation with single client interaction and its applications to federated learning," in *CRYPTO*, 2025.

[95] A. Kate and I. Goldberg, "Distributed private-key generators for identity-based cryptography," in *SCN*, 2010.

[96] A. Kate, E. V. Mangipudi, P. Mukherjee, H. Saleem, and S. A. K. Thyagarajan, "Non-interactive VSS using class groups and application to DKG," in *ACM CCS*, 2024.

[97] A. Kate, P. Mukherjee, H. Saleem, P. Sarkar, and B. Roberts, "ANARKey: A new approach to (socially) recover keys," Cryptology ePrint Archive, 2025.

[98] D. Kaviani, S. Tan, P. G. Kannan, and R. A. Popa, "Flock: A framework for deploying on-demand distributed trust," in *OSDI*, 2024.

[99] T. Kerber, A. Kiayias, and M. Kohlweiss, "Composition with knowledge assumptions," in *CRYPTO*, 2021.

[100] Keybase, "Your keybase account," https://book.keybase.io/account.

[101] E. Kokoris-Kogias, E. C. Alp, L. Gasser, P. Jovanovic, E. Syta, and B. Ford, "Calypso: Private data management for decentralized ledgers," *IACR ePrint*, 2018.

[102] A. Kothapalli, S. T. V. Setty, and I. Tzialla, "Nova: Recursive zero-knowledge arguments from folding schemes," in *CRYPTO*, 2022.

[103] A. Ladia, "Zerowallet," http://amanladia.com/wp/zerowallet/.

[104] R. W. F. Lai, C. Egger, M. Reinert, S. S. M. Chow, M. Maffei, and D. Schröder, "Simple password-hardened encryption services," in *USENIX Security*, 2018.

[105] S. Lawlor and K. Lewi, "Deploying key transparency at whatsapp," 2023, https://engineering.fb.com/2023/04/13/security/whatsapp-key-transparency/.

[106] Ledger, "How ledger device generates 24-word recovery phrase." 2023, https://support.ledger.com/hc/en-us/articles/4415198323089-How-Ledger-device-generates-24-word-recovery-phrase.

[107] V. L. Lemieux and C. Feng, *Building Decentralized Trust: Multidisciplinary Perspectives on the Design of Blockchains and Distributed Ledgers*. Springer, 2021.

[108] J. Len, M. Chase, E. Ghosh, K. Laine, and R. C. Moreno, "OPTIKS: an optimized key transparency system," in *USENIX Security Symposium*. USENIX Association, 2024.

[109] H. Li, H. Lin, A. Polychroniadou, and S. Tessaro, "LERNA: secure single-server aggregation via key-homomorphic masking," in *ASIACRYPT*, 2023.

[110] Y. Lindell, "How to simulate it - A tutorial on the simulation proof technique," in *Tutorials on the Foundations of Cryptography*, 2017.

[111] R. Little, L. Qin, and M. Varia, "Secure account recovery for a privacy-preserving web service," in *USENIX Security*, 2024.

[112] Y. Ma, J. Woods, S. Angel, A. Polychroniadou, and T. Rabin, "Flamingo: Multi-round single-server secure aggregation with applications to private federated learning," in *IEEE S&P*, 2023.

[113] H. Malvai, L. Kokoris-Kogias, A. Sonnino, E. Ghosh, E. Oztürk, K. Lewi, and S. F. Lawlor, "Parakeet: Practical key transparency for end-to-end encrypted messaging," in *NDSS*, 2023.

[114] D. Maram, H. Malvai, F. Zhang, N. Jean-Louis, A. Frolov, T. Kell, T. Lobban, C. Moy, A. Juels, and A. Miller, "Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability," in *IEEE S&P*, 2021.

[115] S. K. D. Maram, F. Zhang, L. Wang, A. Low, Y. Zhang, A. Juels, and D. Song, "CHURP: dynamic-committee proactive secret sharing," in *ACM CCS*, 2019.

[116] A. M. Matlala, "Setup Ceremonies," https://zkproof.org/2021/06/30/setup-ceremonies/, 2021.

[117] D. McGrew and J. Viega, "The galois/counter mode of operation (gcm)," *submission to NIST Modes of Operation Process*, 2004.

[118] K. Murdock, D. F. Oswald, F. D. Garcia, J. V. Bulck, F. Piessens, and D. Gruss, "Plundervolt: How a little bit of undervolting can create a lot of trouble," *IEEE S&P*, 2020.

[119] A. Nambiar and M. K. Wright, "Salsa: a structured approach to large-scale anonymity," in *ACM CCS*, 2006.

[120] K. Nissim and M. Naor, "Certificate revocation and certificate update," in *USENIX Security*, 1998.

[121] A. Nitulescu, "zk-snarks: A gentle introduction," *Ecole Normale Superieure*, 2020.

[122] A. A. of Motor Vehicle Administrators, "AAMVA mobile driver license implementation guidelines, release 1.3," https://www.aamva.org/getmedia/261ed16b-3f5c-4678-a2db-cc3016934234/MobileDLImplementationGuidelines-Version1-3.pdf, 2024.

[123] R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks (extended abstract)," in *ACM PODC*, 1991.

[124] M. Palatinus, P. Rusnak, A. Voisine, and S. Bowe, "Bip39: Mnemonic code for generating deterministic keys," 2013.

[125] K. G. Paterson and S. Srinivasan, "Security and anonymity of identity-based encryption with multiple trusted authorities," in *Pairing*, 2008.

[126] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO*, 1991.

[127] PreVeil, "Preveil: Encrypted email and file sharing," https://www.preveil.com/.

[128] Y. Psaras and D. Dias, "The interplanetary file system and the filecoin network," in *DSN (Supplements)*. IEEE, 2020.

[129] K. Reese, T. Smith, J. Dutson, J. Armknecht, J. Cameron, and K. E. Seamons, "A usability study of five two-factor authentication methods," in *SOUPS @ USENIX Security*, 2019.

[130] M. Rosenberg, J. D. White, C. Garman, and I. Miers, "zk-creds: Flexible anonymous credentials from zksnarks and existing identity infrastructure," in *IEEE S&P*, 2023.

[131] E. Roth, K. Newatia, Y. Ma, K. Zhong, S. Angel, and A. Haeberlen, "Mycelium: Large-scale distributed graph queries with differential privacy," in *ACM SOSP*, 2021.

[132] E. Roth, D. Noble, B. H. Falk, and A. Haeberlen, "Honeycrisp: large-scale differentially private aggregation without a trusted core," in *ACM SOSP*, 2019.

[133] E. Roth, H. Zhang, A. Haeberlen, and B. C. Pierce, "Orchard: Differentially private analytics at scale," in *OSDI*, 2020.

[134] A. Sayler, T. Andrews, M. Monaco, and D. Grunwald, "Tutamen: A next-generation secret-storage platform," in *ACM SoCC*, 2016.

[135] M. Schwarz, M. Lipp, D. Moghimi, J. V. Bulck, J. Stecklina, T. Prescher, and D. Gruss, "Zombieload: Cross-privilege-boundary data sampling," in *ACM CCS*, 2019.

[136] A. Shamir, "How to share a secret," *Commun. ACM*, 1979.

[137] V. Shejwalkar, A. Houmansadr, P. Kairouz, and D. Ramage, "Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning," in *IEEE S&P*, 2022.

[138] R. Sherwood, B. Bhattacharjee, and A. Srinivasan, "P5: A protocol for scalable anonymous communication," in *IEEE S&P*, 2002.

[139] V. Shoup and N. P. Smart, "Lightweight asynchronous verifiable secret sharing with optimal resilience," *J. Cryptol.*, 2024.

[140] Signal, 2024, https://signal.org/bigbrother/.

[141] R. Tamassia, "Authenticated data structures," in *European symposium on algorithms*, 2003.

[142] D. Temoshok, C. Abruzzi, Y.-Y. Choong, J. Fenton, R. Galluzzo, C. LaSalle, N. Lefkovitz, and A. Regenscheid, "Digital identity guidelines: Identity proofing and enrollment," NIST, Tech. Rep., 2024.

[143] N. Trapp, "Key to simplicity: Squeezing the hassle out of encryption key recovery," 2024, https://www.juicebox.xyz/blog/key-to-simplicity-squeezing-the-hassle-out-of-encryption-key-recovery.

[144] N. Tyagi, B. Fisch, A. Zitek, J. Bonneau, and S. Tessaro, "Versa: Verifiable registries with efficient client audits from RSA authenticated dictionaries," in *ACM CCS*, 2022.

[145] S. van Schaik, A. Milburn, S. Österlund, P. Frigo, G. Maisuradze, K. Razavi, H. Bos, and C. Giuffrida, "RIDL: rogue in-flight data load," in *IEEE S&P*, 2019.

[146] S. van Schaik, M. Minkin, A. Kwong, D. Genkin, and Y. Yarom, "Cacheout: Leaking data on intel cpus via cache evictions," in *IEEE S&P*, 2021.

[147] R. Vassantlal, E. Alchieri, B. Ferreira, and A. Bessani, "Cobra: Dynamic proactive secret sharing for confidential bft services," in *IEEE S&P*, 2022.

[148] Whatsapp, "Security of end-to-end encrypted backups," 2021, https://www.whatsapp.com/security/WhatsApp_Security_Encrypted_Backups_Whitepaper.pdf.

[149] ——, "About government requests for user data," 2024, https://faq.whatsapp.com/808280033839222.

[150] M. Whittaker and J. Lund, "Privacy is priceless, but signal is expensive," 2023, https://signal.org/blog/signal-is-expensive/.

[151] A. Whitten and J. D. Tygar, "Why johnny can't encrypt: A usability evaluation of PGP 5.0," in *USENIX Security*, 1999.

[152] A. C.-C. Yao, "How to generate and exchange secrets," in *FOCS*, 1986.

[153] T. Yurek, L. Luo, J. Fairoze, A. Kate, and A. Miller, "hbacss: How to robustly share many secrets," in *NDSS*, 2022.

[154] T. Yurek, Z. Xiang, Y. Xia, and A. Miller, "Long live the honey badger: Robust asynchronous DPSS and its applications," in *USENIX Security*, 2023.

[155] F. Zhang, R. Safavi-Naini, and W. Susilo, "Efficient verifiably encrypted signature and partially blind signature from bilinear pairings," in *INDOCRYPT*, 2003.

Appendix A. Additional Material

A.1. ECPSS Parameter Analysis

We show how to choose parameters n and t based on the fraction of total N clients that can be compromised, κ, or offline, ϕ. ($\phi \geq \kappa$ since compromised clients can always choose to not respond.)

Upper bound on corrupted committee members. Each client is chosen independently with a probability of n/N, and there are $\leq \kappa N$ corrupted clients. Let $B_1 = \kappa n(1+\epsilon_1)$ denote the maximum tolerable corrupted nominating clients. By the Chernoff bound, we can set n and ϵ_1 such that:

$$\Pr[\mathsf{Bin}(\kappa N, n/N) > B_1] < e^{-\kappa n \epsilon_1^2/(2+\epsilon_1)} \leq 2^{-\lambda}, \quad (1)$$

where $\lambda = 128$ is computational security parameter. Note that even though a corrupted server can try different seeds and outcomes, it is computationally bounded and a probability of $2^{-\lambda}$ should suffice.

TABLE 10. PARAMETERS (n, t) VS. CORRUPTION κ AND OFFLINE ϕ FRACTIONS

	$\phi = 25\%$	$\phi = 50\%$	$\phi = 75\%$
$\kappa = 1\%$	$(365, 100)$	$(561, 105)$	$(1214, 121)$
$\kappa = 5\%$	$(449, 144)$	$(761, 176)$	$(2175, 299)$
$\kappa = 10\%$	$(573, 212)$	$(1090, 300)$	$(4596, 793)$
$\kappa = 15\%$	$(725, 299)$	$(1563, 488)$	$(11756, 2369)$
$\kappa = 20\%$	$(923, 416)$	$(2298, 793)$	$(52138, 11833)$

Lower bound on online committee members. Each client is chosen independently with a probability of n/N, and there are $\geq (1-\phi)N$ online clients. Let $B_2 = n(1-\phi)(1-\epsilon_2)$ be the minimum online clients required for availability. By the Chernoff bound, we can set n and ϵ_2 such that:

$$\Pr[\mathsf{Bin}((1-\phi)N, n/N) < B_2] < e^{-(1-\phi)n\epsilon_2^2/2} \leq 2^{-\sigma}, \quad (2)$$

where $\sigma = 80$ is the statistical security parameter. Note that for availability, the server is honest and bounding the probability by $2^{-\sigma}$ ensures this event never happens.

Concrete Parameters. We iterate over all possible values of n and solve for ϵ_1 and ϵ_2 until we hit an n s.t. $B_1 \leq B_2$. Then, we pick that n and set $t = B_1$. The resulting parameters are summarized in Table 10.

Parameter Limits. We require $\kappa < \min\{\phi, 1-\phi\}$:

- $\phi \geq \kappa$ because corrupt clients can choose to not respond.
- $1-\phi > \kappa$, because as n tends to infinity, the upper bound on the corrupt committee members B_1 approaches κn, while the lower bound on the online committee members B_2 approaches $(1-\phi)n$, and we require $B_1 < B_2$.

A.2. Key Manager KM

In this section, we describe how the idealized key manager $\mathcal{F}^{\mathsf{KM}}$ (Figure 2) is thresholdized across dynamic client committees, using ECPSS as a building block (§5.1). The security proof of the end-to-end key manager protocol with ECPSS is deferred to Appendix A.7. In the static threshold IBE setting, each committee member $\mathcal{M}_i$ has a private share msk_i and public share commitment $\mathsf{scmt}_i = g^{\mathsf{msk}_i}$. When using an ECPSS scheme to manage msk, we also have the invariant that for each committee $\mathcal{C}_e$, its member $\mathcal{M}_{e,i}$ holds a private share $\mathsf{msk}_{e,i}$ and a public share commitment which we denote with $\mathsf{scmt}_{e,i}$. These share commitments come for free from the commitments used to link successive handovers (§5.4).

KeyGen. To set up msk, we can adapt the two-round distributed key generation (DKG) protocol from cgVSS [96] to dynamic committees. Let g and g' be two generators of the group $\mathbb{G}_1$. The DKG operates over two epochs:

- On server's instruction, each (online) member $\mathcal{M}_i \in \mathcal{C}_0$ samples a random field element $s_i \in \mathbb{F}_q$ and invokes $\mathsf{ECPSS.Share}(s_i, \mathcal{C}_1)$ while using g' as the generator in the VSS scheme.
- Server initiates $\mathsf{ECPSS.Handover}(\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2)$, where each (online) member $\mathcal{M}_i \in \mathcal{C}_1$ uses generator g' when

invoking saVSS in $\mathsf{msk}_{1,i} \leftarrow \mathsf{ECPSS.Reconstruct}(\mathcal{C}_0)$ and generator g when invoking saVSS in $\mathsf{ECPSS.Share}(\mathsf{msk}_{1,i}, \mathcal{C}_2)$. By doing so, $\mathcal{M}_i \in \mathcal{C}_1$ switched the generator for the share commitments for $\mathcal{C}_2$ from g' to g. To prove that the two share commitments are consistent, $\mathcal{M}_i$ also generates a discrete-log equality NIZK proof [50]. This proof is then additionally verified by $\mathcal{S}$ and $\mathcal{C}_2$-members during $\mathsf{ECPSS.Reconstruct}(\mathcal{C}_1)$ to check consistency with the previous state.

- The server can then use the share commitments of $\mathcal{C}_2$ to interpolate the public key $\mathsf{pk} = g^{\mathsf{msk}}$.

The two-round keygen uses distinct generators to prevent public-key biasing attacks [71]. This requires that commitments from the first round (w.r.t. g') do not reveal anything about pk, and we rely on external Diffie-Hellman (XDH) assumption [21] in $\mathbb{G}_1$ to prove that.

Get public key. Once $\mathcal{C}_1$ has completed its transfer to $\mathcal{C}_2$, any party $\mathcal{P}$ can retrieve the public key pk from $\mathcal{S}$ and verify it relative to the share commitments of $\mathcal{C}_2$.

Recovery. Each epoch, a committee simultaneously responds to a batch of recovery requests and transfers the state to the next committee. For simplicity, we consider a single recovery request per epoch in this section. To perform recovery, the following additional steps are performed during $\mathsf{ECPSS.Handover}(\mathcal{C}_{\mathsf{prev}}, \mathcal{C}_{\mathsf{curr}}, \mathcal{C}_{\mathsf{next}})$:

- Before initiating the handover, $\mathcal{S}$ publishes the new guess-limiting state $\mathsf{st}_{\mathsf{curr}}$ to the transparency log for that epoch.
- Let $\mathsf{st}_{\mathsf{prev}}$ be the public state for the prior epoch. For each (online) member $\mathcal{M}_i \in \mathcal{C}_{\mathsf{curr}}$, $\mathcal{S}$ additionally sends $(\mathsf{rqst}, \mathsf{st}_{\mathsf{prev}}, \mathsf{st}_{\mathsf{curr}}, \pi^{\mathsf{st}})$ to $\mathcal{M}_i$, which ensures that $\mathsf{st}_{\mathsf{prev}}$ and $\mathsf{st}_{\mathsf{curr}}$ match the state commitments on the transparency log for the corresponding epochs. If $\psi_L(\mathsf{rqst}, \mathsf{st}_{\mathsf{curr}}, \mathsf{st}_{\mathsf{prev}}, \pi^{\mathsf{st}}) = 1$ (see §4), $\mathcal{M}_i$ computes the recovery response as $\mathsf{resp}_i \leftarrow \mathsf{rqst}^{\mathsf{msk}_{\mathsf{curr},i}}$. $\mathcal{M}_i$ then additionally sends resp_i to the server $\mathcal{S}$. If any check fails or any ECPSS sub-routine aborts, $\mathcal{M}_i$ aborts.
- Given the recovery responses $\{\mathsf{resp}_i\}_i$, like standard threshold BF-IBE, the server can identify which responses are correct using share commitments. Thus, given $\mathsf{scmt}_{\mathsf{curr},i}$, server can verify the correctness of the responses by checking if $\hat{e}(\mathsf{scmt}_{\mathsf{curr},i}, \mathsf{rqst}) = \hat{e}(g, \mathsf{resp}_i)$. After verification, it can take any t correct responses and interpolate the final response resp.

Here, we need to post the state $\mathsf{st}_{\mathsf{curr}}$ to the transparency log to ensure that an adversary that controls $t-1$ committee members and the server can not contact each honest member $\mathcal{M}_i \in \mathcal{C}_{\mathsf{curr}}$ with a different recovery request while only a single attempt gets logged in $\mathsf{st}_{\mathsf{curr}}$.

A.3. Extended Evaluation

A.3.1. One-time Setup Cost. The following table reports the one-time cost for public key setup, which is similar to the typical costs analyzed in §7.

A.3.2. Chorus Costs with $\kappa = 1\%$ and $\phi = 75\%$. In this section, we evaluate a Chorus configuration where only 25% of the clients need to be online, and at most 1% can be corrupted at any time. This configuration reduces costs for both clients and the server compared to the configuration in §7. For the key-manager client (secret-recovery client is agnostic to ECPSS parameters), compute costs drop by 12% and communication by 28%. The Chorus server sees a 54% reduction in compute and 20% in communication.

Ideal functionality $\mathcal{F}^{\mathsf{PKI}}$ for PKI

This functionality initializes an empty dictionary T which maps a user's unique identity uid to a signature public key $\mathsf{pk}^{\mathsf{sig}}$.

- **Register**: On input $\langle \mathsf{Register}, \mathsf{uid}, \mathsf{pk}^{\mathsf{sig}} \rangle$ from client $\mathcal{P}$, if uid is a real user (see §9), set $T[\mathsf{uid}] \leftarrow \mathsf{pk}^{\mathsf{sig}}$.
- **Verify**: On input $\langle \mathsf{Verify}, \mathsf{uid}, \mathsf{pk}^{\mathsf{sig}} \rangle$ from party $\mathcal{P}$, if $T[\mathsf{uid}] = \mathsf{pk}^{\mathsf{sig}}$, return $\top$, else return $\bot$.

Figure 11. Ideal functionality for PKI.

Ideal functionality $\mathcal{F}^{\mathsf{tlog}}$ for transparency log

Functionality initializes an empty key-value map T. $\mathcal{S}$ denotes the server.

- **Publish**: On input $\langle \mathsf{Publish}, k, v \rangle$ from $\mathcal{S}$, update $T[k] = v$.
- **Verify**: On input $\langle \mathsf{Verify}, k, v \rangle$ from $\mathcal{P}$, if $T[k] = v$, return $\top$, else return $\bot$.

Figure 12. Ideal functionality for transparency log.

	— Time —		— Comm —	
	Client	Server	Client	Server
Epoch 0	15.2 s	23.6 s	4.94 MB	5.39 GB
Epoch 1	21.1 s	26.8 s	12.51 MB	13.64 GB

A.4. anon-IND-ID-CCA property of BF-IBE

In this section, we define the anon-IND-ID-CCA property satisfied by the BF-IBE scheme [33]. This property was proven by Paterson and Srinivasan [125] (henceforth referred to as [PS08]) for a slight variant of BF-IBE in the multiple trusted authority (TA) setting. Specifically, they prove that this BF-IBE variant satisfies the m-RA-IND-ID-CCA property defined in [PS08]. This property is identical to the anon-IND-ID-CCA property we define below when #TA is set to 1. Throughout the paper and in our implementation, BF-IBE denotes the [PS08] variant with #TA=1.

Let $\mathbf{G}_{\mathsf{IBE}}^{\mathsf{anon\text{-}IND\text{-}ID\text{-}CCA}-b}$ denote the anon-IND-ID-CCA game w.r.t. an IBE scheme, and let $W_b(\mathcal{A})$ denote the event that the game $\mathbf{G}_{\mathsf{IBE}}^{\mathsf{anon\text{-}IND\text{-}ID\text{-}CCA}-b}(\mathcal{A})$ outputs 1. Then we call IBE anon-IND-ID-CCA-secure if for all non-uniform PPT adversaries $\mathcal{A}$, $\Pr[W_0(\mathcal{A})] - \Pr[W_1(\mathcal{A})]| \leq \mathrm{negl}(\lambda)$.

$\mathbf{G}^{\text{anon-IND-ID-CCA}-b}_{\mathsf{IBE}}(\mathcal{A})$

$(\mathsf{msk}, \mathsf{pk}) \leftarrow \mathsf{IBE.KeyGen}(1^\lambda)$

$\{\mathsf{id}_i, m_i\}_{i\in\{0,1\}} \leftarrow \mathcal{A}^{\text{EXT},\text{DEC}}(\mathsf{pk})$

$\mathsf{ct}^* \leftarrow \mathsf{IBE.Encrypt}(\mathsf{pk}, \mathsf{id}_b, m_b)$

$b' \leftarrow \mathcal{A}^{\text{EXT},\text{DEC}}(\mathsf{ct}^*)$

return $b' \wedge \mathsf{id}_0, \mathsf{id}_1 \notin E \wedge \mathsf{ct}^* \notin D$

EXT(id)

Add id to E

return $\mathsf{IBE.Extract}(\mathsf{msk}, \mathsf{id})$

DEC(id, ct)

Add ct to D

$d_{\mathsf{id}} \leftarrow \mathsf{IBE.Extract}(\mathsf{msk}, \mathsf{id})$

return $\mathsf{IBE.Decrypt}(d_{\mathsf{id}}, \mathsf{ct})$

A.5. ECPSS Scheme in the Client-Server Setting

An ECPSS scheme is parameterized by threshold t, committee size n, and total clients N. Our scheme also has an additional TTL parameter δ which denotes the duration after which a client that has announced its committee nomination to the server erases its encryption keys (§2.4). The client can later re-join the protocol by sampling fresh encryption keys (§5.2). This scheme uses the server-aided VSS scheme saVSS (Figure 13), and the transparency log ($\mathcal{F}^{\mathsf{tlog}}$, Figure 12) and PKI ($\mathcal{F}^{\mathsf{PKI}}$, Figure 11) ideal functionalities as building blocks. The CommitteeSelection procedure outputs a committee $\mathcal{C}$, where each committee member has an index (called seat in committee selection), and two public keys of interest: the corresponding client's encryption public key $\mathsf{pk}^{\mathsf{enc}}$ and signature public key $\mathsf{pk}^{\mathsf{sig}}$.

The remaining protocols Share, Reconstruct, and Handover build on committee selection, and are only called by the committee members in our key manager protocol (see Appendix A.2). As such, we'll assume that all of these clients have an index, an encryption public and a signature public key. If the protocol has a committee $\mathcal{C}$ as an input, then we assume that the caller of the protocol (i.e., the corresponding committee member) has verified $\mathcal{C}$ using the Get Committee function of committee selection before proceeding. To avoid confusion with other clients in the system, we denote a committee member with index k as $\mathcal{M}_k$ instead of $\mathcal{P}_k$.

We denote the j-th coefficient of the polynomial shared by $\mathcal{M}_i \in \mathcal{C}_{\mathsf{curr}}$ with $\mathsf{ccmt}_{\mathsf{curr},i,j}$. Given the coefficient commitments from t $\mathcal{C}_{\mathsf{curr}}$-members, anyone can interpolate (in the exponent) the commitment to $\mathcal{C}_{\mathsf{next}}$-members' shares. We denote the share commitment of $\mathcal{M}_i \in \mathcal{C}_{\mathsf{curr}}$ with $\mathsf{scmt}_{\mathsf{curr},i}$.

A.5.1. $\mathsf{ECPSS}_{t,n,N}.\mathsf{CommitteeSelection}(e)$**.** Let $\mathcal{P}$ denote the set of all clients. Recall from §5.2 that each party $\mathcal{P}_i$ has a unique index uid_i and three keypairs: a signature keypair $(\mathsf{pk}^{\mathsf{sig}}_i, \mathsf{sk}^{\mathsf{sig}}_i)$ that is registered with $\mathcal{F}^{\mathsf{PKI}}$, a VRF keypair $(\mathsf{pk}^{\mathsf{vrf}}_i, \mathsf{sk}^{\mathsf{vrf}}_i)$, an ephemeral encryption keypair $(\mathsf{pk}^{\mathsf{enc}}_i, \mathsf{sk}^{\mathsf{enc}}_i)$. The set of public keys is referred to as ckeys_i. Whenever clients sample fresh encryption keys $(\mathsf{pk}^{\mathsf{enc}}_i, \mathsf{sk}^{\mathsf{enc}}_i)$ after being on a committee (§5.2), we assume that they also send the server a well-formedness NIZK proof π^{enc}_i for these keys as the pVSS scheme we build on requires that the encryption keys be honestly generated. The goal of committee selection procedure is to output a committee $\mathcal{C}_e$ in epoch e such that only a small subset of the total clients are selected and any client can verifiably retrieve the public keys of the clients in this committee. We now describe the steps in committee selection in chronological order:

- **(Server) Setup Sortition**: Given a client list $\mathcal{P} = \{\mathcal{P}_i\}_i$ such that $\mathcal{F}^{\mathsf{PKI}}$ returns $\top$ on $\langle \mathsf{Verify}, \mathsf{uid}_i, \mathsf{pk}^{\mathsf{sig}}_i \rangle$ and π^{enc}_i is valid for $\mathsf{pk}^{\mathsf{enc}}_i$, construct a Merkle tree (hash function

Server-Aided NIVSS $\mathsf{saVSS}_{n,t}$ in $\mathcal{F}^{\mathsf{tlog}}$-hybrid

n and t denote the number of recipients and the threshold of the secret-sharing scheme. sid denotes the share-id. $\mathcal{S}$ denotes the server, $\mathcal{D}_k$ the dealer, $\{\mathcal{R}_i\}_{i\in[n]}$ the recipients, and $\mathcal{F}^{\mathsf{tlog}}$ the transparency log (Figure 12). pVSS denotes the NIVSS scheme with public verifiability from [96] and aVSS denotes the NIVSS scheme with abort from [52]. Let $\mathsf{pp}^{\mathsf{pv}}$ and $\mathsf{pp}^{\mathsf{ab}}$ denote the corresponding public parameters chosen such that both schemes use the same dlog-hard group $\mathbb{G}$ with generator g and order q for commitments. Let $(\mathsf{pk}^{\mathsf{enc}}_i, \mathsf{sk}^{\mathsf{enc}}_i)_{i\in[n]}$ denote the public key encryption (specified by $\mathsf{pp}^{\mathsf{pv}}$) keypairs of the recipients, and $(\mathsf{pk}^{\mathsf{sig}}_k, \mathsf{sk}^{\mathsf{sig}}_k)$ denote the signature keypair of the dealer. Per the security requirement of pVSS [96], we assume that the encryption keys are well-formed and our ECPSS committee selection introduces checks to guarantee this (Appendix A.5.1).

$\mathsf{Deal}_{\mathcal{D}_k}(\mathsf{sid}, s, \{\mathsf{pk}^{\mathsf{enc}}_i\}_i, \mathsf{sk}^{\mathsf{sig}}_k)$:

- $\forall j \in [t]$, sample $a_j \leftarrow \mathbb{F}_q$. Set $a_0 \leftarrow s$.
- $\forall i \in [n]$, set $s_i \leftarrow \sum_{j\in[t]} a_j \cdot (i+1)^j$.
- Commit to the coeffs $\{a_j\}_j$ to get $\mathsf{cmt} = \{g^{a_j}\}_j$.
- Encrypt the shares with recipients public keys and produce a proof of correct sharing: $(\mathsf{ct}, \pi^{\mathsf{pv}}) \leftarrow \mathsf{pVSS}_{n,t}.\mathsf{EncAndProve}(\mathsf{pp}^{\mathsf{pv}}, \mathsf{cmt}, \{s_i, \mathsf{pk}_i\}_i)$.
- Produce a NIVSS with abort proof w.r.t. cmt: $\pi^{\mathsf{ab}} \leftarrow \mathsf{aVSS}_{n,t}.\mathsf{Prove}(\mathsf{pp}^{\mathsf{ab}}, \{a_j\}_j, \mathsf{cmt})$.
- $\forall i \in [n]$, sign $(\mathsf{sid}, k, i, \mathsf{cmt}, \mathsf{ct}[i])$ using $\mathsf{sk}^{\mathsf{sig}}_k$ to get σ_i.
- Send $\mathsf{dlng} \leftarrow (\mathsf{cmt}, \mathsf{ct}, \{\sigma_i\}_i, \pi^{\mathsf{pv}}, \pi^{\mathsf{ab}})$ to $\mathcal{S}$.

$\mathsf{ProcessDealing}_{\mathcal{S}}(\mathsf{sid}, k, \mathsf{dlng}, \{\mathsf{pk}^{\mathsf{enc}}_i\}_i, \mathsf{pk}^{\mathsf{sig}}_k)$:

- $\mathcal{S}$ parses $\mathsf{dlng} = (\mathsf{cmt}, \mathsf{ct}, \{\sigma_i\}_i, \pi^{\mathsf{pv}}, \pi^{\mathsf{ab}})$ and sends $(\mathsf{sid}, k, \perp)$ to $\{\mathcal{R}_i\}_i$ if any of the following check fails:
  - $\mathsf{pVSS}_{n,t}.\mathsf{Verify}(\mathsf{pp}^{\mathsf{pv}}, \mathsf{cmt}, \mathsf{ct}, \{\mathsf{pk}^{\mathsf{enc}}_i\}_i, \pi^{\mathsf{pv}})$
  - $\mathsf{aVSS}_{n,t}.\mathsf{Verify}(\mathsf{pp}^{\mathsf{ab}}, \mathsf{cmt}, \pi^{\mathsf{ab}})$
  - $\forall i$, verify σ_i w.r.t. $(\mathsf{sid}, k, i, \mathsf{cmt}, \mathsf{ct}[i])$ and $\mathsf{pk}^{\mathsf{sig}}_k$
- Else, send $\langle \mathsf{Publish}, \mathsf{sid}\|k, \mathsf{cmt} \rangle$ to $\mathcal{F}^{\mathsf{tlog}}$ and for $i \in [n]$, send $\mathsf{dlngLite}_i \leftarrow (\mathsf{cmt}, \mathsf{ct}[i], \sigma_i, \pi^{\mathsf{ab}})$ to $\mathcal{R}_i$.

$\mathsf{Receive}_{\mathcal{R}_i}(\mathsf{sid}, k, \mathsf{dlngLite}_i, \mathsf{sk}^{\mathsf{enc}}_i, \mathsf{pk}^{\mathsf{sig}}_k) \to \{s_i, \perp\}$:

- Parse $\mathsf{dlngLite}_i = (\mathsf{cmt}, \mathsf{ct}_i, \sigma, \pi^{\mathsf{ab}})$, decrypt ct_i with $\mathsf{sk}^{\mathsf{enc}}_i$ to get s_i, and perform the following checks:
  - $\mathsf{aVSS}_{n,t}.\mathsf{Verify}(\mathsf{pp}^{\mathsf{ab}}, \mathsf{cmt}, \pi^{\mathsf{ab}})$
  - Verify s_i w.r.t. cmt: $g^{s_i} = \prod_{j\in[t]} \mathsf{cmt}[j]^{(i+1)^j}$
  - Verify σ w.r.t. $(\mathsf{sid}, k, i, \mathsf{cmt}, \mathsf{ct})$ and $\mathsf{pk}^{\mathsf{sig}}_k$.
  - $\mathcal{F}^{\mathsf{tlog}}$ responds to $\langle \mathsf{Verify}, \mathsf{sid}\|k, \mathsf{cmt} \rangle$ with $\top$.
- If all checks pass, output s_i, else abort.

Figure 13. Our Server-Aided NIVSS Scheme

instantiated with random oracle) on the following dictionary $\{\mathsf{uid}_i : \mathsf{pk}_i^{\mathsf{vrf}}\}_{\mathcal{P}_i \in \mathcal{P}}$. Let rt denote the root of the resulting tree. Send rt to all clients in $\mathcal{P}$.

- **(Client) Check Sortition**: During epoch e[2], compute $\mathsf{seed} = H_{\mathsf{seed}}(\mathsf{rt}, e)$, where H_{seed} is a random oracle that has the same output space as the input space of the unbiasable VRF scheme. Check sortition result $\pi^{\mathsf{srt}} \leftarrow \mathsf{Sortition}(\mathsf{sk}^{\mathsf{vrf}}, \mathsf{seed}, n, N)$; sortition is configured to choose client with probability $\frac{n}{N}$. If $\pi^{\mathsf{srt}} = \perp$, return $\perp$. Otherwise, send π^{srt} to $\mathcal{S}$ and then start a local timer that will trigger a timeout after δ time. When the δ-timeout is triggered, delete $\mathsf{sk}_i^{\mathsf{enc}}$ if it was not already erased during handover.
- **(Server) Publish Committee**: For each sortition proof π_i^{srt} received from client $\mathcal{P}_i \in \mathcal{P}$ with keys ckeys_i, verify π_i^{srt} w.r.t. $(\mathsf{pk}_i^{\mathsf{vrf}}, \mathsf{rt}, e)$, and if successful, assign a unique index seat_i to the client (used to define Shamir shares), and add $\mathsf{mmbrDt}_i = \{\mathsf{uid}_i, \mathsf{seat}_i, \mathsf{ckeys}_i, \pi_i^{\mathsf{srt}}, \pi_i^{\mathsf{mt}}\}$ to the set $\mathsf{mmbrsDt}$, where π_i^{mt} is the Merkle inclusion proof on committee members VRF keys w.r.t. rt. Return $\mathsf{cmtDt}_e = (\mathsf{rt}, \mathsf{mmbrsDt})$ and send $\langle \mathsf{Publish}, e \| \text{“cmt-data”}, \mathsf{cmtDt}_e \rangle$ to $\mathcal{F}^{\mathsf{tlog}}$.
- **(Client) Get Committee**: Get $\mathsf{cmtDt}_e = (\mathsf{rt}, \mathsf{mmbrsDt})$ from $\mathcal{S}$ and perform the following checks:
  - Send $\langle \mathsf{Verify}, e \| \text{“cmt-data”}, \mathsf{cmtDt}_e \rangle$ to $\mathcal{F}^{\mathsf{tlog}}$ and verify it responds with $\top$.
  - Verify that $|\mathsf{mmbrsDt} \geq t|$ and $\{\mathsf{uid}_i, \mathsf{seat}_i, \mathsf{ckeys}_i\}$ for each $\mathsf{mmbrDt}_i \in \mathsf{mmbrsDt}$ are distinct.
  - For each $\mathsf{mmbrDt}_i \in \mathsf{mmbrsDt}$, parse $\mathsf{mmbrDt}_i = \{\mathsf{uid}_i, \mathsf{seat}_i, \mathsf{ckeys}_i, \pi_i^{\mathsf{srt}}, \pi_i^{\mathsf{mt}}\}$, parse $\mathsf{ckeys}_i = (\mathsf{pk}_i^{\mathsf{sig}}, \mathsf{pk}_i^{\mathsf{vrf}}, \mathsf{pk}_i^{\mathsf{enc}})$, and perform the following checks:
    - Verify $\mathcal{F}^{\mathsf{PKI}}$ responds to $\langle \mathsf{Verify}, \mathsf{uid}_i, \mathsf{pk}_i^{\mathsf{sig}} \rangle$ with $\top$.
    - Verify π_i^{enc} is valid w.r.t. $\mathsf{pk}_i^{\mathsf{enc}}$.
    - Verify π_i^{mt} and check π_i^{srt} is valid w.r.t. $(\mathsf{pk}_i^{\mathsf{vrf}}, \mathsf{rt}, e)$.

  Return $\mathsf{cmtKeys}_e \leftarrow \{\mathsf{uid}_i, \mathsf{seat}_i, \mathsf{pk}_i^{\mathsf{enc}}, \mathsf{pk}_i^{\mathsf{sig}}\}_{\mathsf{mmbrDt}_i}$ if all checks pass, else return $\perp$.

A.5.2. $\mathsf{ECPSS}_{t,n,N}.\mathsf{Share}(s, \mathcal{C})$**.** Let some committee member $\mathcal{M}_k$ be the caller of this API in epoch e. Let $\mathsf{PK} = \{\mathsf{pk}_i^{\mathsf{enc}}\}_{\mathcal{M}_i \in \mathcal{C}}$ denote the encryption public keys of the committee $\mathcal{C}$-members and $\mathsf{pk}_k^{\mathsf{sig}}$ denote the signature public key of $\mathcal{M}_k$. The sharing proceeds as follows:

- If δ-timeout has already been triggered for $\mathcal{M}_k$, it aborts. Else, $\mathcal{M}_k$ invokes $\mathsf{saVSS.Deal}(e, s, \mathsf{PK}, \mathsf{sk}_k^{\mathsf{sig}})$ to generate a dealing dlng_k that shares secret s to the committee $\mathcal{C}$ while setting session-id $\mathsf{sid} = e$, which is then sent to the server $\mathcal{S}$ for processing.
- The server calls $\mathsf{saVSS.ProcessDealing}(e, k, \mathsf{dlng}_k, \mathsf{PK}, \mathsf{pk}_k^{\mathsf{sig}})$ to get lite-dealings $\{\mathsf{dlngLite}_{k,i}\}_{\mathcal{M}_i \in \mathcal{C}}$ for each committee $\mathcal{C}$-member. The server then sends $\mathsf{dlngLite}_{k,i}$ to $\mathcal{M}_i \in \mathcal{C}$ when they come online.

A.5.3. $\mathsf{ECPSS}_{t,n,N}.\mathsf{Reconstruct}(\mathcal{C})$**.** Let $\mathcal{M}_i$ be a committee member that is supposed to receive a secret from committee $\mathcal{C}$ that was shared in epoch e. Let $\{\mathsf{pk}_k^{\mathsf{sig}}\}_{\mathcal{M}_k \in \mathcal{C}}$ denote the signature public keys of the the committee $\mathcal{C}$-members. The reconstruction proceeds as follows:

- $\mathcal{M}_i$ waits to receive t lite-dealings $\{\mathsf{dlngLite}_{k,i}\}_{\mathcal{M}_k \in \mathcal{C}'}$ from the server where $\mathcal{C}' \subset \mathcal{C}$ and $|\mathcal{C}'| = t$. If the δ-timeout is triggered before it receives the lite-dealings, $\mathcal{M}_i$ aborts.
- $\forall \mathcal{M}_k \in \mathcal{C}'$, $\mathcal{M}_i$ invokes $s_k \leftarrow \mathsf{saVSS.Receive}(e, k, \mathsf{dlngLite}_{k,i}, \mathsf{sk}_i^{\mathsf{enc}}, \mathsf{pk}_k^{\mathsf{sig}})$. If any Receive call aborts, $\mathcal{M}_k$ aborts.
- Otherwise, given $\{i, s_i\}_{\mathcal{M}_i \in \mathcal{C}'}$, $\mathcal{M}_k$ interpolates the secret s (using indices i as the evaluation points during Shamir reconstruction), and outputs s.

A.5.4. $\mathsf{ECPSS}_{t,n,N}.\mathsf{Handover}(\mathcal{C}_{\mathsf{prev}}, \mathcal{C}_{\mathsf{curr}}, \mathcal{C}_{\mathsf{next}})$**.** During epoch e, server $\mathcal{S}$ initiates the handover, where committee $\mathcal{C}_{\mathsf{curr}}(\mathsf{curr} = e - 2)$ receives the secret (msk of the IBE scheme in Appendix A.2) from $\mathcal{C}_{\mathsf{prev}}(\mathsf{prev} = e - 3)$, and re-shares it to committee $\mathcal{C}_{\mathsf{next}}(\mathsf{next} = e - 1)$. In addition to performing the handover, $\mathcal{C}_{\mathsf{curr}}$-members also verify that the handover generated by $\mathcal{C}_{\mathsf{prev}}$ is consistent with the handover received by $\mathcal{C}_{\mathsf{prev}}$. To aid this verification, the server $\mathcal{S}$ publishes the share commitments for each committee to the transparency log, and the receiving committee members verify that each sender committee member created a dealing consistent with its share commitment. Moreover, the receiving committee members verify the share commitments the server posted for them and only perform the handover if these commitments are consistent with the handover from the sender committee. In more detail, the handover proceeds as follows:

- Each (online) party $\mathcal{M}_i \in \mathcal{C}_{\mathsf{curr}}$ performs the following operations (aborting if Share, Reconstruct abort or any of the checks fail):
  - Invoke $\mathsf{msk}_i \leftarrow \mathsf{ECPSS}_{t,n,N}.\mathsf{Reconstruct}(\mathcal{C}_{\mathsf{prev}})$, and receive $\mathsf{scmt}_{\mathsf{prev}}$ and $\mathsf{scmt}_{\mathsf{curr}}$ in addition to the handover from $\mathcal{S}$.
  - Verify that $\mathcal{F}^{\mathsf{tlog}}$ responds with $\top$ on queries $\langle \mathsf{Verify}, \mathsf{prev} \| \text{“scmt”}, \mathsf{scmt}_{\mathsf{prev}} \rangle$ and $\langle \mathsf{Verify}, \mathsf{curr} \| \text{“scmt”}, \mathsf{scmt}_{\mathsf{curr}} \rangle$.
  - $\forall \mathcal{M}_k \in \mathcal{C}_{\mathsf{prev}}$, verify that $\mathsf{ccmt}_{\mathsf{prev},k,0} = \mathsf{scmt}_{\mathsf{prev},k}$.
  - Interpolate $\mathsf{scmt}'_{\mathsf{curr}}$ from $\{\mathsf{ccmt}_{\mathsf{prev},k,j}\}_{\mathcal{M}_k \in \mathcal{C}_{\mathsf{prev}}, j \in [t]}$ and verify that $\mathsf{scmt}_{\mathsf{curr}} = \mathsf{scmt}'_{\mathsf{curr}}$.
  - Invoke $\mathsf{ECPSS}_{t,n,N}.\mathsf{Share}(\mathsf{msk}_i, \mathcal{C}_{\mathsf{next}})$ and erase the share msk_i. Also erase $\mathsf{sk}_i^{\mathsf{enc}}$ if it was not already deleted due to the local δ-timeout.
- As part of the Share invocations by members of $\mathcal{C}_{\mathsf{curr}}$, server $\mathcal{S}$ receives the dealings generated by $\mathcal{C}_{\mathsf{curr}}$. In addition to the saVSS validity checks performed independently on these dealings in Share, $\mathcal{S}$ also checks

2. Here e refers to the epoch maintained locally at the client which could be different from the system epoch maintained by the server. Server needs to ensure that the local and system epochs are (loosely) synchronized to ensure availability. Our security proof shows that a malicious server that does not synchronize the epochs can only affect availability and not privacy.

Privacy Game $\mathbf{G}^{\mathsf{SR,priv}}_{\mathcal{PW},L}$ for Key Recovery

This game is parameterized by a password distribution $\mathcal{PW}$ and password guess limit L, and defined w.r.t. a key recovery scheme $\mathsf{SR}_L = (\mathsf{KeyGen}, \mathsf{Backup}, \mathsf{Recover})$. During the game, the challenger and the adversary $\mathcal{A}$ play the role of the honest and corrupted entities, resp., in SR_L protocols.

- **Setup**. Challenger samples a vector of passwords $\mathsf{PW} \leftarrow_{\$} \mathcal{PW}$ for each user of the system, and the challenger and $\mathcal{A}$ invoke $\mathsf{SR}_L.\mathsf{KeyGen}()$.
- **Queries**. At any point during the game (before or after the challenge), $\mathcal{A}$ can make the following oracle queries:
  - $\mathsf{Backup}(\mathsf{uid}, \mathsf{msg})$: challenger plays the role of honest client $\mathcal{P}_{\mathsf{uid}}$ and invokes $\mathsf{SR}_L.\mathsf{Backup}(\mathsf{uid}, \mathsf{PW}[\mathsf{uid}], \mathsf{msg})$.
  - $\mathsf{Recover}(\mathsf{uid})$: challenger plays the role of honest client $\mathcal{P}_{\mathsf{uid}}$ and invokes $\mathsf{SR}_L.\mathsf{Recover}(\mathsf{uid}, \mathsf{PW}[\mathsf{uid}])$.
  - $\mathsf{Corrupt}(\mathsf{uid})$: $\mathcal{A}$ receives $\mathsf{PW}[\mathsf{uid}]$ from the challenger and it now controls client $\mathcal{P}_{\mathsf{uid}}$.
- **Challenge**. The adversary sends two messages $\mathsf{msg}_0, \mathsf{msg}_1$ s.t. $|\mathsf{msg}_0| = |\mathsf{msg}_1|$ and a user identity uid^* to the challenger. The challenger samples a random bit $b \leftarrow_{\$} \{0,1\}$ and calls $\mathsf{Backup}(\mathsf{uid}^*, \mathsf{msg}_b)$.
- **Output**. Finally, the adversary outputs a bit b' and wins the game if $b = b'$ and no $\mathsf{Corrupt}$ queries were made for uid^*.

Figure 14. Privacy of the key recovery scheme.

that these dealings are consistent with the share commitments for those committee members. It then interpolates $\mathsf{scmt}_{\mathsf{next}}$ from the current handover and sends $\langle \mathsf{Publish}, \mathsf{next} \| \text{“scmt”}, \mathsf{scmt}_{\mathsf{next}} \rangle$ to $\mathcal{F}^{\mathsf{tlog}}$ to publish the share-commitments for $\mathcal{C}_{\mathsf{next}}$. Finally, $\mathcal{S}$ forwards only the lite-dealings corresponding to dealings that are both consistent and valid to $\mathcal{C}_{\mathsf{next}}$ (t valid and consistent dealings are sufficient).

A.6. Security of the Secret Recovery Scheme

In this section, we prove privacy (Appendix A.6.1) and correctness (Appendix A.6.2) of our secret recovery scheme.

A.6.1. Privacy of the secret recovery scheme. The secrets in our system are secured using passwords which do not follow a uniform distribution over the password space $\mathbb{P}$. Thus, inspired by prior works on password-based primitives [19], [24], [26], we define our privacy game $\mathbf{G}^{\mathsf{SR,priv}}_{\mathcal{PW},L}$ w.r.t. a password distribution space $\mathcal{PW}$ that outputs a password vector, containing a password from $\mathbb{P}$ for each user in the system. Our privacy game (Figure 14) allows the adversary to corrupt the server and an arbitrary subset of clients except one honest client $\mathcal{P}_{\mathsf{uid}^*}$. The adversary can also ask $\mathcal{P}_{\mathsf{uid}^*}$ and other honest clients to perform arbitrary sequences of backups and recoveries. We define a scheme to be private if with all these capabilities, the best an adversary can do to win the game is guess the password within L attempts given knowledge of distribution $\mathcal{PW}$. Now we formally state our definition.

Password Guessing Game $\mathbf{G}^{\mathsf{PG}}_{\mathcal{PW},L}$

This game is parameterized by a password distribution $\mathcal{PW}$ and password guess limit L, where $\mathcal{PW}$ outputs a vector containing a password for each user.

- **Setup**. Challenger samples a vector of passwords for each user of the system: $\mathsf{PW} \leftarrow_{\$} \mathcal{PW}$.
- **Queries**. At any point after the setup (before or after the challenge), $\mathcal{A}$ can make the following queries:
  - $\mathsf{Guess}(\mathsf{uid}, \mathsf{pwd})$: challenger sends $\top$ to $\mathcal{A}$ if $\mathsf{pwd} = \mathsf{PW}[\mathsf{uid}]$ and $\bot$ otherwise.
  - $\mathsf{Reveal}(\mathsf{uid})$: $\mathcal{A}$ receives $\mathsf{PW}[\mathsf{uid}]$ from the challenger.
- **Challenge**. $\mathcal{A}$ sends uid^* to the challenger.
- **Output**. Finally, $\mathcal{A}$ outputs the password guess pwd^*. If $\mathsf{pwd}^* = \mathsf{PW}[\mathsf{uid}^*]$, and $\mathcal{A}$ made $\leq L$ Guess queries and no Reveal queries for uid^*, $\mathcal{A}$ wins the game.

Figure 15. Password Guessing Game

Definition 4. Let $\mathsf{SR}_L = (\mathsf{KeyGen}, \mathsf{Backup}, \mathsf{Recover})$ parameterized by security parameter λ and guess limit L be a secret recovery scheme and let $\mathbf{G}^{\mathsf{SR,priv}}_{\mathcal{PW},L}$ be the secret recovery privacy game (Figure 14) for SR_L w.r.t. a password distribution $\mathcal{PW}$. We define the advantage of an adversary $\mathcal{A}$ playing this game as follows: $\mathbf{Adv}^{\mathsf{SR,priv}}_{\mathcal{PW},L}(\mathcal{A}) := | \Pr[\mathcal{A} \text{ wins } \mathbf{G}^{\mathsf{SR,priv}}_{\mathcal{PW},L}] - 1/2 |$.

Let $\mathbf{Adv}^{\mathsf{PG}}_{\mathcal{PW},L}(\mathcal{B}^{\mathsf{PG}})$ denote the winning probability of an adversary $\mathcal{B}^{\mathsf{PG}}$ in the password-guessing game $\mathbf{G}^{\mathsf{PG}}_{\mathcal{PW},L}$ (Figure 15). Then SR_L is a *private* secret recovery scheme if $\forall$ PPT adversaries $\mathcal{A}$, $\exists$ $\mathcal{B}^{\mathsf{PG}}$ s.t. $\mathbf{Adv}^{\mathsf{SR,priv}}_{\mathcal{PW},L}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{PG}}_{\mathcal{PW},L}(\mathcal{B}^{\mathsf{PG}}) + \mathrm{negl}(\lambda)$.

Theorem 5. *The secret recovery scheme (Figure 3) is private as per Definition 4 for any PPT adversary that corrupts the server $\mathcal{S}$ assuming the $\mathcal{F}^{\mathsf{KM}}$ functionality from Figure 2.*

Proof. Consider the following sequence of hybrids, where the difference in winning probability between consecutive hybrids is negligible in λ unless stated otherwise:

- H_0 **(The real game $\mathbf{G}^{\mathsf{SR,priv}}_{\mathcal{PW},L}$)**: this is the same as the challenger and the adversary $\mathcal{A}$ playing the privacy game, where the invocations to the secret recovery scheme are performed as specified by the protocol. Since the challenger plays the role of honest parties, it knows msk of the IBE scheme (on behalf of $\mathcal{F}^{\mathsf{KM}}$) and the trapdoor td w.r.t. the honestly generated CRS for the zkSNARK [121].
- H_1 **(Zero-knowledge of zkSNARK [121])**: the challenger replaces honest client's zkSNARK proofs with simulated proofs using the trapdoor td.
- H_2 **(The blind is a one-time pad)**: the challenger replaces rqst for honest clients with a random $\mathbb{G}_2$ element. Since the challenger already replaced the zkSNARK proof with a simulated proof, it can generate a proof for a random rqst without knowing the witness. This is identical to the previous hybrid from the adversary's perspective because the blind r perfectly hides the recovery request ($\mathbb{G}_2$ is of

prime-order). At this point, the honest client's recovery requests do not reveal any additional information besides the uid.

- H_3 **(Invariant Soundness of authenticated dictionary [144])**: the challenger aborts if $\mathcal{F}^{\mathsf{KM}}$ responds to $> L$ Recover requests for any uid.
- H_4 **(Weak Simulation-extractability of zk-SNARK [121])**: the challenger aborts if it can not extract (pwd, r) from the recovery request made by the adversary.
- H_5 **(anon-IND-ID-CCA prop. of the IBE scheme; Appendix A.4)**:
  - The challenger aborts if the adversary makes a recovery request with inputs $\mathsf{id}^* = \mathsf{uid}^* \| \mathsf{PW}[\mathsf{uid}^*]$. It can detect the abort condition because it knows $(\mathsf{uid}, \mathsf{pwd})$ for all of adversary's recovery requests (from H_4).
  - The challenger samples a random identity id' and AE key k', and replaces the IBE ciphertext $\mathsf{ct}_{\mathsf{ibe}}$ in the challenge backup with $\mathsf{ct}'_{\mathsf{ibe}} \leftarrow \mathsf{IBE.Encrypt}(\mathsf{pk}, \mathsf{id}', k')$.

  Note that this hybrid contains no information about $\mathsf{PW}[\mathsf{uid}^*]$ and the AE key k used to encrypt msg_b. In Lemma 6, we prove that (i) adversary's probability of guessing the right password in H_4 and H_5 is negligibly different, (ii) the adversary can't distinguish the hybrids if it does not guess the password, and (iii) adversary's probability of guessing the right password in H_5 is $\mathbf{Adv}^{\mathsf{PG}}_{\mathcal{PW},L}$. All three combined show that adversary's advantage in distinguishing H_4 and H_5 is $\leq \mathbf{Adv}^{\mathsf{PG}}_{\mathcal{PW},L} + \mathrm{negl}(\lambda)$.
- H_6 **(IND-CCA2 prop. of Authenticated Encryption [23])**: the challenger replaces msg_b with a random message msg' in the AE ciphertext $\mathsf{ct}_{\mathsf{ae}}$ of the challenge backup.

Overall, this shows that the adversary's probability of distinguishing H_0 and H_6 is $\leq \mathbf{Adv}^{\mathsf{PG}}_{\mathcal{PW},L} + \mathrm{negl}(\lambda)$. Since the final hybrid contains no information about b, the adversary can only guess the bit and this completes the proof. □

Lemma 6. *Let W_4 and W_5 denote the adversary $\mathcal{A}$'s winning probability in Hybrid H_4 and H_5 from the proof of Theorem 5. $\forall \mathcal{A}$, $\exists$ $\mathcal{B}^{\mathsf{PG}}$ s.t. $|\Pr[W_4] - \Pr[W_5]| \leq \mathbf{Adv}^{\mathsf{PG}}_{\mathcal{PW},L}(\mathcal{B}^{\mathsf{PG}}) + \mathrm{negl}(\lambda)$.*

Proof. Let E_i denote the event that the adversary makes a recovery request for id^* in hybrid H_i. Then we can write:

$$\Pr[W_4] = \Pr[E_4] \cdot \Pr[W_4|E_4] + \Pr[\neg E_4] \cdot \Pr[W_4|\neg E_4]$$

Similarly, we can write:

$$\begin{aligned}\Pr[W_5] &= \Pr[E_5] \cdot \Pr[W_5|E_5] + \Pr[\neg E_5] \cdot \Pr[W_5|\neg E_5] \\ &= \Pr[\neg E_5] \cdot \Pr[W_5|\neg E_5] \qquad (\Pr[W_5|E_5] = 0)\end{aligned}$$

Now, we argue the following: (i) $|\Pr[E_4] - \Pr[E_5]| \leq \mathrm{negl}(\lambda)$, (ii) $|\Pr[W_4|\neg E_4] - \Pr[W_5|\neg E_5]| \leq \mathrm{negl}(\lambda)$, and (iii) there exists $\mathcal{B}^{\mathsf{PG}}$ such that $\Pr[E_5] = \mathbf{Adv}^{\mathsf{PG}}_{\mathcal{PW},L}(\mathcal{A})$, which implies:

$$\begin{aligned}&|\Pr[W_4] - \Pr[W_5]| \\ &\quad\leq |\Pr[E_4] \cdot \Pr[W_4|E_4]| \\ &\qquad + |\Pr[\neg E_4] \cdot \Pr[W_4|\neg E_4] - \Pr[\neg E_5] \cdot \Pr[W_5|\neg E_5]| \\ &\quad\leq |\Pr[E_4]| + |\Pr[\neg E_4] \cdot (\Pr[W_4|\neg E_4] - \Pr[W_5|\neg E_5])| \\ &\qquad + |(\Pr[\neg E_4] - \Pr[\neg E_5]) \cdot \Pr[W_5|\neg E_5]| \\ &\quad\leq |\Pr[E_5] + (\Pr[E_4] - \Pr[E_5])| + |\Pr[\neg E_4] \cdot \mathrm{negl}(\lambda)| \\ &\qquad + |(\Pr[E_5] - \Pr[E_4]) \cdot \Pr[W_5|\neg E_5]| \\ &\quad\leq \Pr[E_5] + \mathrm{negl}(\lambda) \leq \mathbf{Adv}^{\mathsf{PG}}_{\mathcal{PW},L}(\mathcal{B}^{\mathsf{PG}}) + \mathrm{negl}(\lambda)\end{aligned}$$

$|\Pr[E_4] - \Pr[E_5]| \leq \mathrm{negl}(\lambda)$: To prove this, we consider a modified hybrid H'_i which is the same as H_i except the adversary $\mathcal{A}$ wins if and only if event E_i happens. We show that if $\mathcal{A}$ has a non-negligible advantage ϵ in winning in H'_4 compared to H'_5, then we can construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ in a black-box way to win the anon-IND-ID-CCA game of the IBE scheme, leading to a contradiction. $\mathcal{B}$ works as follows: it behaves the same way as the H'_4/H'_5 challenger except it doesn't know msk of the IBE scheme. Nevertheless, it can perform the role of $\mathcal{F}^{\mathsf{KM}}$ by querying the (outside) anon-IND-ID-CCA challenger through extraction and decryption queries. Even though the recovery requests to $\mathcal{F}^{\mathsf{KM}}$ are blinded, the challenger knows the blind r and $\mathsf{id} = \mathsf{pwd} \| \mathsf{uid}$ for each recovery request in both hybrid H'_4/H'_5. Thus, it can simply ask for extraction on id from the outside challenger and then blind the response with r before forwarding it to $\mathcal{A}$. When $\mathcal{A}$ chooses a challenge uid^* and messages msg_0 and msg_1, $\mathcal{B}$ does the following: it samples a bit b, a random identity id', and random AE keys k, k'. It then encrypts msg_b with key k to get $\mathsf{ct}^*_{\mathsf{ae}}$ and chooses the challenge identities and messages for the outside challenger as $\mathsf{id}_0 = \mathsf{id}^*, \mathsf{id}_1 = \mathsf{id}', m_0 = k, m_1 = k'$. The outside challenger chooses a bit b', and sends the challenge ciphertext $\mathsf{ct}^*_{\mathsf{ibe}}$ to $\mathcal{B}$. In response to $\mathcal{A}$'s challenge, $\mathcal{B}$ sets $\mathsf{ct}_{\mathsf{uid}^*} = (\mathsf{ct}^*_{\mathsf{ibe}}, \mathsf{ct}^*_{\mathsf{ae}})$ as the ciphertext during challenge backup. Note that from the perspective of $\mathcal{A}$, if the outside challenger picked $b' = 0$, then we're exactly in H'_4 and if $b' = 1$, then we're exactly in H'_5. If event E happens, i.e., $\mathcal{A}$ makes a recovery request for id^*, then $\mathcal{B}$ outputs 0, else it outputs 1. If the winning probability in H'_4 and H'_5 differs by non-negligible probability ϵ, then it is immediate that the probability that $\mathcal{B}$ outputs 1 in case $b' = 0$ and $b' = 1$ also differs by non-negligible probability.

$|\Pr[W_4|\neg E_4] - \Pr[W_5|\neg E_5]| \leq \mathrm{negl}(\lambda)$: This proof follows through a similar reduction to the previous one. We can construct $\mathcal{B}$ that wins the anon-IND-ID-CCA game using an adversary $\mathcal{A}$ that has a non-negligible advantage in winning in H_4 over H_5 (conditioned on not making a recovery request for id^*). $\mathcal{B}$ works the same as the previous reduction, except $\mathcal{B}$ outputs 0 if $\mathcal{A}$ guesses b correctly, else it outputs 1. Since the winning condition requires not querying $\mathsf{id}_0 = \mathsf{id}^*$ and $\mathcal{A}$ has negligible probability of guessing $\mathsf{id}_1 = \mathsf{id}'$, $\mathcal{B}$ never has to query outside challenger on $\mathsf{id}_0, \mathsf{id}_1$. Finally, if the winning probability in H_4 and H_5 differs by non-negligible probability ϵ, then it is immediate that the

probability that $\mathcal{B}$ outputs 1 in case $b' = 0$ and $b' = 1$ also differs by non-negligible probability.

$\Pr[E_5] = \mathbf{Adv}^{\mathsf{PG}}_{\mathcal{PW},L}(\mathcal{B}^{\mathsf{PG}})$: Since H_5 contains no information about the password $\mathsf{PW}[\mathsf{uid}^*]$, it is easy to show a reduction to the password guessing game. In particular, we can show that if $\mathcal{A}$ makes a recover request for id^* with ϵ probability, then we can construct adversary $\mathcal{B}^{\mathsf{PG}}$ that has the same probability to win the password guessing game. $\mathcal{B}^{\mathsf{PG}}$ works as follows: it faithfully performs the role of challenger in H_5, except it doesn't sample PW during Setup, and instead, relies on queries of the password guessing game to answer the queries of the privacy game. The Corrupt queries correspond directly to the Reveal query in the password guessing game, and the Backup and Recover queries can also be answered for $\mathsf{uid} \neq \mathsf{uid}^*$ through the Reveal query. To answer Backup and Recover queries on uid^* in H_5, note that we don't need to know $\mathsf{PW}[\mathsf{uid}^*]$. When $\mathcal{A}$ performs recovery request for $(\mathsf{pwd}, \mathsf{uid}^*)$, we use the Guess query to see if it is the correct password. If Guess returns $\top$, we know that event E_5 occurred, and we return the corresponding password as the output of the password guessing game. Since $\Pr[E_5] = \epsilon$, $\mathcal{B}^{\mathsf{PG}}$ should also win the game by probability ϵ, and thus we arrive at a contradiction. □

A.6.2. Correctness of the secret recovery scheme. We define the correctness game $\mathbf{G}^{\mathsf{SR},\mathsf{corr}}_L$ for a secret recovery scheme SR in Figure 16. The game models that if the server is honest (meaning we have availability), for all $\mathsf{msg}^* \in \{0,1\}^*, \mathsf{uid}^* \in \mathbb{U}, \mathsf{pwd}^* \in \mathbb{P}$, honest user $\mathcal{P}_{\mathsf{uid}^*}$ will always be able to retrieve its secret message msg^* provided that it guesses the right password within L attempts. Here, like prior work on password-based secret recovery, we assume that an honest server will employ alternative authentication mechanisms to prevent the attacker from exhausting the guess limit for an honest user (see "Authentication Server" in [55]).

Definition 7. Let $\mathsf{SR}_L = (\mathsf{KeyGen}, \mathsf{Backup}, \mathsf{Recover})$ be a secret recovery scheme parameterized by security parameter λ and guess limit L. Let $\mathbf{G}^{\mathsf{SR},\mathsf{corr}}_L$ be the secret key recovery correctness game (Figure 16) for SR_L. We define the advantage of an adversary $\mathcal{A}$ playing this game as follows: $\mathbf{Adv}^{\mathsf{SR},\mathsf{corr}}_L(\mathcal{A}) := \Pr[\mathcal{A} \text{ wins } \mathbf{G}^{\mathsf{SR},\mathsf{corr}}_L]$

Then SR_L is a *correct* key recovery scheme if for all PPT adversaries $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{SR},\mathsf{corr}}_L(\mathcal{A}) = 0$.

Theorem 8. *The key recovery scheme (defined in Figure 3) is correct as per Definition 7 for any PPT adversary assuming the $\mathcal{F}^{\mathsf{KM}}$ functionality from Figure 2 and the server $\mathcal{S}$ is honest.*

The proof follows straightforwardly from the correctness of the IBE scheme and the AE scheme, and the completeness of zkSNARK.

A.7. Security of the KM protocol

In this section, we prove that the KM protocol described in Appendix A.2 securely realizes the $\mathcal{F}^{\mathsf{KM}}$ functionality described in Figure 2.

Correctness Game $\mathbf{G}^{\mathsf{SR},\mathsf{corr}}_L$ for Secret Recovery

This game is parameterized by password guess limit L and a secret recovery scheme $\mathsf{SR}_L = (\mathsf{KeyGen}, \mathsf{Backup}, \mathsf{Recover})$. The challenger plays the role of all parties in SR_L protocols.

- **Setup**. same as protocol; challenger sends pk to $\mathcal{A}$.
- **Challenge**. The adversary sends msg^*, $\mathsf{uid}^* \in \mathbb{U}$, $\mathsf{pwd}^* \in \mathbb{P}$, and L password guesses $\{\mathsf{pwd}_i\}_{i\in[L]}$ to the challenger. On behalf of $\mathcal{P}_{\mathsf{uid}^*}$, the challenger calls $\mathsf{SR}_L.\mathsf{Backup}(\mathsf{uid}^*, \mathsf{pwd}^*, \mathsf{msg}^*)$, and then for each password guess pwd_i, the challenger calls $\mathsf{msg}_i \leftarrow \mathsf{SR}_L.\mathsf{Recover}(\mathsf{uid}^*, \mathsf{pwd}_i)$.
- **Output**. The adversary wins the game if $\exists\ i \in [L]$ s.t. $\mathsf{pwd}_i = \mathsf{pwd}^*$ and $\mathsf{msg}_i \neq \mathsf{msg}^*$.

Figure 16. Correctness of the secret recovery scheme.

A.7.1. Threat Model. We begin by defining the capabilities of the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$: $\mathcal{Z}$ and $\mathcal{A}$ interact throughout the protocol. $\mathcal{Z}$ is responsible for choosing honest party inputs, when they make protocol or ideal functionality API invocations, and it gets the honest party outputs. $\mathcal{A}$ can corrupt at most κ fraction of the total clients. It is mobile and the corruption it makes take effect after δ time. $\mathcal{A}$ can arbitrarily schedule and drop messages with the constraint that for all e, the messages exchanged among the following honest parties during epochs e through $e+2$ are delivered within δ time from the start of epoch e: the honest server, the ideal functionalities, and $1-\phi$ fraction of honest clients (see §2.4). We model this by assuming that all parties (including ideal functionalities) are connected through a star topology, at the center of which is a router that $\mathcal{A}$ controls.

We assume a constraint on $\mathcal{A}$'s ability to control the network for simplicity. If this assumption is violated, we claim that this only affects availability and not privacy (similarly to how a corrupted server only affects availability). To see why, note that our modeling allows $\mathcal{A}$ to corrupt the server $\mathcal{S}$, which in the real protocol serves the same role as the router, and as such, can arbitrarily control message delivery and drop messages without any constraints. Nevertheless, we introduce the router in our model to capture the security of our protocol in case the server is not corrupted.

A.7.2. Definition. Before we state the formal definition, we first define the ideal and real world experiments. Both experiments are parameterized by the security parameter λ, the password guess-limit L, and the auxiliary input to the environment $z \in \{0,1\}^{\mathrm{poly}(\lambda)}$.

The ideal experiment. In the ideal world, the honest parties interact with $\mathcal{F}^{\mathsf{KM}}$ and the adversary $\mathcal{A}$ interacts with the environment $\mathcal{Z}$ and the ideal-world adversary (or simulator) Sim which simulates the honest parties for $\mathcal{A}$. $\mathcal{A}$ controls

the corrupted parties. $\mathcal{Z}$ specifies the inputs of the honest parties. If the server $\mathcal{S}$ is honest, $\mathcal{Z}$ adaptively specifies when KeyGen and the subsequent Recover APIs are called and the inputs to the Recover invocation. The honest server adds any Recover calls to a queue and executes them in order after calling KeyGen. Although the real protocol processes a batch of requests at a time, we assume a single request is processed at a time in Figure 2 and in this proof for simplicity. $\mathcal{Z}$ also specifies which honest parties $\mathcal{P}$ will call the PublicKey API and it can arbitrarily interleave these requests. At the end of the experiment ($\mathcal{Z}$ decides when), $\mathcal{Z}$ outputs a bit, which we denote by $\mathrm{IDEAL}_{\mathcal{F}^{\mathsf{KM}},\mathsf{Sim},\mathcal{Z}}(\lambda, L, z)$.

The real experiment. In the real world, the (honest) parties send messages to each other and the ideal functionalities $\mathcal{F}^{\mathsf{tlog}}, \mathcal{F}^{\mathsf{PKI}}$ according to a specified key manager protocol KM. The adversary $\mathcal{A}$ interacts with the honest parties and $\mathcal{F}^{\mathsf{tlog}}, \mathcal{F}^{\mathsf{PKI}}$ in this world in addition to the environment $\mathcal{Z}$. Identical to the ideal experiment, $\mathcal{A}$ controls the corrupted parties, and $\mathcal{Z}$ specifies the inputs of honest parties with the same constraints and looks at their outputs. At the end of the experiment, $\mathcal{Z}$ outputs a bit $\mathrm{REAL}^{\mathcal{F}^{\mathsf{tlog}},\mathcal{F}^{\mathsf{PKI}}}_{\mathsf{KM},\mathcal{A},\mathcal{Z}}(\lambda, L, z)$.

Definition 9. Let KM be a key manager protocol in the $(\mathcal{F}^{\mathsf{tlog}}, \mathcal{F}^{\mathsf{PKI}})$-hybrid model. Then KM securely realizes $\mathcal{F}^{\mathsf{KM}}$ in the $(\mathcal{F}^{\mathsf{tlog}}, \mathcal{F}^{\mathsf{PKI}})$-hybrid model if for all PPT adversaries $\mathcal{A}$, there exists a PPT simulator Sim such that for every PPT environment $\mathcal{Z}$ subject to constraints defined in Appendix A.7.1, and every $\lambda \in \mathbb{N}$, $L \in \mathbb{N}$, and $z \in \{0,1\}^{\mathrm{poly}(\lambda)}$, we have:

$$\Big|\Pr[\mathrm{IDEAL}_{\mathcal{F}^{\mathsf{KM}},\mathsf{Sim},\mathcal{Z}}(\lambda, L, z) = 1] - \Pr[\mathrm{REAL}^{\mathcal{F}^{\mathsf{tlog}},\mathcal{F}^{\mathsf{PKI}}}_{\mathsf{KM},\mathcal{A},\mathcal{Z}}(\lambda, L, z) = 1]\Big| \leq \mathrm{negl}(\lambda)$$

Theorem 10. *As per Definition 9, the key manager protocol* KM *defined in Appendix A.2 securely realizes* $\mathcal{F}^{\mathsf{KM}}$ *in the* $(\mathcal{F}^{\mathsf{tlog}}, \mathcal{F}^{\mathsf{PKI}})$*-hybrid model.*

A.7.3. Notation for the proof (Appendix A.7.4) and simulator (Appendix A.7.5). For a committee $\mathcal{C}_{\mathsf{curr}}$, let $\mathcal{C}^C_{\mathsf{curr}}$ denote the corrupted $\mathcal{C}_{\mathsf{curr}}$-members, and similarly, let $\mathcal{C}^H_{\mathsf{curr}}$ denote the honest $\mathcal{C}_{\mathsf{curr}}$-members. We denote the shares of $\mathcal{C}_{\mathsf{curr}}$-members by $\mathsf{shares}_{\mathsf{curr}}$, and the shares chosen by $\mathcal{M}_i \in \mathcal{C}_{\mathsf{curr}}$ for $\mathcal{C}_{\mathsf{next}}$-members by $\mathsf{shares}_{i\to\mathsf{next}}$. The collection of $O(n^2)$ shares chosen by $\mathcal{C}_{\mathsf{curr}}$ members for $\mathcal{C}_{\mathsf{next}}$-members is denoted by $\mathsf{shares}_{\mathsf{curr}\to\mathsf{next}}$. Similarly, we denote the coefficients of the sharing polynomials chosen by $\mathcal{C}_{\mathsf{curr}}$-members with $\mathsf{coeffs}_{\mathsf{curr}}$. We add a superscript $\{C, H\}$ to the committee labels if we want to denote just the corrupted or honest members on that committee, respectively.

A.7.4. Proof of Theorem 10.

Proof. Now, we argue that with the simulator description above, the ideal and real experiments are indistinguishable:

- H_0 **(The real experiment)**
- H_1 **(Simulate honest parties)**: Sim simulates the honest clients, the server if it is honest, and $\mathcal{F}^{\mathsf{PKI}}$ and $\mathcal{F}^{\mathsf{tlog}}$ functionalities. Since it controls $\mathcal{F}^{\mathsf{PKI}}$, it can register the simulated honest clients with a signature key that it samples. Since it controls honest committee members, for any consecutive epochs prev and curr, Sim knows $\mathsf{shares}_{\mathsf{prev}^H\to\mathsf{curr}^C}$, i.e., random shares chosen by honest $\mathcal{C}_{\mathsf{prev}}$-members for corrupted $\mathcal{C}_{\mathsf{curr}}$-members.
- H_2 **(Pseudorandomness of VRF [76])**: in case the server is honest, Sim aborts if it receives messages from $< t$ honest committee members on any committee. This happens with negligible probability due to the pseudorandomness of VRF outputs of honest parties and parameter selection (Appendix A.1) which guarantees that $\geq t$ honest clients will always be selected with high probability.
- H_3 **(Key-binding property of authenticated dictionary [144]**: Sim aborts (happens with negligible probability) if $\mathcal{A}$ is able to include two or more sets of VRF key-pairs in the client list for the same PKI-registered $(\mathsf{uid}, \mathsf{pk}^{\mathsf{sig}})$ pair.
- H_4 **(Lemma 11)**: Sim aborts if the adversary posts a valid committee that has $\geq t$ corrupted clients. This happens with negligible probability from Lemma 11.
- H_5 **(Zero-knowledge property of NIZK [121]**: Sim replaces the following proofs with simulated proofs:
  - PKE well-formedness proof π^{enc} such that the simulated proof does not contain any information about the encryption secret key of honest clients.
  - pVSS proof π^{pv} and aVSS proof π^{ab}, such that the simulated proofs do not contain any information about the shares chosen by an honest committee member for the next committee.
  - Discrete-log equality proof π^{dleq} such that the simulated proof does not contain any information about the secret share of an honest committee member.
- H_6 **(Simulation-extractability of SE-NIZK [85])**: For any consecutive epochs prev and curr, Sim extracts the witness ($\mathsf{coeffs}_{\mathsf{prev}^C}$) underlying the dealings generated by corrupted $\mathcal{C}_{\mathsf{prev}}$-members if an honest $\mathcal{C}_{\mathsf{curr}}$-member does not abort during Reconstruct. Given $\mathsf{shares}_{\mathsf{prev}^H\to\mathsf{curr}^C}$, $\mathsf{coeffs}_{\mathsf{prev}^C}$, and $\mathsf{scmts}_{\mathsf{curr}^C}$ posted by $\mathcal{S}$, Sim interpolates $\mathsf{shares}_{\mathsf{curr}^C}$ which are consistent with $\mathsf{scmts}_{\mathsf{curr}^C}$. Note that the interpolated shares are the same across all honest $\mathcal{C}_{\mathsf{curr}}$-members that do not abort since they are checked against the same $\mathsf{scmts}_{\mathsf{curr}^C}$ posted to the transparency log. Here, we require a straight-line extractor for the (batched) Schnorr Sigma protocol. In our implementation, we make the Sigma protocol non-interactive using the Fiat-Shamir transform [66], and we can obtain a white-box straight line extractor for this NIZK in the Random Oracle Model (ROM) plus the Algebraic Group Model (AGM) through a direct extension of ideas from Fuchsbauer et al. [70]. Assuming the environment does not interact with the AGM oracle for this NIZK, which is the case for our secret recovery protocol, Kerber et al. [99] showed that white-box extraction can be composed. Alternatively, for general composition, one can replace Fiat-Shamir with the Fischlin transform [67].
- H_7 **(Soundness of NIZK [85]**: Sim aborts (happens

with negligible probability) if any of the following events happen:

- An honest committee member accepts a committee even though one of the member's encryption keys was ill-formed.
- If $\mathcal{S}$ is honest, the server accepts a pVSS proof π^{pv} even though the shares received by honest parties do not match the corresponding coefficient commitment.
- An honest $\mathcal{C}_2$-member accepts the DL-EQ proof π^{dleq} even though $\mathsf{dlog}_g(\mathsf{ccmt}_{1,i,0}) \neq \mathsf{dlog}_{g'}(\mathsf{scmt}_{1,i})$ for some $\mathcal{M}_i \in \mathcal{C}_{2^C}$.

- H_8 (**EUF-CMA property of signature [80]**): Sim aborts if an honest committee member $\mathcal{M}_i \in \mathcal{C}_{\mathsf{curr}}$ accepts a dealing from an honest committee member $\mathcal{M}_k \in \mathcal{C}_{\mathsf{prev}}$ that $\mathcal{M}_k$ did not generate for epoch curr and $\mathcal{M}_i$.
- H_9 (**mr-IND-CPA property of encryption scheme [96]**): in this hybrid, for any consecutive epochs prev, curr, next and each honest member $\mathcal{M}_i \in \mathcal{C}_{\mathsf{curr}}$, we (i) set the shares $\mathsf{shares}_{i\to\mathsf{next}^H}$ for honest $\mathcal{C}_{\mathsf{next}}$-members to all 0s, and (ii) do not abort when the decrypted share does not match the commitment for honest $\mathcal{C}_{\mathsf{prev}}$-members, and (iii) still output the same share as the previous hybrid to ensure the output consistency. Since PKI setup ensures that the encryption keys of corrupted clients are well-formed, this change is indistinguishable from the previous hybrid due to the mr-IND-CPA property of the encryption scheme.
- H_{10} (**XDH assumption in $\mathbb{G}_1$ [21]**): Sim samples a random $\mathsf{msk} \in \mathbb{F}_q$ and sets $\mathsf{pk} \leftarrow g^{\mathsf{msk}}$. Given msk and shares_{1^C} (note that $|\mathsf{shares}_{1^C}| < t$), Sim interpolates shares'_{1^H} such that when t shares out of $\mathsf{shares}_{1^C} \cup \mathsf{shares}'_{1^H}$ are combined, the result is msk. For each honest client $\mathcal{M}_i \in \mathcal{C}_1^H$ that does not abort, $\mathcal{M}_i$ re-shares $\mathsf{shares}'_{1^H,i}$ instead of $\mathsf{shares}_{1^H_i}$ to $\mathcal{C}_2$. Accordingly, now $\mathcal{M}_i$ provides a simulated π^{dleq} proof on a false statement, which shows that $\mathsf{dlog}_g(g^{\mathsf{shares}_{1^H,i}}) = \mathsf{dlog}_{g'}(g'^{\mathsf{shares}_{1^H,i}})$ even though they are independent of each other. The adversary will have access to false DL-EQ proofs now, but it is okay because the DL-EQ proof we use are simulation-sound [64].
- H_{11} (**The ideal experiment**): in this hybrid, we introduce the ideal functionality $\mathcal{F}^{\mathsf{KM}}$. Honest parties only talk to $\mathcal{F}^{\mathsf{KM}}$ and the Sim talks to $\mathcal{F}^{\mathsf{KM}}$ on behalf of the corrupted parties. Sim simulates the honest parties for $\mathcal{A}$ and it interacts with $\mathcal{A}$ and $\mathcal{F}^{\mathsf{KM}}$ as detailed in the previous section. Note that this is the same as the ideal experiment. Now, we argue that this hybrid is indistinguishable from the previous hybrid.

  It is easy to see that the simulated honest parties' behavior is identical to hybrid H_{10} from the adversary's perspective. We first argue that the simulator needs access to the same information in both hybrids:

  - In both hybrids, for any honest committee member $\mathcal{M}_i \in \mathcal{C}_{\mathsf{curr}}$, simulated $\mathcal{M}_i$ does not need to know its share $\mathsf{shares}_{\mathsf{curr},i}$. All information sent by $\mathcal{M}_i$ can be simulated using $\mathsf{sk}_i^{\mathsf{enc}}$ (chosen by Sim), $< t$ shares of corrupted committee members for any epoch curr (i.e., $\mathsf{shares}_{\mathsf{curr}^C}$), and the expected public key pk and the recovery responses resp.
  - The simulator does not need secret information to simulate the honest clients, the server, parties fetching pk, and the ideal functionalities.

  Now, we argue that the output distribution of honest parties is also identical:

  - The simulator can change the public key pk in H_{10} to the one output by $\mathcal{F}^{\mathsf{KM}}$. Since $\mathcal{F}^{\mathsf{KM}}$ generates pk the same way as the Sim does in hybrid H_{10}, this change is indistinguishable. So, the public key output by honest server in H_{10} and the ideal experiment are the same.
  - Similarly, Sim can change the responses of honest clients in H_{10} such that they match the recovery response resp output by $\mathcal{F}^{\mathsf{KM}}$ to ensure that the response output by honest server in H_{10} and the ideal experiment are the same.
  - Whenever an honest party tries to fetch pk in H_{10}, $\mathcal{F}^{\mathsf{KM}}$ informs simulator and it does the same in the ideal experiment with a simulated party. If this party fails to retrieve pk in H_{10}, Sim ensures the same in the ideal experiment by denying that request on behalf of $\mathcal{A}$.
  - In the ideal experiment, if the server is honest, it eventually gets the recovery responses. In H_{10}, the honest server also always gets $\geq t$ valid responses because there are $\geq t$ honest committee members in each committee and their responses are delivered before they time out due to the network assumption we make (see Appendix A.7.1).
  - If during any epoch curr, a corrupted server $\mathcal{S}$ deviates from the protocol such that enough honest $\mathcal{C}_{\mathsf{curr}}$-members time out or abort, the server in H_{10} will not get $\geq t$ valid handovers. Consequently, the honest $\mathcal{C}_{\mathsf{next}}$-members will also abort because they verify that there are $\geq t$ valid handovers. This process will go on for all subsequent committees. As such, when Sim detects this condition, it sends $\langle\mathsf{Terminate}\rangle$ to $\mathcal{F}^{\mathsf{KM}}$ because no recoveries can be made this point onwards.
  - Finally, Sim informs $\mathcal{F}^{\mathsf{KM}}$ to send outputs when the simulated parties get them in the ideal experiment. So, in the previous hybrid and the ideal experiment, honest parties get their outputs at the same time.

  That concludes the proof and we describe the final simulator in Appendix A.7.5. □

Lemma 11. *If an adversary $\mathcal{A}$ that controls the server and at most $\kappa\%$ of clients can output a committee that has $\geq t$ corrupted clients with non-negligible probability, then we can construct an adversary $\mathcal{B}$ that wins the VRF unbiasability game ([76], Definition 8) with non-negligible probability.*

Proof. Without loss of generality, we will only focus on the fairness of committee selection during a single epoch e. We can then use the hybrid argument to extend it to

polynomially many epochs. Let $\mathcal{A}_e$ denote the adversary that outputs a committee in epoch e with $\geq t$ corrupted clients with non-negligible probability $\epsilon(\lambda)$. Then we construct an adversary $\mathcal{B}$ that wins the VRF unbiasability game with non-negligible probability. $\mathcal{B}$ simulates the random oracle H used by $\mathcal{A}_e$ to build the Merkle tree, the random oracle H_{seed}, and the ideal functionalities $\mathcal{F}^{\mathsf{tlog}}$ and $\mathcal{F}^{\mathsf{PKI}}$. It works as follows:

- Let Q be the upper bound on the polynomially-many queries made by $\mathcal{A}_e$ to oracle H. $\mathcal{B}$ guesses the index $i \in [Q]$ that will correspond to the root rt of the Merkle tree. Without loss of generality, we can assume that the final root rt chosen by $\mathcal{A}_e$ will be in the H queries. If $\mathcal{A}_e$ does not query it, we can always construct a wrapper adversary that makes this query when $\mathcal{A}_e$ publishes rt to $\mathcal{F}^{\mathsf{tlog}}$ at the end of committee selection.
- It treats the i-th query to H as a root and infers the entire tree based on the other queries made to H. Since a random oracle is collision-resistant, there will be at most one query that is the pre-image of the root, and we can extend this argument all the way to the leaves. If $\mathcal{B}$ cannot construct the tree all the way to the leaves that has the client keys, $\mathcal{B}$ aborts. Else, $\mathcal{B}$ gets to extract c VRF public keys $\{\mathsf{pk}_j^{\mathsf{vrf}}\}_{j\in[c]}$ for the identifiers corresponding to the c corrupted clients in the system. $\mathcal{B}$ knows these identifiers because it simulates $\mathcal{F}^{\mathsf{PKI}}$.
- $\mathcal{B}$ sends these VRF keys to the VRF game challenger along with a PPT computable monotone predicate p that is defined as follows: $p(y_1, \ldots, y_c) = \mathbf{1}\{\sum_{j\in[c]} \mathbf{1}\{y_j \neq \perp \wedge\ y_j \leq \frac{n}{N}\} \geq t\}$, i.e., whether the number of keys on the committee exceed the threshold t. $\mathcal{B}$ receives a seed x from the outside challenger, and it programs the output of H_{seed} at input (rt, e) to x.
- Finally, $\mathcal{A}_e$ publishes the committee $\mathcal{C}_e$ on $\mathcal{F}^{\mathsf{tlog}}$. Let rt' be the root published. If the roots don't match, $\mathcal{B}$ aborts. If the honest parties abort while verifying this committee or it has less than t corrupted members, $\mathcal{B}$ aborts. Else, $\mathcal{B}$ takes the VRF outputs and proofs for corrupted parties in $\mathcal{C}_e^C$, puts $\perp$ in place of the output and proof for corrupted parties not on the committee, and forwards them to the outside challenger. This concludes the reduction.

To argue that $\mathcal{B}$ has a non-negligible advantage, observe that when the unbiasability experiment is instantiated with $b = 1$ and p is evaluated on uniformly random values, we have already proved for our parameters that p returns 1 with negligible probability (Appendix A.1). When $b = 0$, note that if $\mathcal{B}$ does not abort, then p returns 1. Thus, we just need to argue that $\mathcal{B}$ does not abort with non-negligible probability. This argument is simple: with probability $1/Q$ which is polynomial in security parameter, we guess the right root. The adversary $\mathcal{A}_e$'s view is identical in the real protocol and in this reduction, and it outputs a committee with $\geq t$ corrupted clients with non-negligible probability $\epsilon(\lambda)$. Thus, with non-negligible probability $\epsilon(\lambda)/Q$, $\mathcal{B}$ does not abort and it wins the VRF unbiasability game. □

A.7.5. Simulator Description. To describe the simulator Sim, we only mention the deltas from the actual protocol. If the server $\mathcal{S}$ is honest, it always follows the actual protocol. Thus, we only focus on how the honest clients which are simulated by Sim behave differently.

$\{\mathcal{C}_e, \perp\} \leftarrow \mathsf{Sim.CommitteeSelection}(e)$:

Same as $\mathsf{ECPSS.CommitteeSelection}(e)$ except:

- **(Client) Setup**: π^{enc} is replaced with a simulated proof.
- **(Client) Get Committee**: Sim aborts if the simulated client does not abort and the committee has $\geq t$ corrupted members.
- If $\mathcal{S}$ is honest, Sim also aborts if it does not receive messages from $\geq t$ honest committee members for any committee.

$\mathsf{shares}_{i\to\mathsf{next}^C} \leftarrow \mathsf{Sim.Share}(\mathsf{scmt}_{\mathsf{curr},i}, \mathcal{C}_{\mathsf{next}})$:

Same as $\mathsf{ECPSS.Share}(\mathsf{shares}_{\mathsf{curr},i}, \mathcal{C}_{\mathsf{next}})$ except Sim does not know $\mathsf{shares}_{\mathsf{curr},i}$ (it only knows commitment to $\mathsf{shares}_{\mathsf{curr},i}$, i.e., $\mathsf{scmt}_{\mathsf{curr},i} = g^{\mathsf{shares}_{\mathsf{curr},i}}$), and as such, it makes the following changes to $\mathsf{saVSS.Deal}$ on behalf of $\mathcal{M}_i \in \mathcal{C}_{\mathsf{curr}}$:

- Sim chooses shares $\mathsf{shares}_{i\to\mathsf{next}^C}$ (w.l.o.g., $|\mathsf{shares}_{i\to\mathsf{next}^C}| = t-1$) for corrupted $\mathcal{C}_{\mathsf{next}}$-members on behalf of simulated $\mathcal{M}_i$ at random. From these shares, it gets the corresponding share-commitments $\mathsf{scmts}_{i\to\mathsf{next}^C}$. Given $\mathsf{scmts}_{i\to\mathsf{next}^C}$ and $\mathsf{scmt}_{\mathsf{curr},i}$, Sim interpolates the commitment $\mathsf{ccmt}_{\mathsf{curr},i}$ to coefficients of the sharing polynomial for $\mathcal{M}_i \in \mathcal{C}_{\mathsf{curr}}$ (note that $\mathsf{ccmt}_{\mathsf{curr},i,0} = \mathsf{scmt}_{\mathsf{curr},i}$).
- It sets the shares $\mathsf{shares}_{i\to\mathsf{next}^H}$ of honest $\mathcal{C}_{\mathsf{next}}$-members to all 0.
- Sim generates a simulated π^{ab} proof using the corresponding zero-knowledge simulator w.r.t. cmt. Note that it does not need to know the polynomial coefficients since it's a simulated proof.
- It generates a simulated π^{pv} proof using the corresponding zero-knowledge simulator for the fake statement where cmt does not match the encrypted shares.
- Finally, Sim returns $\mathsf{shares}_{i\to\mathsf{next}^C}$.

$\{\mathsf{shares}_{\mathsf{curr}^C}, \perp\} \leftarrow \mathsf{Sim.Reconstruct}(\mathsf{shares}_{\mathsf{prev}^H\to\mathsf{curr}^C}, \mathcal{C}_{\mathsf{prev}})$:

Same as $\mathsf{ECPSS.Reconstruct}(\mathcal{C}_{\mathsf{prev}})$ except:

- Simulated committee member $\mathcal{M}_i \in \mathcal{C}_{\mathsf{curr}}$ does not abort even if the decrypted share does not match the commitment for dealings from honest committee members (given other checks pass).
- If the simulated honest committee member $\mathcal{M}_i$ does not abort, Sim uses the extractor for π^{ab} to extract the sharing-polynomial coefficients $\mathsf{coeffs}_{\mathsf{prev}^C}$ from the corrupted $\mathcal{C}$-members dealings $\mathcal{M}_i$ receives.
- Given $\mathsf{shares}_{\mathsf{prev}^H\to\mathsf{curr}^C}, \mathsf{coeffs}_{\mathsf{prev}^C}$, and $\mathsf{scmts}_{\mathsf{curr}^C}$ posted by $\mathcal{S}$, Sim interpolates $\mathsf{shares}_{\mathsf{curr}^C}$ which are consistent with $\mathsf{scmts}_{\mathsf{curr}^C}$. It returns $\mathsf{shares}_{\mathsf{curr}^C}$ if $\mathcal{M}_i$ did not abort, else it returns $\perp$.

$(\mathsf{shares}_{\mathsf{curr}^C}, \mathsf{shares}_{\mathsf{curr}^H\to\mathsf{next}^C})$
$\leftarrow \mathsf{Sim.Handover}(\mathsf{pk}, \mathsf{shares}_{\mathsf{prev}^H\to\mathsf{curr}^C}, \mathcal{C}_{\mathsf{prev}}, \mathcal{C}_{\mathsf{curr}}, \mathcal{C}_{\mathsf{next}})$:

Same as $\mathsf{ECPSS.Handover}(\mathcal{C}_{\mathsf{prev}}, \mathcal{C}_{\mathsf{curr}}, \mathcal{C}_{\mathsf{next}})$ except:

- Without loss of generality, we assume that $\mathcal{C}_{\mathsf{curr}}$ has $t-1$ corrupted members. If at least one simulated honest member $\mathcal{M}_i \in \mathcal{C}^H_{\mathsf{curr}}$ does not abort during ECPSS.Reconstruct, Sim gets $\mathsf{shares}_{\mathsf{curr}^C} \leftarrow \mathsf{Sim.Reconstruct}(\mathsf{shares}_{\mathsf{prev}^H \to \mathsf{curr}^C}, \mathcal{C}_{\mathsf{prev}})$. Else, the simulator sends $\langle \mathsf{Terminate} \rangle$ to $\mathcal{F}^{\mathsf{KM}}$ on behalf of the corrupted server.
- Based on $\mathsf{shares}_{\mathsf{curr}^C}$ and pk, Sim interpolates the expected share-commitments $\mathsf{scmts}_{\mathsf{curr}}$ w.r.t. generator g. For each honest simulated member $\mathcal{M}_i \in \mathcal{C}^H_{\mathsf{curr}}$ that has not aborted yet, Sim calls $\mathsf{shares}_{i \to \mathsf{next}^C} \leftarrow \mathsf{Sim.Share}(\mathsf{scmts}_{\mathsf{curr},i}, \mathcal{C}_{\mathsf{next}})$.
- Sim defines $\mathsf{shares}_{\mathsf{curr}^H \to \mathsf{next}^C} = \{\mathsf{shares}_{i \to \mathsf{next}^C}\}_{\mathcal{M}_i \in \mathcal{C}^H_{\mathsf{curr}}}$, and returns $\mathsf{shares}_{\mathsf{curr}^C}$ for higher-level protocol and $\mathsf{shares}_{\mathsf{curr}^H \to \mathsf{next}^C}$ for the next handover.

KM.KeyGen()

- If $\mathcal{S}$ is corrupted, Sim sends $\langle \mathsf{KeyGen} \rangle$ to $\mathcal{F}^{\mathsf{KM}}$ on behalf of the corrupted server. Irrespective of whether the server is corrupted, Sim receives $\langle \mathsf{KeyGen}, \mathsf{pk} \rangle$ from $\mathcal{F}^{\mathsf{KM}}$.
- **Epoch 0**: Sim follows the protocol, invoking ECPSS.Share from simulated honest committee members in $\mathcal{C}_0$, and stores $\mathsf{shares}_{0^H \to 1^C}$.
- **Epoch 1**: Epoch 1 proceeds the same way as the protocol, except honest parties follow $(\mathsf{shares}_{1^C}, \mathsf{shares}_{1^H \to 2^C}) \leftarrow \mathsf{Sim.Handover}(\mathsf{pk}, \mathsf{shares}_{0^H \to 1^C}, \mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2)$, where each simulated client uses generator g' when invoking Sim.Reconstruct and g when invoking Sim.Share. Each simulated client $\mathcal{M}_i \in \mathcal{C}^H_1$ that does not abort also replaces π^{dleq} with a simulated proof for a fake statement, which shows that $\mathsf{dlog}_g(\mathsf{ccmt}_{1,i,0}) = \mathsf{dlog}_{g'}(\mathsf{scmt}_{1,i})$ even though the former depends on pk while the latter is independent of it.
- If $\mathcal{S}$ is honest, when the simulated $\mathcal{S}$ interpolates pk, Sim responds to $\mathcal{F}^{\mathsf{KM}}$ on behalf of $\mathcal{A}$ to forward pk to honest server. Else, Sim sends this response when the corrupted $\mathcal{S}$ posts pk to the transparency log at the end of epoch 1.

KM.PublicKey()

- When $\mathcal{F}^{\mathsf{KM}}$ receives a $\langle \mathsf{PublicKey} \rangle$ request from an honest party $\mathcal{P}$, $\mathcal{F}^{\mathsf{KM}}$ informs Sim on behalf of $\mathcal{A}$. Sim then tries to run the get public key protocol while simulating $\mathcal{P}$. If it is able to retrieve pk, Sim sends $\top$ to $\mathcal{F}^{\mathsf{KM}}$ on behalf of $\mathcal{A}$; else it sends $\bot$. (only happens if $\mathcal{S}$ is corrupted).

KM.Recover($\mathsf{rqst}, \mathsf{st}_{\mathsf{new}}, \pi^{\mathsf{st}}$)

The protocol proceeds the same way except:

- If $\mathcal{S}$ is honest, Sim receives $\langle \mathsf{Recover}, \mathsf{rqst}, \mathsf{st}_{\mathsf{new}}, \pi^{\mathsf{st}}, \mathsf{resp} \rangle$ from $\mathcal{F}^{\mathsf{KM}}$ on behalf of $\mathcal{A}$. The simulated server initiates the processing of this request in the coming epoch curr.
- In epoch curr, honest parties follow $(\mathsf{shares}_{\mathsf{curr}^C}, \mathsf{shares}_{\mathsf{curr}^H \to \mathsf{next}^C}) \leftarrow \mathsf{Sim.Handover}(\mathsf{pk}, \mathsf{shares}_{\mathsf{prev}^H \to \mathsf{curr}^C}, \mathcal{C}_{\mathsf{prev}}, \mathcal{C}_{\mathsf{curr}}, \mathcal{C}_{\mathsf{next}})$ instead of $\mathsf{ECPSS.Handover}(\mathcal{C}_{\mathsf{prev}}, \mathcal{C}_{\mathsf{curr}}, \mathcal{C}_{\mathsf{next}})$.
- If $\mathcal{S}$ is corrupted and a simulated committee member $\mathcal{M}_i \in \mathcal{C}^H_{\mathsf{curr}}$ receives $(\mathsf{rqst}, \mathsf{st}_{\mathsf{prev}}, \mathsf{st}_{\mathsf{curr}}, \pi^{\mathsf{st}})$ and does not abort for epoch curr (Sim sends $\langle \mathsf{Terminate} \rangle$ if all abort), Sim sends $\langle \mathsf{Recover}, \mathsf{rqst}, \mathsf{st}_{\mathsf{new}}, \pi^{\mathsf{st}} \rangle$ on behalf of corrupted $\mathcal{S}$ to $\mathcal{F}^{\mathsf{KM}}$. In response, it receives $\langle \mathsf{Recover}, \mathsf{rqst}, \mathsf{st}_{\mathsf{new}}, \pi^{\mathsf{st}}, \mathsf{resp} \rangle$ from $\mathcal{F}^{\mathsf{KM}}$. Note that irrespective of server corruption, Sim now has resp corresponding to rqst.
- From Sim.Handover, Sim knows $\mathsf{shares}_{\mathsf{curr}^C}$. Given resp and these shares, it can interpolate the expected response resp_i for each simulated $\mathcal{M}_i \in \mathcal{C}^H_{\mathsf{curr}}$, and this is what the simulated $\mathcal{M}_i$ sends back to the server.
- If $\mathcal{S}$ is honest, when the simulated $\mathcal{S}$ interpolates resp, Sim informs $\mathcal{F}^{\mathsf{KM}}$ on behalf of $\mathcal{A}$ to forward resp to honest server. Else, Sim informs $\mathcal{F}^{\mathsf{KM}}$ to move forward when the corrupted $\mathcal{S}$ posts a new guess-limiting state to the transparency log.

Appendix B. Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

B.1. Summary

The paper presents a client-aided key management system for secure backup and recovery of user secrets that avoids entrusting a centralized server with decryption capability. This approach distributes trust among all clients of a client-server system where a group of clients are responsible for recovering a secret for one of these clients that have lost all of their devices. The protocol protects every client's secret against a mobile adversary that is capable of corrupting a fraction of them at a time. To tolerate a mobile adversary, the protocol changes the committees at intermittent intervals.

B.2. Scientific Contributions

- Creates a New Tool to Enable Future Science.
- Provides a Valuable Step Forward in an Established Field.

B.3. Reasons for Acceptance

1) The ability to use asymmetry neatly, such that the server is only relied upon for availability and security, is derived from a large number of clients without requiring the clients to be online.
2) Present an evaluation that shows that clients have little effort in backing up, and also not-so-high costs for recovery (in terms of compute time, communication, and battery usage).
3) Password-protect backups that reduce trust in the server are an important topic.
4) Addresses an important problem: the idea of distributing key management across rotating client committees is well motivated.

B.4. Noteworthy Concerns

1) The technical novelty appears limited, and as a result the potential impact of the solution is unclear. In other words, would such system ever be deployed? The burden is highly dependent on adoption.

Appendix C. Response to the Meta-Review

Novelty. Chorus takes a new approach to secret recovery and demonstrates its feasibility for 100M users on mobile devices. In addition, it proposes new, albeit conceptually simple, techniques that all help bring this approach from theory to practice. Specifically, our new secret-recovery scheme restricts committee involvement to relatively rare recovery operations, our new server-aided NIVSS scheme reduces server and client costs by two orders of magnitude, and our new seed-selection scheme reduces assumptions on the PKI made by prior work.

Broader impact. Our techniques can potentially benefit applications beyond secret recovery (see §8). In particular, several works on federated learning and federated analytics distribute trust across client committees, and both of our ECPSS techniques can benefit those settings: our committee-selection mechanism reduces PKI trust assumptions compared to Lotto, and our server-aided NIVSS scheme removes the need for intra-committee interaction during state transfer. In the secret-recovery setting itself, Chorus targets large-scale deployments such as Signal; with 100M users, a client participates roughly once every four months for under 30 seconds.