Verifiable Aggregate Receipts with Applications to User Engagement Auditing

Ioannis Kaklamanis*
Yale University, IC3
giannis.kaklamanis@yale.edu

Wenhao Wang*
Yale University, IC3
wenhao.wang@yale.edu

Harjasleen Malvai
UIUC, IC3
hmalvai2@illinois.edu

Sen Yang
Yale University, IC3
sen.yang@yale.edu

Fan Zhang
Yale University, IC3
f.zhang@yale.edu

***Abstract*—Accurate measurements of user engagement underpin important decisions in various settings, such as determining advertising fees based on viewership of online content, allocating public funding based on a clinic's reported patient volume, or determining whether a group chat app disseminated a message without censorship. While common, self-reporting is inherently untrustworthy due to misaligned incentives (e.g., to inflate).**

Motivated by this problem, we introduce the notion of Verifiable Aggregate Receipts (VAR). A VAR system allows an issuer to issue receipts to users and to verify the number of receipts possessed by a prover, who is given receipts upon serving users. An ideal VAR system should satisfy inflation soundness (the prover cannot overstate the count), privacy (the verifier learns only the count), and be performant for large-scale applications with millions of users.

We formalize VAR using an ideal functionality and present two novel constructions. Our first protocol, S-VAR, leverages bottom-up secret-sharing to enable tiered "fuzzy" audits, and achieves constant-size receipts regardless of the number of supported thresholds. Our second protocol, P-VAR, uses bilinear pairings to aggregate receipts into a proof verifiable in constant time, enables exact auditing, and can be extended to handle a dynamic user set. We prove both constructions secure with respect to our ideal functionality.

We implement and benchmark our VAR constructions, showing that they outperform all baseline and existing solutions by at least an order of magnitude in both proving and verification time. Finally, we show how VAR applies to the aforementioned use cases and further demonstrate its practicality by integrating it with Bluesky, a decentralized social network.

1. Introduction

Delegation is a common practice in modern economies: individuals and institutions rely on specialized third parties to perform services on their behalf. This spans from content creators relying on social media platforms to reach followers, to corporations outsourcing employee-benefits administration to external providers. Across these settings, a fundamental security problem is to *verify service quality*, e.g., how many eligible users were reached or how many employees were actually served. In practice, however, such verifiability is often based on trust or is completely missing.

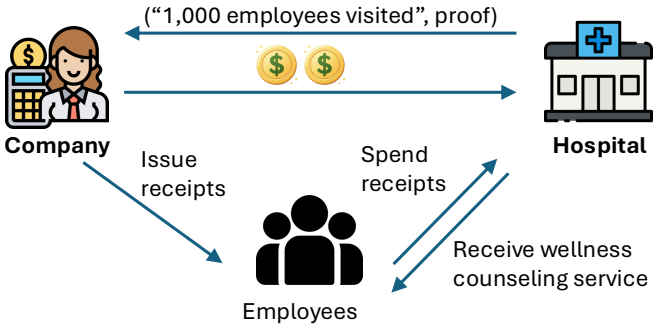


Figure 1: The workflow of user engagement auditing using VAR in the application of company-sponsored benefits.

Consider the concrete example of company-sponsored benefits, where a company C partners with an external provider P to offer subsidized benefits (such as wellness counseling) to its employees. In practice, it's common for P to self-report the number of visits to C, based on which it gets reimbursed. However, self-reporting is prone to fraud because P has an incentive to *inflate* the report, which is often only caught afterwards, through costly auditing and lawsuits. Multiple DOJ enforcement actions [1], [2], [3] document healthcare providers inflating visit counts or billing for services not rendered – fraud of tens of millions of dollars.

Similar verification is desired in advertising (e.g., verifying impression claims), the creator economy (e.g., auditing billing of paid boosts such as Boost Posts on Instagram, Promote on TikTok, Promoted Posts in X), group chats (e.g., verifying censorship-free delivery of broadcast messages), and any delegation setting where a principal needs to verify an intermediary's claims about the number of services rendered to eligible users. In this paper, we formalize this problem as *user-engagement auditing*, and study the

*. These authors contributed equally to this work.

cryptographic solutions for it.

Verifiable Aggregate Receipts. Motivated by this problem, we introduce the notion of *Verifiable Aggregate Receipts* (VAR). We assume each party has a well-known public key. A VAR protocol involves a principal who delegates services, a set U of users who are meant to receive services, and a third-party service provider $\mathcal{P}$ that renders services. In our model, the principal plays the role of both the issuer $\mathcal{I}$ and the verifier $\mathcal{V}$. It first issues receipts to users, who access certain services by "spending" a receipt at $\mathcal{P}$. At a later time, the principal can request and verify a succinct cryptographic proof of the number of receipts obtained by $\mathcal{P}$.

Modeling after the above user-engagement auditing tasks, VAR aims to achieve *inflation soundness*, meaning $\mathcal{P}$ cannot overstate the count. Note that $\mathcal{P}$ is naturally disincentivized to deflate – it only gets reimbursed less if it underreports. In addition, the above examples illustrate the need for *user privacy*, e.g., employees don't wish to reveal whether/when they use certain benefits to their employer as this may disclose sensitive medical information. Thus, VAR aims to guarantee that $\mathcal{V}$ learns only the count, not individual identities. Finally, a VAR system should scale to support large-scale applications involving millions of users. In particular, we require that the audit cost of the principals and users does not scale with the number of users. Since $\mathcal{P}$ serves users, it incurs a cost at least linear in the number of users.

Strawman solutions that do not work. To illustrate the challenge, we explain why closely related primitives are not sufficient for building VAR. Most closely related are aggregate signatures, such as BLS [4]. A strawman idea is to ask users to sign when receiving the service, and use the aggregate signature (of constant size) as audit evidence. However, this simple approach violates privacy and fails to yield a succinct proof of count. In particular, verifying an aggregate signature requires $\mathcal{V}$ to know the set of public keys included in the aggregation, which both compromises privacy and makes verification scale with the number of users. In principle, these issues can be addressed using zero-knowledge techniques, but state-of-the-art constructions [5], [6] as well as baseline approaches (e.g., generic zk-SNARKs) do not scale to the number of users that VAR aims to support. Moreover, requiring users to sign can be undesirable, as user signatures are irrefutable proof that they received a specific service, which our solutions avoid. In §A, we explain why other related primitives, such as ring signatures and Aggregate MACs, do not meet performance or security requirements of VAR.

1.1. Our Contributions

Two Constructions. Our first construction, S-VAR, is a secret-sharing-based VAR protocol which supports *tiered* audits: $\mathcal{V}$ fixes certain tiers, or *thresholds*, $t_1 < \cdots < t_k \leq n$, where n is the number of users, and $\mathcal{P}$ proves the highest threshold that has been reached. The main advantage of this construction is compact user state: using a tool called bottom-up secret sharing [7], each user stores a single seed, independent of the number of thresholds. The resulting protocol is well suited to settings where approximate, threshold-based auditing is sufficient and where avoiding pairings is desirable. Our second construction, P-VAR, lets $\mathcal{P}$ prove the *exact* number of users who have spent while hiding the spending set, and naturally extends to repeated audits. P-VAR uses pairing-based arguments and thus features concretely fast audit proving time and succinct verification. We also extend P-VAR to dynamic user populations, so that costs scale with the active set of users rather than with a maximum population fixed at setup.

Implementation and Evaluation. We implement both S-VAR and P-VAR in Rust. We benchmark on two AWS configurations ("small" and "large"), varying the number of users $n \in \{2^{10}, \ldots, 2^{20}\}$, and measure issuance bandwidth and time, as well as audit proving and verification time. For a million users, issuance takes less than $2\,\mathrm{s}$ for either scheme, and for audit proving time, P-VAR requires $12\,\mathrm{s}$, and S-VAR requires $21\,\mathrm{s}$. The key takeaway is that both our protocols are concretely practical at a million-user scale and significantly outperform heavyweight baseline and existing solutions.

Applications of VAR. To showcase the power of VAR, we discuss four socially important applications: auditing social media platforms, preventing fraudulent reimbursement claims, detecting censorship in broadcast, and verifying contributions in peer-to-peer file sharing. We briefly describe the latter three in §6; here, we focus on the social media setting and our concrete integration of VAR with Bluesky.

Content creators often lack a way to verify whether social media platforms actually deliver promised visibility, especially for paid boosts and other promotion mechanisms. VAR provides a cryptographic audit mechanism: users who receive or view promoted content spend anonymous receipts, and the platform proves the resulting count without revealing which users saw the post.

As a concrete case study, we implement **TrueReach**[1], an integration of VAR with Bluesky, a decentralized social network built on the AT protocol [8]. In TrueReach, the feed generator that recommends promoted content acts as $\mathcal{P}$, while the creator acts as $\mathcal{V}$: the creator issues receipts to target users, and at audit time the feed generator proves the number of distinct user views. Implemented with a Rust-to-WASM cryptographic backend, TrueReach supports 1M users on a realistic creator-side device, requiring $53.3\,\mathrm{s}$ for receipt issuance, $16.1\,\mathrm{s}$ for encryption, and only $39\,\mathrm{ms}$ for audit verification, showing it is practical at large scale.

Summary of Contributions. In summary, this paper makes the following contributions:

- We introduce VAR, a new cryptographic primitive with applications to user engagement auditing. We model the security properties of VAR via an ideal functionality (§3).
- We propose two VAR constructions and prove them secure: S-VAR (§4) is a secret-sharing-based protocol which supports tiered auditing, and P-VAR (§5) is a pairing-

1. An anonymous instance is available at https://bsky.truereach.cc/.

based protocol which supports exact auditing. We also extend P-VAR to support a dynamic user set (§5.2).
- We implement and benchmark our protocols, showing they are concretely practical at large scale and outperform existing solutions by at least an order of magnitude (§7).
- We showcase four diverse applications of VAR (§6), as well as TrueReach (§7.5), an end-to-end integration of VAR with the Bluesky social network.

2. Preliminaries

In this section, we go over notation and preliminaries.

Notation and common primitives. We use λ for the security parameter. For any $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. We use $|S|$ to denote the cardinality of the set S. We use "PPT" to denote "probabilistic polynomial time". We use boldface to denote vectors, i.e., $\boldsymbol{x} = (x_1, \ldots, x_n)$. A family of polynomial-time functions is a pseudorandom function (PRF) if a uniformly chosen function from the family is indistinguishable from a truly random function to any PPT adversary. We denote evaluation by $\mathsf{PRF}(\mathsf{seed}; x)$. We instantiate PRF as $\mathsf{SHA256}(\mathsf{seed}\|x)$ [9, Section V.A]. We model hash functions in the random oracle model (ROM) [10]. We write $\mathsf{H}(x)$ to denote hash function H evaluated on input x. The domain and range of H will be clear from context. A digital signature scheme is a tuple of PPT algorithms $\mathsf{DS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$. We rely on the standard correctness and unforgeability definitions [11].

Pairing-friendly groups. Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be the pairing-friendly groups with the pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Let $g_1 \in \mathbb{G}_1$, $g_2, h_2 \in \mathbb{G}_2$ be random and independent group generators in $\mathbb{G}_1$ and $\mathbb{G}_2$.

KZG polynomial commitment scheme. In the KZG polynomial scheme [12], the common reference string (CRS) is $(g_1, g_1^{\tau}, \ldots, g_1^{\tau^d})$ and $(g_2, g_2^{\tau}, \ldots, g_2^{\tau^d})$ for some random τ and maximum degree d, and $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$ are uniformly sampled generators in $\mathbb{G}_1$ and $\mathbb{G}_2$. The commitment of a polynomial $f(x) = \sum_{i=0}^{d} f_i x^i$ is $\mathsf{cm}_f := g_1^{f(\tau)}$, which is obtained by computing $\mathsf{cm}_f = \prod_{i=0}^{d} (g_1^{\tau^i})^{f_i}$. To show that a polynomial $Z(x)$ divides $f(x)$ with $q(x)$ as the quotient, the succinct proof is $\pi := g_1^{q(\tau)}$. The proof can be verified via the pairing check $e(\mathsf{cm}_f, g_2) \stackrel{?}{=} e(\pi, g_2^{Z(\tau)})$.

Secret sharing. An (n, t)-secret sharing scheme [13] allows a dealer to distribute a secret s among n parties such that s can be recovered with any t shares where $1 \leq t \leq n$, while $t - 1$ shares give no information on s. In a $(t_1, \ldots, t_k)$-*multi*-secret sharing scheme [14], [15], k independent secrets $s_1, \ldots, s_k$ are secret-shared; each secret s_i can be recovered with at least t_i shares, but not with fewer.

3. Problem Statement

3.1. Problem: Verifiable Aggregate Receipts

Broadly, we investigate protocols for building "verifiable aggregate receipts", i.e., protocols that enable issuance of receipts and their privacy-preserving aggregation into a compact proof of count. The goal is to allow an issuer $\mathcal{I}$ to issue receipts to a set of users U, so that a prover $\mathcal{P}$ can later prove to a verifier $\mathcal{V}$ that a certain number of users from U have spent their receipts, while keeping the identity of those users secret. We focus on the setting where the issuer $\mathcal{I}$ and the verifier $\mathcal{V}$ are the same entity. We also focus on guarding against inflation of the total count, i.e., we assume that a malicious $\mathcal{P}$'s only goal is to convince $\mathcal{V}$ of a count that is larger than the number of received receipts.

3.2. Models and Definitions

Here, we describe the system model, threat model, and the security and efficiency properties we require in VAR.

System model. A Verifiable Aggregate Receipts (VAR) system involves three kinds of parties: a Verifier $\mathcal{V}$, a Prover $\mathcal{P}$ and a set of n users $U = \{\mathcal{U}_i\}_{i=1}^{n}$; in VAR, $\mathcal{V}$ also acts as the issuer. We assume each party has a well-known public key (i.e., we assume a PKI), so communication between parties takes place in a secure channel. A VAR protocol involves three procedures:

- Issue: $\mathcal{V}$ interacts with all users in U to issue them receipts.
- Spend: Once issuance is complete, any $\mathcal{U} \in U$ can interact with $\mathcal{P}$ to spend their receipt.
- Audit: At any point after issuance, $\mathcal{V}$ may interact with $\mathcal{P}$ to ascertain how many $\mathcal{U} \in U$ have spent their receipts. At the end, $\mathcal{V}$ outputs either an integer $k \leq |U|$ or $\perp$.

We model the essential properties of VAR using an ideal functionality $\mathcal{F}_{\mathsf{VAR}}$, presented in Fig. 2. To separate different instances of $\mathcal{F}_{\mathsf{VAR}}$, all messages are tagged with a unique *session id* denoted sid.

In $\mathcal{F}_{\mathsf{VAR}}$, we allow $\mathcal{V}$ to specify a set T of allowed counts in order to enable two kinds of VAR protocols: (1) "exact-audit" protocols which enable $\mathcal{P}$ to prove the exact count of spent receipts, and (2) "fuzzy/tiered-audit" protocols which enable $\mathcal{P}$ to prove counts from a specific set of allowed counts, capturing threshold-based solutions. Note that in exact-audit protocols, $\mathcal{V}$ will set $T := [|U|]$.

At issuance, $\mathcal{V}$ specifies an allowed-count set $T \subseteq [|U|]$; $\mathcal{F}_{\mathsf{VAR}}$ records T and sends it to $\mathcal{P}$. After issuance, the spend and audit procedures may be invoked in any order and may interleave. If an honest user sends a spend message to $\mathcal{F}_{\mathsf{VAR}}$, the functionality adds $\mathcal{U}$ to the set R who have spent their receipts and notifies $\mathcal{P}$. When $\mathcal{V}$ makes an audit request, $\mathcal{F}_{\mathsf{VAR}}$ computes $\mathsf{spent} = |R|$ (the true number of spenders) and forwards the audit request to $\mathcal{P}$. $\mathcal{P}$ replies with a pair $(b, \mathsf{claimed})$, where $b \in \{0, 1\}$ signals whether the prover wishes to make a valid or bad audit claim. If $b = 1$, $\mathsf{claimed} \in T$, and $\mathsf{claimed} \leq \mathsf{spent}$, $\mathcal{F}_{\mathsf{VAR}}$ sends $\mathsf{claimed}$ to $\mathcal{V}$; otherwise, it notifies $\mathcal{V}$ of a bad audit claim. During the audit procedure, $\mathcal{F}_{\mathsf{VAR}}$ allows a *corrupted* prover to perform "extra" spending on behalf of corrupted users who have not yet spent, before $\mathcal{P}$ sends his claimed count.

Ideal Functionality $\mathcal{F}_{\mathsf{VAR}}$

Parties. Verifier $\mathcal{V}$, prover $\mathcal{P}$, user set $U = \{\mathcal{U}_i\}_{i\in[n]}$, adversary $\mathcal{A}$.

State.

- sid: the unique identifier of the current session.
- T: list of allowed counts to prove at audit (increasing order).
- $e_{\max}$: maximum number of times a successful audit may be run with unique spend values.
- $U_{\mathcal{C}}$: set of corrupted users.
- e: the current epoch number.
- R: set of users who have spent receipts.
- $U'_{\mathcal{C}}$: set of corrupted users who have spent receipts.

Initialize (One-time)
On message $(\mathsf{sid}, \textsc{Init}, U_{\mathcal{C}})$ from $\mathcal{A}$, store $(\mathsf{sid}, U_{\mathcal{C}})$, and set $R := \emptyset$, $U'_{\mathcal{C}} := \emptyset$, and $e = 0$.

Issue (One-time)
On message $(\mathsf{sid}, \textsc{Issue}, T, e_{\max})$ from $\mathcal{V}$, if there exists record $(\mathsf{sid}, U_{\mathcal{C}})$, store $(\mathsf{sid}, \mathcal{V}, T, e_{\max})$. Send message $(\mathsf{sid}, \textsc{Issue}, T, e_{\max})$ to $\mathcal{A}$, $\mathcal{P}$, and to each $\mathcal{U}_i \in U$.

After initialize and issue, parties can call the following two procedures any number of times and in any order, and the ideal functionality defines the output as follows.

Spend
On message $(\mathsf{sid}, \textsc{Spend})$ from $\mathcal{U}$, if there exists record $(\mathsf{sid}, \mathcal{V}, T, e_{\max})$:

- If $\mathcal{U} \notin R$, store and send $(\mathsf{sid}, \textsc{Spend}, \mathcal{U})$ to $\mathcal{P}$.
- Set $R = R \cup \{\mathcal{U}\}$. If $\mathcal{U} \in U_{\mathcal{C}}$, set $U'_{\mathcal{C}} = U'_{\mathcal{C}} \cup \{\mathcal{U}\}$.

Audit

- On message $(\mathsf{sid}, \textsc{Audit})$ from $\mathcal{V}$, if there exists record $(\mathsf{sid}, \mathcal{V}, T, e_{\max})$: set $\mathsf{spent} = |R|$, store $(\mathsf{sid}, \textsc{Audit}, \mathcal{V}, \mathsf{spent})$, and send message $(\mathsf{sid}, \textsc{AuditReq}, \mathcal{V})$ to $\mathcal{P}$.
- If $e \geq e_{\max}$, send $(\mathsf{sid}, \textsc{AuditBad})$ to $\mathcal{V}$.
- If $\mathcal{P}$ is corrupted: on message $(\mathsf{sid}, \textsc{SpendExtra}, U''_{\mathcal{C}})$ from $\mathcal{A}$, if there exists record $(\mathsf{sid}, \textsc{Audit}, \mathcal{V}, \mathsf{spent})$, check that $U''_{\mathcal{C}} \cup U'_{\mathcal{C}} \subseteq U_{\mathcal{C}}$ and $U'_{\mathcal{C}} \cap U''_{\mathcal{C}} = \emptyset$. If so, add each $\mathcal{U} \in U''_{\mathcal{C}}$ to R, and change the record $(\mathsf{sid}, \textsc{Audit}, \mathcal{V}, \mathsf{spent})$ to $(\mathsf{sid}, \textsc{Audit}, \mathcal{V}, \mathsf{spent} + |U''_{\mathcal{C}}|)$. Also, set $U'_{\mathcal{C}} = U'_{\mathcal{C}} \cup U''_{\mathcal{C}}$.
- On message $(\mathsf{sid}, \textsc{AuditResp}, b, \mathsf{claimed})$ from $\mathcal{P}$, and if there exists $(\mathsf{sid}, \textsc{Audit}, \mathcal{V}, \mathsf{spent})$, then:
  - If $b = 1$, $\mathsf{claimed} \in T$, and $\mathsf{claimed} \leq \mathsf{spent}$, send $(\mathsf{sid}, \textsc{AuditOK}, \mathsf{claimed})$ to $\mathcal{V}$.
  - Otherwise, send $(\mathsf{sid}, \textsc{AuditBad})$ to $\mathcal{V}$.

  Set $e := e + 1$ and purge $(\mathsf{sid}, \textsc{Audit}, \mathcal{V}, \mathsf{spent})$.

Figure 2: Ideal Functionality for VAR.

This extra spending aims to emulate the real-world ability of the adversary to use additional receipts (from corrupted users) at audit time to inflate the claimed count.

To model corrupted users, at initialization, the adversary $\mathcal{A}$ fixes and sends a corrupted user set $U_{\mathcal{C}} \subseteq U$ to $\mathcal{F}_{\mathsf{VAR}}$. $\mathcal{F}_{\mathsf{VAR}}$ models the correctness, security (inflation soundness), and privacy properties discussed in §3.2.

Adverserial model. We consider a static, computationally bounded adversary $\mathcal{A}$ that corrupts parties. Concretely, $\mathcal{P}$ and any subset of users $U_{\mathcal{C}} \subseteq U$ can be *malicious*, i.e., they may deviate arbitrarily from the protocol. We assume that $\mathcal{V}$ is *semi-honest*, i.e., he follows the protocol but may try to infer additional information from its view. We discuss the semi-honest assumption in §9.

Below we discuss the correctness, security, and privacy guarantees we require from a VAR system.

- *Correctness.* Given honest $\mathcal{V}, \mathcal{P}$, and U, a VAR protocol should ensure that at the end of the audit procedure, $\mathcal{V}$ is convinced of the count claimed by $\mathcal{P}$.
- *Inflation soundness.* Informally, if $\mathcal{A}$ has corrupted $\mathcal{P}$ and $|U_{\mathcal{C}}|$ users, and if ℓ honest parties spend their receipts, then $\mathcal{V}$ should only accept an audit count of at most $\ell + |U_{\mathcal{C}}|$. Since $\mathcal{P}$ is the only party that interacts with users during spending, it may censor some of the spent receipts; recall, though, we deem deflation of receipt count out-of-scope.
- *Privacy guarantee.* Given an honest $\mathcal{P}$, a semi-honest $\mathcal{V}$, let $U' \subseteq U$ be the set of honest users who engaged in the spending procedure upon an audit request from $\mathcal{V}$ to $\mathcal{P}$. If $\mathcal{V}$ accepts the claimed audit count k, he should not learn any other information besides the count k. Informally, from the point of view of $\mathcal{V}$, the set U' is as likely to be the set of honest users who spent as any other $|U'|$-sized subset of honest users.

As we discuss in §9, it is also desirable that spending does not produce irrefutable evidence linking a user to a specific service. We do not formalize this property in our ideal functionality $\mathcal{F}_{\mathsf{VAR}}$, but both of our constructions, by not requiring signatures, take a step in the direction of *deniability*.

Efficiency. Broadly, we are interested in VAR protocols that are efficient in terms of *computation time* and *bandwidth* across all parties and procedures. When we say that a party's running time or bandwidth is "constant", we mean it is independent of the number of users, the number of users who spent, as well as the set of allowed counts; it can still be polynomial in the security parameter λ.

We focus on VAR protocols where users run in constant time and use constant bandwidth during both issuance and spending. Further, a VAR protocol should guarantee that $\mathcal{V}$'s issuance time is linear in the number of users, and that $\mathcal{V}$'s audit time and bandwidth is sublinear, ideally constant. Note that in most of the VAR applications in §6, $\mathcal{P}$ is usually an entity with significantly more compute and bandwidth resources compared to individual users and $\mathcal{V}$. As such, our focus is on minimizing the computation and bandwidth usage of $\mathcal{V}$ and of each $\mathcal{U} \in U$; at the same time, we wish to preserve $\mathcal{P}$'s time and bandwidth at reasonable levels towards practical deployment at scale.

4. S-VAR: VAR Based on Secret Sharing

In this section, we propose the use of multi-secret sharing to construct VAR with *tiered* auditing. By "tiered", we mean that the prover $\mathcal{P}$ can only produce audit proofs for some predetermined count values, or tiers. In other words, this VAR construction supports "fuzzy" auditing, rounded down to the closest pre-determined threshold.

At a high-level, our construction uses reconstruction of a secret as a proof of count. $\mathcal{V}$ associates each allowed threshold t_j with a secret s_j, and users receive receipt material that includes "shares" of each s_j. When users spend, they reveal this material to $\mathcal{P}$; for audit, $\mathcal{P}$ reconstructs a secret s_j^* for the highest possible threshold t_j and sends it to $\mathcal{V}$ who checks if $s_j^* = s_j$. The main efficiency obstacle is that independent Shamir sharings would make each receipt grow linearly with the number of t_j. We avoid this with bottom-up multi-secret sharing [7], which compresses each user's receipt material to a single seed. Since the set of spending users is arbitrary, reconstruction cannot rely on FFT-friendly evaluation sets; instead, we use barycentric Lagrange interpolation with fast weight computation.

After introducing an abstraction for multi-secret sharing in §4.1, we present a generic VAR construction in §4.2. In §4.3, we present S-VAR, our optimized instantiation based on a bottom-up secret-sharing scheme.

4.1. Multi-Secret Sharing Abstraction

We consider two paradigms in *single*-secret sharing that are relevant to our construction. In the traditional *top-down* paradigm [13] with n users and a threshold $t < n$, the dealer usually samples a t-degree polynomial and distributes its evaluations as user shares. In the more recent *bottom-up* paradigm [7], instead, user shares are in some sense *pre-existing*, i.e., they exist before the sharing occurs; the dealer now interpolates an n-degree polynomial over the pre-existing shares (and the secret) and publishes some *auxiliary data* to support t-threshold reconstruction. The choice of paradigm for *multi*-secret sharing yields a clear tradeoff. *Bottom-up* can enable a single, fixed per-user share that works across all thresholds, at the cost of publishing auxiliary data to support reconstruction. *Top-down*, by contrast, requires a different per-user share for each threshold but needs no auxiliary data to reconstruct.

Generic multi-secret sharing (GMSS). To capture both paradigms, we model multi-secret sharing as an abstract primitive which we call Generic Multi-Secret Sharing (GMSS). GMSS allows the Share algorithm to output user shares *and* auxiliary data, and the Recon algorithm to take as input a reconstruction set *and* the auxiliary data. Intuitively, GMSS captures both paradigms by abstracting two design knobs: the size of the per-user share and the use (or absence) of the auxiliary data. Formally, an (n, k)-GMSS scheme for n users and k thresholds is a tuple of algorithms $\mathsf{GMSS} = (\mathsf{Share}, \mathsf{Recon})$ with the following syntax:

- $\mathsf{Share}(\boldsymbol{t}, \boldsymbol{s}) \to (\{\mathsf{sh}_i\}_{i\in[n]}, \mathsf{aux})$: Takes as input a tuple of thresholds $\boldsymbol{t} = (t_1, \ldots, t_k)$ and secrets $\boldsymbol{s} = (s_1, \ldots, s_k)$, and outputs shares $\{\mathsf{sh}_i\}_{i\in[n]}$ and auxiliary data aux.
- $\mathsf{Recon}(\boldsymbol{t}, I, \{\mathsf{sh}_i\}_{i\in I}, \mathsf{aux}) \to s$: Takes as input a tuple of thresholds $\boldsymbol{t}$, a reconstruction set $I \subseteq [n]$ such that $|I| = t \in \boldsymbol{t}$, a set of shares $\{\mathsf{sh}_i\}_{i\in I}$, and the auxiliary data aux and outputs a secret s or $\perp$.

An (n, k)-GMSS scheme should satisfy correctness and security. Informally, *correctness* states that for any number $m \le n$, any m valid shares – together with the auxiliary data aux – are sufficient to reconstruct every secret whose threshold is at most m. *Security* states that for every $j \in [k]$, any collection of fewer than t_j shares together with aux reveals no information about secret s_j. Looking ahead, we prove security of S-VAR by relying on the security of single-secret sharing and hash functions in ROM. As such, we do not present a formal security definition of GMSS. For a formal treatment of closely related primitives, we refer the reader to the work of Blundo [14] on multi-secret sharing, as well as to the recent work of Kate et al. [7] that formalizes bottom-up secret sharing.

4.2. VAR Based on GMSS

We now describe our generic VAR construction based on GMSS; the formal protocol is presented in Fig. 3.

Setup. Given the security parameter λ and the number of users n, we fix a scalar field $\mathbb{F}$ with order $|\mathbb{F}| \ge 2^\lambda$. The protocol uses a multi-secret sharing scheme $\mathsf{GMSS} = (\mathsf{Share}, \mathsf{Recon})$ and a digital signature scheme $\mathsf{DS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$. At setup, $\mathcal{V}$ samples $(sk, vk) \leftarrow \mathsf{DS.Gen}(1^\lambda)$ and sends vk to $\mathcal{P}$.

Issuance. At issuance, $\mathcal{V}$ selects a number of tiers k, chooses increasing thresholds $\boldsymbol{t} = (t_j)_{j\in[k]}$, and samples one secret $s_j \leftarrow \mathbb{F}$ per tier j. He forms the tuple $\boldsymbol{s} = (s_j)_{j\in[k]}$ and stores $(\boldsymbol{t}, \boldsymbol{s})$. Then $\mathcal{V}$ computes $(\{\mathsf{sh}_i\}_{i\in[n]}, \mathsf{aux}) \leftarrow \mathsf{GMSS.Share}(\boldsymbol{t}, \boldsymbol{s})$, and sends the thresholds $\boldsymbol{t}$ and the auxiliary data aux to $\mathcal{P}$. For each $i \in [n]$, $\mathcal{V}$ signs $\sigma_i \leftarrow \mathsf{DS.Sign}(sk, i\|\mathsf{sh}_i)$, forms receipt $\mathsf{rx}_i = (\mathsf{sh}_i, \sigma_i)$, and sends rx_i to $\mathcal{U}_i$. On receiving $(\boldsymbol{t}, \mathsf{aux})$, $\mathcal{P}$ initializes an empty set R of received shares.

Spending. To spend her receipt, user $\mathcal{U}_i$ sends rx_i to $\mathcal{P}$. On receiving $\mathsf{rx}_i = (\mathsf{sh}_i, \sigma_i)$, $\mathcal{P}$ checks $\mathsf{DS.Verify}(vk, i\|\mathsf{sh}_i, \sigma_i) = 1$ and, if valid, adds sh_i into the set R of received shares.

Audit. When $\mathcal{V}$ requests an audit, $\mathcal{P}$ identifies the largest threshold $t_{j^*} \in \boldsymbol{t}$ such that $t_{j^*} \le |\mathrm{R}|$. He chooses a t_{j^*} - sized reconstruction set $I \subseteq [n]$ from R, reconstructs $s' = \mathsf{GMSS.Recon}(\boldsymbol{t}, I, \{\mathsf{sh}_i\}_{i\in I}, \mathsf{aux})$, and sends (j^*, t_{j^*}, s') to $\mathcal{V}$. Then $\mathcal{V}$ checks if t_{j^*} is one of the thresholds in $\boldsymbol{t}$ and whether $s' = s_{j^*}$. If so, $\mathcal{V}$ accepts; else, he rejects.

Efficiency implications of GMSS for VAR. We explain how GMSS costs cleanly factor into VAR. Each user receives, holds, and spends exactly one receipt – their GMSS share – so the user's bandwidth and storage is $|\mathsf{sh}_i|$. As the number of thresholds grows, $|\mathsf{sh}_i|$ can grow as well depending on how GMSS is instantiated. The issuance bandwidth for $\mathcal{V}$ is $|\mathsf{aux}| + \sum_{i=1}^{n} |\mathsf{sh}_i| = |\mathsf{aux}| + n|\mathsf{sh}_i|$, since he sends one share to each of the n users, as well as the auxiliary data aux to $\mathcal{P}$. On $\mathcal{P}$'s side, the storage is $|\mathsf{aux}|$ up front plus the number of sh_i that arrive during spending, for a worst-case total of $|\mathsf{aux}| + n|\mathsf{sh}_i|$. Based on the protocol description in Fig. 3 $\mathcal{V}$'s storage is $O(k)$, since he has to store the k (threshold, secret) pairs. To reduce $\mathcal{V}$'s storage in to $O(1)$, instead of storing all k secrets, $\mathcal{V}$ can instead store a single secret

Public Parameters

- Security parameter λ and number of users $n = |U|$.
- Scalar field $\mathbb{F}$ of order $|\mathbb{F}| \geq 2^\lambda$.
- Multi-Secret Sharing scheme $\mathsf{GMSS} = (\mathsf{Share}, \mathsf{Recon})$.
- Signature scheme $\mathsf{DS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$.

Setup

1) $\mathcal{V}$ samples $(sk, vk) \leftarrow \mathsf{DS.Gen}(1^\lambda)$ and sends vk to $\mathcal{P}$, and sets $e_{\max} := \infty$.

Issue

1) To issue receipts, $\mathcal{V}$ proceeds as follows.
   - Chooses a number $k \in [n]$ and thresholds $\{t_j\}_{j\in[k]}$ such that $t_j < t_{j+1}$ for all j. Let $\boldsymbol{t} := (t_j)_{j\in[k]}$.
   - Samples secrets $s_j \leftarrow \mathbb{F}$ for $j \in [k]$. Let $\boldsymbol{s} = (s_j)_{j\in[k]}$.
   - Computes $(\{\mathsf{sh}_i\}_{i\in[n]}, \mathsf{aux}) \leftarrow \mathsf{GMSS.Share}(\boldsymbol{t}, \boldsymbol{s})$.
   - Signs $\sigma_i \leftarrow \mathsf{DS.Sign}(sk, i \| \mathsf{sh}_i)$ for each $i \in [n]$.
   - Sends $\mathsf{rx}_i := (\mathsf{sh}_i, \sigma_i)$ to user $\mathcal{U}_i$ for $i \in [n]$.
   - Sends $(\boldsymbol{t}, \mathsf{aux})$ to $\mathcal{P}$.
   - Stores $\{\boldsymbol{t}, \boldsymbol{s}\}$.
2) On receiving $(\boldsymbol{t}, \mathsf{aux})$ from $\mathcal{V}$, $\mathcal{P}$ initializes the set $\mathrm{R} := \emptyset$ of received shares.

Spend

1) User $\mathcal{U}_i$ sends rx_i to $\mathcal{P}$.
2) On receiving $\mathsf{rx}_i = (z_i, \sigma_i)$ from user $\mathcal{U}_i$, $\mathcal{P}$ first checks that $(i, z') \notin \mathrm{R}$ for any z' and $(i', z_i) \notin \mathrm{R}$ for any i'. Then, if $\mathsf{DS.Verify}(vk, i \| z_i, \sigma_i) = 1$, he updates $\mathrm{R} = \mathrm{R} \cup \{(i, z_i)\}$.

Audit

1) $\mathcal{V}$ sends an audit request to $\mathcal{P}$.
2) On receiving an audit request, $\mathcal{P}$ proceeds as follows:
   - Sets $j^* = \max\{j \in [k] \mid t_j \leq |\mathrm{R}|\}$ and chooses a set $I \subseteq [n]$ such that $\{(i, z_i)\}_{i\in I} \subseteq \mathrm{R}$ and $|I| = t_{j^*}$.
   - Fetches $\{\mathsf{sh}_i\}_{i\in I}$ from $\{z_i\}_{i\in I}$.
   - Computes $s' = \mathsf{GMSS.Recon}(\boldsymbol{t}, \{\mathsf{sh}_i\}_{i\in I}, \mathsf{aux})$.
   - Sends (j^*, t_{j^*}, s') to $\mathcal{V}$.
3) On receiving (j^*, t_{j^*}, s'), $\mathcal{V}$ outputs $(\mathsf{accept}, t_{j^*})$ if $t_{j^*} \in \boldsymbol{t}$ and $s' = s_{j^*}$; otherwise, he outputs reject.

Figure 3: VAR based on GMSS

seed and use it to derive all k secrets. This optimization is orthogonal to the GMSS instantiation, and thus $\mathcal{V}$'s storage overhead does not inform this choice.

A top-down instantiation of GMSS. GMSS can be instantiated by running k independent top-down (e.g., Shamir [13]) single-secret-sharing instances, one per threshold. Under this instantiation, each GMSS share sh_i consists of k top-down shares (one per tier), so $|\mathsf{sh}_i| = \Theta(k)$, and no auxiliary data is required ($|\mathsf{aux}| = 0$). This yields simple mechanics but an $O(k)$-sized receipt per user.

4.3. S-VAR: VAR Based on Bottom-Up GMSS

We observe that the *Bottom-Up Secret Sharing (BUSS)* paradigm of Kate et al. [7], which inverts the usual top-down secret sharing process, can help toward the constant-sized-receipt goal. In bottom-up secret sharing, rather than sampling a fresh t-degree polynomial and distributing its evaluations as user shares, the dealer uses an n-degree polynomial whose evaluations are deterministically derived from *pre-existing* user seeds and the secret. To achieve an effective t-out-of-n threshold, the dealer reveals $(n - t + 1)$ additional public points (disjoint from user points), reducing the reconstruction requirement from n to t.

S-VAR: Using a bottom-up instantiation of GMSS. We now present a bottom-up instantiation of GMSS, resulting in our final S-VAR construction. Each user's GMSS share is a single seed x_i (reused across all tiers), and the GMSS auxiliary data aux consists of all public points across all tiers. To share, for each tier j, the dealer fixes an n-degree polynomial f_j whose user evaluations are set to $f_j(i) = \mathsf{H}(j, x_i)$, for some hash function H, and whose constant term is the tier's secret; the dealer also publishes $(n - t_j + 1)$ additional public points of f_j (disjoint from user points), which collectively form aux. To reconstruct for tier j, one combines any t_j user evaluations $H(j, x_i)$ with the corresponding public points from aux to interpolate f_j and recover the secret. More formally, let $\mathsf{H} : \mathbb{F}^2 \to \mathbb{F}$ be a hash function modeled in ROM. The construction of an (n, k)-GMSS scheme over a finite field $\mathbb{F}$ works as follows:

- $\mathsf{GMSS.Share}(\boldsymbol{t}, \boldsymbol{s})$:
  Parse $\boldsymbol{t}$ as $(t_j)_{j\in[k]}$ and $\boldsymbol{s}$ as $(s_j)_{j\in[k]}$. For each $i \in [n]$, sample $x_i \leftarrow \mathbb{F}$, and set $\mathsf{sh}_i := x_i$. For each $j \in [k]$, define a polynomial f_j of degree n over $\mathbb{F}$ such that $f_j(0) = s_j$ and for all $i \in [n]$: $f_j(i) = \mathsf{H}(j, x_i)$. Then compute $\phi_j := \big(f_j(-1), \ldots, f_j(-(n - t + 1))\big)$. Output $\{\mathsf{sh}_i\}_{i\in[n]}$ and $\mathsf{aux} := (\phi_j)_{j\in[k]}$.
- $\mathsf{GMSS.Recon}(\boldsymbol{t}, I, \{\mathsf{sh}_i\}_{i\in I}, \mathsf{aux})$:
  Parse $\boldsymbol{t}$ as $(t_j)_{j\in[k]}$ and let j^* be such that $t_{j^*} = |I|$. Parse aux as $(\phi_j)_{j\in[k]}$ and ϕ_{j^*} as $\big(f_{j^*}(-1), \ldots, f_{j^*}(-(n - t_{j^*} + 1))\big)$. Concatenate these $(n - t_{j^*} + 1)$ points with the following $|I| = t_{j^*}$ points: $\{(i, f_{j^*}(i)) = (i, \mathsf{H}(j^*, \mathsf{sh}_i))\}_{i\in I}$. Then use Lagrange interpolation to compute the unique n-degree polynomial f_{j^*} that passes through those $n + 1$ points. Finally output $s_{j^*} = f_{j^*}(0)$.

Performance gain of the bottom-up paradigm. In top-down GMSS, we have $|\mathsf{sh}_i| = \Theta(k)$ and $|\mathsf{aux}| = 0$. In bottom-up GMSS, we have $|\mathsf{sh}_i| = O(1)$ and $|\mathsf{aux}| = \sum_{j=1}^{k}(n - t_j + 1)$, where the concrete value of $|\mathsf{aux}|$ depends on the threshold schedule chosen by $\mathcal{V}$. However, regardless of the schedule, the sum $|\mathsf{aux}| + n|\mathsf{sh}_i|$ is asymptotically the same ($\Theta(nk)$) under both top-down and bottom-up GMSS. Therefore, bottom-up GMSS is overall superior to top-down GMSS, and this is why S-VAR uses bottom-up GMSS.

Using bottom-up GMSS also saves up on the prover's storage in some cases. For instance, under an arithmetic threshold schedule (i.e., $t_{j+1} = t_j + \Delta$ for some step Δ), top-down GMSS incurs a worst-case prover storage overhead of $|\mathsf{aux}| + n|\mathsf{sh}_i| = nk = \frac{n^2}{\Delta}$, while bottom-up GMSS incurs $|\mathsf{aux}| + n|\mathsf{sh}_i| \approx \frac{1}{2}\frac{n^2}{\Delta} + n$, an approximately $2\times$ saving for large n. Therefore, bottom-up GMSS *concretely* reduces the prover's storage by a factor of 2. We note that bottom-up GMSS always incurs the $|\mathsf{aux}|$-sized upfront prover's storage, which is not the case for top-down GMSS.

Efficiency. To issue receipts for k thresholds, $\mathcal{V}$ performs $O(kn \log n)$ field operations and $O(kn)$ hash evaluations. To produce an audit proof, $\mathcal{P}$ must perform barycentric Lagrange interpolation [16] over arbitrary points, which costs $O(n \log^2 n)$ field operations. Verification costs $O(1)$ field operations since it only compares two field elements.

Security. S-VAR is secure with respect to our ideal functionality $\mathcal{F}_{\mathsf{VAR}}$. We present the full security proof to §F.1.

Choice of threshold schedule. S-VAR leaves the threshold schedule up to $\mathcal{V}$'s choice; in practice, this choice will be application-specific. In §B, we discuss the tradeoffs among different threshold schedules; we also present a minimal modification to S-VAR that yields *dynamic* thresholds that can adapt to realized user spending.

5. P-VAR: VAR Based on Pairings

Now, we present P-VAR, a pairing-based construction for VAR that supports succinct verification of exact counts.

At a high level, the protocol proceeds in *epochs*, and each run of the audit procedure advances the epoch number by one. Let e denote the epoch number, starting at 1 with an upper bound $e_{\max}$. To issue receipts, $\mathcal{V}$ samples random field elements $\{r_i\}$ and sends to each user $\mathcal{U}_i$ the tuple (r_i, e, σ_i) where σ_i is the signature on (i, r_i, e) under $\mathcal{V}$'s key. When users spend, $\mathcal{P}$ verifies the signature and collects the receipts. At audit time, $\mathcal{P}$ proves that it counted a hidden subset of the receipts correctly while revealing only the total count of collected receipts in this epoch. After an audit, both $\mathcal{P}$ and $\mathcal{V}$ increase the epoch number by 1.

A core cryptographic challenge is to prove properties of this hidden subset without revealing its members. We encode the subset as a hidden indicator vector, or bitmap, whose i-th entry is 1 exactly when user $\mathcal{U}_i$ is counted. $\mathcal{P}$ must convince $\mathcal{V}$ that the hidden bitmap (i) is indeed Boolean, (ii) selects only valid $\mathcal{V}$-issued receipts, and (iii) has the claimed cardinality. This requires a careful way to blind the corresponding committed polynomials, so that they remain hiding while still supporting the necessary algebraic checks.

A second challenge is supporting *multi-epoch auditing*. The audit proof relies on $\mathcal{V}$ revealing epoch receipt material to $\mathcal{P}$ after $\mathcal{P}$ has committed to its hidden set; this reveal is what allows the parties to check that the counted receipts are valid. However, if the same receipts were reused, then revealing this material would make future audits unsound, since $\mathcal{P}$ would learn all receipts for that epoch. Thus, a multi-audit protocol must ensure that each audit uses fresh receipt material while also preventing users from being counted more than once across epochs.

In §5.1, we present the full multi-epoch P-VAR construction. In §5.2, we present an optimization named "dynamic user sets," so that the protocol overhead only scales with the *actual* number of active users, instead of the maximum number of users fixed at setup.

Setup$(1^\lambda, n)$ The input of setup is $(1^\lambda, n)$ with λ being the security parameter and n being the number of users. Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be the pairing-friendly groups with bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with λ bits of security, and let $\mathbb{F}$ be the corresponding scalar field. A trusted party other than $\mathcal{P}$ and $\mathcal{V}$ computes CRS as follows:

1) Let $g_1 \in \mathbb{G}_1$ be a uniformly random generator of $\mathbb{G}_1$; let $h, g_2 \in \mathbb{G}_2$ be uniformly random generators of $\mathbb{G}_2$.
2) Let $\Omega = \{1, \omega, \ldots, \omega^{n-1}\}$ be the n-th roots of unity of $\mathbb{F}$, and let $Z(x) := x^n - 1$.
3) Sample $\tau \xleftarrow{\$} \mathbb{F}$ such that $\tau \notin \Omega \cup \{0\}$.
4) Compute $\boldsymbol{g_1} := (g_1, g_1^\tau, \ldots, g_1^{\tau^{n+2}})$, $\boldsymbol{g_2} := (g_2, g_2^\tau, \ldots, g_2^{\tau^{n+1}})$, and $\boldsymbol{h} := (h, h^\tau, \ldots, h^{\tau^{n-1}})$.
5) Let $\mathsf{H_{FS}}$ be a hash function modeled as a random oracle for Fiat-Shamir.

The trusted party outputs public parameters $\mathsf{pp} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbb{F}, e(\cdot,\cdot), \boldsymbol{g_1}, \boldsymbol{g_2}, \boldsymbol{h}, \Omega, \mathsf{H_{FS}})$ to $\mathcal{P}$ and $\mathcal{V}$. $\mathcal{V}$ samples $(sk, vk) \leftarrow \mathsf{DS.Gen}(1^\lambda)$ and publishes vk to $\mathcal{P}$.

- $\mathcal{P}$ initializes a polynomial $\tilde{b}_0(x) := 0$.
- $\mathcal{V}$ initializes an accumulator commitment $\tilde{\mathsf{cm}}_0 := g_2^0 \in \mathbb{G}_2$, a total count $K := 0$, and publishes his chosen maximum number of epochs $e_{\max}$.
- $\mathcal{P}$ and $\mathcal{V}$ set epoch number $e = 1$.

Issue For each $e \in [e_{\max}]$, $\mathcal{P}$ and $\mathcal{V}$ run the following:

1) $\mathcal{V}$ samples seed_e for this epoch. For each $i \in [n]$:
   a) $\mathcal{V}$ gets $r_i \leftarrow \mathsf{PRF}(\mathsf{seed}_e; i)$ and computes $\sigma_i \leftarrow \mathsf{DS.Sign}(sk, i \| r_i \| e)$.
   b) $\mathcal{V}$ sends receipt $\mathsf{rx}_{i,e} := (i, r_i, e, \sigma_i)$ to user $\mathcal{U}_i$. Note that $\mathcal{U}_i$ discards $\mathsf{rx}_{i,e}$ if they have spent a receipt in any previous epoch.
2) Let $r_e(x)$ be a degree-$(n-1)$ polynomial with $r_e(\omega^{i-1}) = \mathsf{PRF}(\mathsf{seed}_e; i)$ for $i \in [n]$. $\mathcal{V}$ stores $\mathsf{cm}_{r_e,2} := g_2^{r_e(\tau)}$.
3) $\mathcal{P}$ initializes the received receipt set $\mathrm{R}_e := \emptyset$.

Spend (epoch e)

1) $\mathcal{U}_i$ sends $\mathsf{rx}_{i,e}$ to $\mathcal{P}$.
2) $\mathcal{P}$ ignores receipts from any epoch $e' \neq e$, and receipts from users who have spent receipts in previous epochs.
3) On receiving a receipt $\mathsf{rx}_{i,e} = (i, r_i, e, \sigma_i)$ from $\mathcal{U}_i$, $\mathcal{P}$ checks $\mathsf{DS.Verify}(vk, i \| r_i \| e, \sigma_i) \stackrel{?}{=} 1$ and updates $\mathrm{R}_e := \mathrm{R}_e \cup \{(r_i, i)\}$ if the check passes.

Figure 4: Setup, Issue, and Spend procedures of P-VAR.

5.1. Protocol Details

We now describe the multi-epoch P-VAR protocol in detail. The formal protocol is described in Figs. 4 and 5.

Setup. A trusted party runs the following algorithm $\mathsf{Setup}(1^\lambda, n)$, where λ is the security parameter and n is the number of users. It first samples pairing-friendly groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ with a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and an associated scalar field $\mathbb{F}$. It fixes uniformly random generators $g_1 \in \mathbb{G}_1$ and $h, g_2 \in \mathbb{G}_2$, chooses a trapdoor $\tau \leftarrow \mathbb{F}$, and publishes powers-of-τ vectors $\boldsymbol{g_1} := (g_1, g_1^\tau, \ldots, g_1^{\tau^{n+2}})$, $\boldsymbol{g_2} := (g_2, g_2^\tau, \ldots, g_2^{\tau^{n+1}})$, and $\boldsymbol{h} := (h, h^\tau, \ldots, h^{\tau^{n-1}})$. The trapdoor τ itself is not published. The algorithm also outputs

the n-th roots of unity $\Omega = \{1, \omega, \ldots, \omega^{n-1}\} \subset \mathbb{F}^\times$. The public parameter $\mathsf{pp} := \big(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbb{F}, e(\cdot,\cdot), \boldsymbol{g_1}, \boldsymbol{g_2}, \boldsymbol{h}, \Omega\big)$ is given to both $\mathcal{P}$ and $\mathcal{V}$. In addition, $\mathcal{V}$ runs $(sk, vk) \leftarrow \mathsf{DS.Gen}(1^\lambda)$ and publishes vk.

$\mathcal{P}$ maintains a cumulative polynomial $\tilde{b}_e(x)$ such that $\tilde{b}_e(\omega^{i-1}) = 1$ if $\mathcal{U}_i$ has spent a valid receipt in epoch e or in prior epochs, and $\tilde{b}_e(\omega^{i-1}) = 0$ otherwise. $\mathcal{V}$ maintains a commitment to $\tilde{b}_e(x)$, denoted $\tilde{\mathsf{cm}}_e$, and a total count K, initialized to 0. $\tilde{b}_0(x)$ is initialized to the 0 polynomial, and $\tilde{\mathsf{cm}}_0$ is initialized to g_2^0. $\mathcal{V}$ chooses and publishes the maximum number of epochs $e_{\max}$, and the epoch number is initialized to $e = 1$.

Issuance. For each $e \in [e_{\max}]$, $\mathcal{V}$ samples a fresh seed seed_e. For each user $\mathcal{U}_i \in U$, $\mathcal{V}$ computes $r_i \leftarrow \mathsf{PRF}(\mathsf{seed}_e; i)$ and issues a receipt $\mathsf{rx}_{i,e} := (i, r_i, e, \sigma_i)$ to $\mathcal{U}_i$, where $\sigma_i := \mathsf{DS.Sign}(sk, i \| r_i \| e)$. Let $r_e(x)$ be a polynomial of degree $(n-1)$ with $r_e(\omega^{i-1}) = \mathsf{PRF}(\mathsf{seed}_e; i)$ for $i \in [n]$. $\mathcal{V}$ computes and stores $\mathsf{cm}_{r_e,2} := g_2^{r_e(\tau)}$, and $\mathcal{P}$ initializes the set of received receipts for this epoch to $\mathrm{R}_e := \emptyset$.

Deployment note. Although the specification pre-generates all epoch receipts, the same guarantees hold if receipts are streamed on demand: at each epoch, $\mathcal{V}$ sends that epoch's receipts to $\mathcal{P}$, who relays them only to users who have not yet spent. See §9 for this alternative deployment mode.

Spending. User $\mathcal{U}_i$ learns the current epoch number e from $\mathcal{P}$. To spend, $\mathcal{U}_i$ submits her receipt $\mathsf{rx}_{i,e} = (i, r_i, e, \sigma_i)$ to $\mathcal{P}$, who checks whether $\mathsf{DS.Verify}(vk, i \| r_i \| e, \sigma_i) = 1$. If the check passes, $\mathcal{P}$ adds (r_i, i) to R_e. $\mathcal{P}$ ignores all receipts that are not from epoch e and receipts from users who have spent in previous epochs.

Audit. To audit the total number of spent receipts in epoch e, $\mathcal{V}$ first sends an audit request to $\mathcal{P}$. If the current epoch number is $e > e_{\max}$, both $\mathcal{P}$ and $\mathcal{V}$ halt. Otherwise, they proceed as follows.

Receipt commitments. First, $\mathcal{P}$ commits to the receipts it actually received. Let $b_1, \ldots, b_n \in \{0,1\}$ be the bits indicating whether user $\mathcal{U}_i$'s receipt has been received in this epoch, i.e., $b_i = 1$ iff $(r_i, i) \in \mathrm{R}_e$. Define $f_1, \ldots, f_n$ such that $f_i = r_i$ if $(r_i, i) \in \mathrm{R}_e$ and $f_i = 0$ otherwise. $\mathcal{P}$ interpolates $f_e(x)$ and $b_e(x)$ over Ω with $f_e(\omega^{i-1}) = f_i$ and $b_e(\omega^{i-1}) = b_i$. Then $\mathcal{P}$ samples blinding factors $\rho_{b_e}, \rho_{f_e} \leftarrow \mathbb{F}$ and defines $\widehat{b}_e(x) := b_e(x) + \rho_{b_e} x Z(x)$ and $\widehat{f}_e(x) := f_e(x) + \rho_{f_e} Z(x)$. It computes blinded KZG commitments $\mathsf{cm}_{b_e,1} := g_1^{\widehat{b}_e(\tau)}$, $\mathsf{cm}_{b_e,2} := g_2^{\widehat{b}_e(\tau)}$, and $\mathsf{cm}_{f_e} := g_1^{\widehat{f}_e(\tau)}$. Then $\mathcal{P}$ computes the received receipt count $k_e := \sum_{i=1}^n b_i$ and sends $(\mathsf{cm}_{b_e,1}, \mathsf{cm}_{b_e,2}, \mathsf{cm}_{f_e}, k_e)$ to $\mathcal{V}$.

Note a subtle difference in the blinding terms for $b(x)$ and $f(x)$ ($\rho x Z(x)$ v.s. $\rho Z(x)$). While it is intuitive to use $\rho Z(x)$ to blind both, doing so for the bitmap $b(x)$ can leak the hidden set through the cardinality proof; we elaborate on this subtlety at the end of §5.1.

Seed reveal. After receiving these commitments, $\mathcal{V}$ reveals seed_e to $\mathcal{P}$. Using the seed, $\mathcal{P}$ interpolates the polynomial $r_e(x)$ with $r_e(\omega^{i-1}) = \mathsf{PRF}(\mathsf{seed}_e; i)$ for $i \in [n]$.

Correctness proofs. $\mathcal{P}$ computes the following proofs and sends them to $\mathcal{V}$.

- **(C1) Consistency of $f_e(x)$ with $b_e(x)$ and $r_e(x)$.** First, $\mathcal{P}$ shows that $f_i = b_i r_i$ for all $i \in [n]$. This is implied by the existence of a polynomial $q_{1,e}(x)$ such that $\widehat{b}_e(x) r_e(x) - \widehat{f}_e(x) = Z(x) q_{1,e}(x)$. $\mathcal{P}$ sends $\pi_{1,e} := g_1^{q_{1,e}(\tau)}$ to $\mathcal{V}$, who verifies by checking $e(\pi_{1,e}, g_2^{Z(\tau)}) \cdot e(\mathsf{cm}_{f_e}, g_2) \stackrel{?}{=} e(\mathsf{cm}_{b_e,1}, \mathsf{cm}_{r_e,2})$.
- **(C2) Bitness of $b_e(x)$.** Second, $\mathcal{P}$ shows that $b_e(x)$ evaluates to $\{0,1\}$ over Ω with the proof $\pi_{2,e} := g_1^{q_{2,e}(\tau)}$, where $\widehat{b}_e(x)(1 - \widehat{b}_e(x)) = Z(x) q_{2,e}(x)$. The proof is verified by checking $e(\mathsf{cm}_{b_e,1}, g_2 / \mathsf{cm}_{b_e,2}) \stackrel{?}{=} e(\pi_{2,e}, g_2^{Z(\tau)})$ and $e(\mathsf{cm}_{b_e,1}, g_2) \stackrel{?}{=} e(g_1, \mathsf{cm}_{b_e,2})$. This ensures that $b_i \in \{0,1\}$ and that $\mathsf{cm}_{b_e,1}, \mathsf{cm}_{b_e,2}$ are consistent.
- **(C3) Structure proof** Next, $\mathcal{P}$ proves that $\mathsf{cm}_{b_e,1}$ commits to a polynomial of the form $b_e(x) + a_e x Z(x)$, where $\deg(b_e) \le n-1$. Let $\mathsf{u}_Z := g_1^{\tau Z(\tau)} = g_1^{\tau^{n+1}} / g_1^{\tau}$, which is publicly computable from the CRS. The statement is that there exist $a_e \in \mathbb{F}$ and a degree-$(n-1)$ polynomial $b_e(x)$ such that $\mathsf{cm}_{b_e,1} = g_1^{b_e(\tau)} \cdot \mathsf{u}_Z^{a_e}$. In the honest execution, we have $a_e = \rho_{b_e}$. To prove this relation in zero knowledge, $\mathcal{P}$ samples $a_0 \leftarrow \mathbb{F}$ and $b_0(x) \leftarrow \mathbb{F}^{\le n-1}[x]$, and computes $T := g_1^{b_0(\tau)} \cdot \mathsf{u}_Z^{a_0}$. It computes the Fiat-Shamir [17] challenge $c := \mathsf{H}_{\mathsf{FS}}(\mathsf{sid}, e, \mathsf{cm}_{b_e,1}, \mathsf{cm}_{b_e,2}, \mathsf{cm}_{f_e}, k_e, T)$, sets $a^\star := a_0 + c a_e$ and $b^\star(x) := b_0(x) + c b_e(x)$, and sends $\pi_{\mathsf{str},e} := (T, a^\star, B^\star, H^\star)$ where $B^\star := g_1^{b^\star(\tau)}$ and $H^\star := h^{b^\star(\tau)}$. The verifier recomputes c and checks $B^\star \cdot \mathsf{u}_Z^{a^\star} \stackrel{?}{=} T \cdot \mathsf{cm}_{b_e,1}^c$ and $e(B^\star, h) \stackrel{?}{=} e(g_1, H^\star)$.
- **(C4) Cardinality.** Next, $\mathcal{P}$ shows that there are exactly k_e non-zero elements among the b_is. For the roots-of-unity domain, the degree-$(n-1)$ interpolant satisfies $b_e(0) = n^{-1} \sum_{i=1}^n b_i$. Since the xZ-blinding vanishes at 0, we also have $\widehat{b}_e(0) = b_e(0)$. Therefore, proving $k_e = \sum_i b_i$ is equivalent to proving $\widehat{b}_e(0) = k_e n^{-1}$. $\mathcal{P}$ computes $q_{\mathsf{card},e}(x) := (\widehat{b}_e(x) - k_e n^{-1})/x$ and sends $\pi_{\mathsf{card},e} := g_1^{q_{\mathsf{card},e}(\tau)}$ to $\mathcal{V}$. The verifier checks $e(\mathsf{cm}_{b_e,1} / g_1^{k_e n^{-1}}, g_2) \stackrel{?}{=} e(\pi_{\mathsf{card},e}, g_2^\tau)$.
- **(D) Disjointness proof.** To ensure no user already counted in previous epochs is double-counted in this epoch, $\mathcal{P}$ computes a disjointness proof showing that the current bitmap is disjoint from $\tilde{b}_{e-1}(x)$ when evaluated over Ω. Specifically, $\mathcal{P}$ computes $q_{D,e}(x)$ such that $\widehat{b}_e(x) \tilde{b}_{e-1}(x) = Z(x) q_{D,e}(x)$. The prover sends $\pi_{D,e} := g_1^{q_{D,e}(\tau)}$ to $\mathcal{V}$, who verifies with $e(\pi_{D,e}, g_2^{Z(\tau)}) \stackrel{?}{=} e(\mathsf{cm}_{b_e,1}, \tilde{\mathsf{cm}}_{e-1})$. The proof fails if a user is counted twice by $\mathcal{P}$.

Verification and updates. $\mathcal{V}$ accepts the count k_e if all of the above checks pass. Then $\mathcal{P}$ and $\mathcal{V}$ update their recorded values accordingly: $\mathcal{P}$ updates $\tilde{b}_e(x) := \tilde{b}_{e-1}(x) + \widehat{b}_e(x)$, and $\mathcal{V}$ updates $\tilde{\mathsf{cm}}_e := \tilde{\mathsf{cm}}_{e-1} \cdot \mathsf{cm}_{b_e,2}$ and $K := K + k_e$. Finally, the protocol enters the next epoch.

Efficiency. To issue receipts, the total computation of $\mathcal{V}$ is $O(e_{\max} n \log n)$ field operations and $O(e_{\max} n)$ group operations if all epoch receipts are generated in advance.

Audit (epoch e)

1) $\mathcal{V}$ sends an audit request to $\mathcal{P}$. If $e > e_{\max}$, both halt.
2) *Receipt commitments.* Define $(b_i)_{i=1}^n$ where $b_i = 1$ if $(r_i, i) \in \mathrm{R}_e$ for some r_i, otherwise $b_i = 0$. Also define $(f_i)_{i=1}^n$ where $f_i = r_i$ if $(r_i, i) \in \mathrm{R}_e$ for some r_i, otherwise $f_i = 0$. $\mathcal{P}$ interpolates $f_e(x)$ and $b_e(x)$ over Ω with $f_e(\omega^{i-1}) = f_i$ and $b_e(\omega^{i-1}) = b_i$. Then $\mathcal{P}$ samples $\rho_{b_e}, \rho_{f_e} \xleftarrow{\$} \mathbb{F}$ and defines $\widehat{b}_e(x) := b_e(x) + \rho_{b_e} xZ(x)$ and $\widehat{f}_e(x) := f_e(x) + \rho_{f_e} Z(x)$. $\mathcal{P}$ sends $\mathsf{cm}_{b_e,1} := g_1^{\widehat{b}_e(\tau)}$, $\mathsf{cm}_{b_e,2} := g_2^{\widehat{b}_e(\tau)}$, $\mathsf{cm}_{f_e} := g_1^{\widehat{f}_e(\tau)}$, and the claimed count $k_e := \sum_i b_i$ to $\mathcal{V}$.
3) *Seed reveal.* $\mathcal{V}$ sends seed_e to $\mathcal{P}$. $\mathcal{P}$ interpolates $r_e(x)$ over Ω such that $r_e(\omega^{i-1}) = \mathsf{PRF}(\mathsf{seed}_e; i)$ for $i \in [n]$.
4) *Proofs.*
   - **(C1) Consistency**: $\mathcal{P}$ computes $q_{1,e}(x)$ such that $\widehat{b}_e(x) r_e(x) = \widehat{f}_e(x) + q_{1,e}(x) Z(x)$. Then $\mathcal{P}$ sends $\pi_{1,e} := g_1^{q_{1,e}(\tau)}$ to $\mathcal{V}$.
   - **(C2) Bitness**: $\mathcal{P}$ computes $q_{2,e}(x)$ such that $\widehat{b}_e(x)(1 - \widehat{b}_e(x)) = Z(x) q_{2,e}(x)$ and sends $\pi_{2,e} := g_1^{q_{2,e}(\tau)}$ to $\mathcal{V}$.
   - **(C3) Structure proof**: $\mathcal{P}$ proves that $\mathsf{cm}_{b_e,1}$ commits to a polynomial of the form $b_e(x) + a_e xZ(x)$, where $\deg(b_e) \le n-1$. Let $\mathsf{u}_Z := g_1^{\tau Z(\tau)} = g_1^{\tau^{n+1}} / g_1^{\tau}$. The statement is that there exist $a_e \in \mathbb{F}$ and a degree-$(n-1)$ polynomial $b_e(x)$ such that $\mathsf{cm}_{b_e,1} = g_1^{b_e(\tau)} \cdot \mathsf{u}_Z^{a_e}$. To prove this relation, $\mathcal{P}$ samples $a_0 \xleftarrow{\$} \mathbb{F}$ and $b_0(x) \xleftarrow{\$} \mathbb{F}^{\le n-1}[x]$, and computes $T := g_1^{b_0(\tau)} \cdot \mathsf{u}_Z^{a_0}$. Then $\mathcal{P}$ computes $c := \mathsf{H}_{\mathsf{FS}}(\mathsf{sid}, e, \mathsf{cm}_{b_e,1}, \mathsf{cm}_{b_e,2}, \mathsf{cm}_{f_e}, k_e, T)$, sets $a^\star := a_0 + c a_e$ and $b^\star(x) := b_0(x) + c b_e(x)$, and sends $\pi_{\mathsf{str},e} := (T, a^\star, B^\star, H^\star)$ where $B^\star := g_1^{b^\star(\tau)}$ and $H^\star := h^{b^\star(\tau)}$.
   - **(C4) Cardinality**: $\mathcal{P}$ computes $q_{\mathsf{card},e}(x) := (\widehat{b}_e(x) - k_e \cdot n^{-1})/x$. Then $\mathcal{P}$ sends $\pi_{\mathsf{card},e} := g_1^{q_{\mathsf{card},e}(\tau)}$ to $\mathcal{V}$.
   - **(D) Disjointness**: $\mathcal{P}$ computes $q_{D,e}(x)$ such that $\widehat{b}_e(x) \tilde{b}_{e-1}(x) = Z(x) q_{D,e}(x)$. Then $\mathcal{P}$ sends $\pi_{D,e} := g_1^{q_{D,e}(\tau)}$ to $\mathcal{V}$.
5) *Verifier checks.* $\mathcal{V}$ accepts the claimed count k_e in this epoch if and only if the following checks pass:
   - **(C1) Consistency**: $e(\pi_{1,e}, g_2^{Z(\tau)}) \cdot e(\mathsf{cm}_{f_e}, g_2) \stackrel{?}{=} e(\mathsf{cm}_{b_e,1}, \mathsf{cm}_{r_e,2})$.
   - **(C2) Bitness**: $e(\mathsf{cm}_{b_e,1}, g_2 / \mathsf{cm}_{b_e,2}) \stackrel{?}{=} e(\pi_{2,e}, g_2^{Z(\tau)})$ and $e(\mathsf{cm}_{b_e,1}, g_2) \stackrel{?}{=} e(g_1, \mathsf{cm}_{b_e,2})$.
   - **(C3) Structure proof**: $\mathcal{V}$ recomputes $c := \mathsf{H}_{\mathsf{FS}}(\mathsf{sid}, e, \mathsf{cm}_{b_e,1}, \mathsf{cm}_{b_e,2}, \mathsf{cm}_{f_e}, k_e, T)$, recomputes $\mathsf{u}_Z := g_1^{\tau Z(\tau)} = g_1^{\tau^{n+1}} / g_1^{\tau}$, and checks $B^\star \cdot \mathsf{u}_Z^{a^\star} \stackrel{?}{=} T \cdot \mathsf{cm}_{b_e,1}^{c}$ and $e(B^\star, h) \stackrel{?}{=} e(g_1, H^\star)$.
   - **(C4) Cardinality**: $e(\mathsf{cm}_{b_e,1} / g_1^{k_e \cdot n^{-1}}, g_2) \stackrel{?}{=} e(\pi_{\mathsf{card},e}, g_2^{\tau})$.
   - **(D) Disjointness**: $e(\pi_{D,e}, g_2^{Z(\tau)}) \stackrel{?}{=} e(\mathsf{cm}_{b_e,1}, \tilde{\mathsf{cm}}_{e-1})$.
6) *Updates.* $\mathcal{P}$ updates $\tilde{b}_e(x) := \tilde{b}_{e-1}(x) + \widehat{b}_e(x)$. $\mathcal{V}$ updates $\tilde{\mathsf{cm}}_e := \tilde{\mathsf{cm}}_{e-1} \cdot \mathsf{cm}_{b_e,2}$ and $K := K + k_e$. They both update the epoch number $e := e + 1$.

Figure 5: Audit procedure of P-VAR.

To run an audit each epoch, $\mathcal{P}$ performs $O(n \log n)$ field operations and $O(n)$ group operations, and $\mathcal{V}$ runs $O(1)$ bilinear pairings and group operations to verify proofs. The structure proof and opening-at-zero check increase proof size and verification cost only by a constant number of group elements, scalars, and pairings.

Security. P-VAR is secure with respect to our ideal functionality $\mathcal{F}_{\mathsf{VAR}}$ under the stated assumptions. At a high level, the xZ-blinded commitment hides the bitmap, the structure proof certifies that the commitment has the correct form $b_e(x) + a_e xZ(x)$ with $\deg(b_e) \le n-1$, and the opening at 0 proves the claimed count because $b_e(0) = n^{-1} \sum_i b_i$ over the roots-of-unity domain. We present the full security proof of P-VAR in §F.2.

Why naive blinding of the P-VAR bitmap fails. A tempting alternative is to blind the P-VAR bitmap as $\widehat{b}(x) = b(x) + \rho Z(x)$ and use the univariate-sumcheck-style cardinality proof $\widehat{b}(x) = q_3(x) Z(x) + x q_4(x) + k \cdot n^{-1}$. However, in this form the low-degree polynomial $q_4(x)$ is completely determined by the hidden bitmap: reducing the identity modulo $Z(x)$ gives $b(x) = x q_4(x) + k \cdot n^{-1}$, and hence $q_4(x) = (b(x) - k \cdot n^{-1})/x$. Thus, for a claimed count k, $\mathcal{V}$ can enumerate all bit vectors of Hamming weight k, interpolate each candidate $b(x)$, compute the corresponding $q_4(x)$ and commitment, and compare it against π_4; the matching candidate reveals the spender set. Our construction avoids this dictionary attack by using the blinding term $\rho x Z(x)$ and replacing the above cardinality proof with a structure proof plus a constant-term check.

5.2. Supporting Dynamic User Sets

In §5.1, the runtime of $\mathcal{P}$ and $\mathcal{V}$ scales with the number of users fixed at setup, rather than the (potentially much smaller) number of users that are actually active during an audit. This is wasteful in settings where receipt spending occurs gradually. We therefore seek audit complexity that scales with the current active population ("dynamic user sets"). We present an extension of P-VAR that supports dynamic user sets; the full protocol is in Figs. 6 and 7.

Dynamic user model. At the start of epoch e, an *active user set* U_e is decided by $\mathcal{V}$ according to application-specific eligibility rules, known to both $\mathcal{P}$ and $\mathcal{V}$. We assume monotone growth $U_{e-1} \subseteq U_e$ with a global cap N. Let $n_e := |U_e|$ be the number of active users in epoch e. Each user $\mathcal{U}_i$ is associated with an interpolation point ω_i for all epochs. The goal is to perform all operations using polynomials whose degrees scale with the current active population, rather than the global cap N, while preserving the same soundness and privacy guarantees.

Nested interpolation domains. The key requirement is that for each epoch e, the interpolation points used by the active users form an FFT-friendly interpolation domain, while users are not *required* to be re-associated with new interpolation points across epochs. This requires a series of growing FFT-friendly interpolation domains that form a *nested* structure, so that each user is assigned a fixed interpolation

Setup$(1^\lambda, N)$ Runs all Setup steps in Fig. 4, except for the following changes to the CRS:

- Computes a sequence of elements $(\omega_1, \ldots, \omega_N)$ in $\mathbb{F}$ such that $\Omega^{(i)} := \{\omega_1, \ldots, \omega_{N^{(i)}}\}$ is a multiplicative subgroup of $\mathbb{F}$, $N^{(i)} < N^{(i+1)}$ for $i < \ell$, and $N^{(\ell)} = N$.
- The $\mathbb{G}_1$ SRS contains powers up to τ^{N+2} and the $\mathbb{G}_2$ SRS contains powers up to τ^{N+1}.
- Let $h^{(1)}, h^{(2)}, \ldots, h^{(\ell)}$ be uniformly random generators of $\mathbb{G}_2$. Compute and publish $\boldsymbol{h}^{(i)} := (h^{(i)}, (h^{(i)})^\tau, \ldots, (h^{(i)})^{\tau^{N^{(i)}-1}})$ to $\mathcal{P}$ and $\mathcal{V}$.

Issue (epoch e)

1) *Determine interpolation set.* Let n_e be the number of receipts to be issued in epoch e. If a user has received a receipt in epoch $e-1$, then they must also be issued a receipt in epoch e. Let i be the smallest index with $N^{(i)} \geq n_e$, and set $\Omega_e := \Omega^{(i)}$ and $N_e := |\Omega^{(i)}|$. Note that $\Omega_e = \{\omega_1, \ldots, \omega_{N_e}\}$. If $\Omega_e \neq \Omega_{e-1}$, define $\Delta_e := \Omega_e \setminus \Omega_{e-1}$.
2) $\mathcal{V}$ samples seed_e. For each user $i \in [n_e]$:
   - $\mathcal{V}$ gets $r_i \leftarrow \mathsf{PRF}(\mathsf{seed}_e; i)$ and computes $\sigma_i \leftarrow \mathsf{DS.Sign}(sk, i \| r_i \| e)$.
   - $\mathcal{V}$ sends receipt $\mathsf{rx}_{i,e} := (i, r_i, e, \sigma_i)$ to user $\mathcal{U}_i$. Note that $\mathcal{U}_i$ discards $\mathsf{rx}_{i,e}$ if they have spent a receipt in a previous epoch.
3) Let $r_e(x)$ be a degree-$(N_e - 1)$ polynomial with $r_e(\omega_i) = \mathsf{PRF}(\mathsf{seed}_e; i)$ for $i \in [N_e]$. $\mathcal{V}$ computes $\mathsf{cm}_{r_e,2} := g_2^{r_e(\tau)}$.
4) $\mathcal{P}$ initializes the received receipt set $\mathrm{R}_e := \emptyset$.

Spend (epoch e) Identical to the Spend procedure in Fig. 4.

Figure 6: Setup, Issue, and Spend procedures of P-VAR that supports a dynamic user set.

point across epochs. Thus, we need a sequence of points $(\omega_1, \ldots, \omega_N) \subset \mathbb{F}$ and a series of domain sizes $\{N^{(i)}\}_{i \in [\ell]}$ such that each prefix $\Omega^{(i)} = \{\omega_1, \ldots, \omega_{N^{(i)}}\}$ forms a multiplicative subgroup of size $N^{(i)}$, and $\Omega^{(i)} \subseteq \Omega^{(i+1)}$. In epoch e, we use the *smallest* i such that $|\Omega^{(i)}| \geq n_e$ to perform all polynomial operations. Let $\Omega_e := \Omega^{(i)}$ and $N_e := |\Omega_e|$ denote the domain and its size. From the nested structure, it also follows that if a user $\mathcal{U}_i$ assigned to interpolation point ω_i is counted in any epoch, then ω_i is included in the domains of all future epochs. We refer the reader to §C for an example construction of such sets.

Setup. We fix a *hierarchy* of domains as above. Concretely, choose $(\omega_1, \ldots, \omega_N) \subset \mathbb{F}$ so that for each $i \in [\ell]$ the set $\Omega^{(i)} = \{\omega_1, \ldots, \omega_{N^{(i)}}\}$ is a multiplicative subgroup of size $N^{(i)}$ and $\Omega^{(i)} \subseteq \Omega^{(i+1)}$. In addition to the standard KZG SRS vectors in $\mathbb{G}_1$ and $\mathbb{G}_2$ for degree $N+2$, the CRS publishes, for each $i \in [\ell]$, a short $\mathbb{G}_2$ vector $\boldsymbol{h}^{(i)} := \left(h^{(i)}, (h^{(i)})^\tau, \ldots, (h^{(i)})^{\tau^{N^{(i)}-1}}\right)$, with $h^{(i)} \xleftarrow{\$} \mathbb{G}_2$, to certify degree bounds relative to $\Omega^{(i)}$ in the structure proof. The $\mathbb{G}_1$ SRS contains powers up to τ^{N+2}, and the $\mathbb{G}_2$ SRS contains powers up to τ^{N+1}. Similar to the Setup procedure in §5.1, $\mathcal{P}$ initializes $\tilde{b}_0(x) := 0$, $\mathcal{V}$ initializes $\tilde{\mathsf{cm}}_0 := g_2^0$, and both $\mathcal{P}$ and $\mathcal{V}$ set epoch number $e = 1$.

Issuance. When a new epoch e starts, $\mathcal{V}$ issues new receipts to the users in U_e. The interpolation domain is $\Omega_e := \Omega^{(i^\star)}$, where $i^\star = \min\{i : N^{(i)} \geq n_e\}$. Let $N_e := |\Omega_e|$ denote the size of Ω_e. If the domain grew, we set $\Delta_e := \Omega_e \setminus \Omega_{e-1}$, which is known to both parties. $\mathcal{V}$ samples a new random seed seed_e. For each $i \in [N_e]$, $\mathcal{V}$ computes $r_i \leftarrow \mathsf{PRF}(\mathsf{seed}_e; i)$. For each active user $\mathcal{U}_i \in U_e$, $\mathcal{V}$ issues receipt $\mathsf{rx}_{i,e} = (i, r_i, e, \sigma_i)$ for $\sigma_i = \mathsf{DS.Sign}(sk, i \| r_i \| e)$. Let $r_e(x)$ be a polynomial of degree $(N_e - 1)$ with $r_e(\omega_i) = \mathsf{PRF}(\mathsf{seed}_e; i)$ for $i \in [N_e]$. $\mathcal{V}$ computes $\mathsf{cm}_{r_e,2} := g_2^{r_e(\tau)} \in \mathbb{G}_2$. $\mathcal{P}$ initializes the received receipt set for this epoch, $\mathrm{R}_e := \emptyset$.

Spending. Spending is identical to §5.1.

Audit. The audit procedure for epoch e starts when $\mathcal{V}$ sends an audit request to $\mathcal{P}$. $\mathcal{P}$ computes $b_e(x)$, $\widehat{b}_e(x)$, $f_e(x)$, and $\widehat{f}_e(x)$ with R_e in the same way as in §5.1, with the interpolation set Ω_e replacing Ω, the vanishing polynomial $Z_{\Omega_e}(x)$ replacing $Z(x)$, and the domain size N_e replacing n. In particular, $\widehat{b}_e(x) := b_e(x) + \rho_{b_e} x Z_{\Omega_e}(x)$ and $\widehat{f}_e(x) := f_e(x) + \rho_{f_e} Z_{\Omega_e}(x)$. $\mathcal{P}$ sends $(\mathsf{cm}_{b_e,1}, \mathsf{cm}_{b_e,2}, \mathsf{cm}_{f_e})$ and the number of received receipts k_e to $\mathcal{V}$. After receiving seed_e from $\mathcal{V}$, $\mathcal{P}$ also produces correctness proofs (C1–C4) as in §5.1, with Ω_e replacing Ω, $Z_{\Omega_e}(x)$ replacing $Z(x)$, N_e^{-1} replacing n^{-1}, $h^{(i^\star)}$ replacing h in the structure proof, and $\mathsf{u}_{Z,e} := g_1^{\tau Z_{\Omega_e}(\tau)}$ replacing u_Z.

Before running the disjointness proof, the prover extends $\tilde{b}_{e-1}(x)$ from Ω_{e-1} to Ω_e with an **extension proof**. If $\Omega_e = \Omega_{e-1}$, we set $c_e(x) := \tilde{b}_{e-1}(x)$ and $\mathsf{cm}_{c_e,2} := \tilde{\mathsf{cm}}_{e-1}$, and no extension proof is needed. Otherwise, the prover interpolates $c_e(x)$ such that $c_e(x)|_{x \in \Omega_{e-1}} = \tilde{b}_{e-1}(x)$ and $c_e(x)|_{x \in \Delta_e} = 0$, and commits $\mathsf{cm}_{c_e,1} := g_1^{c_e(\tau)}$ and $\mathsf{cm}_{c_e,2} := g_2^{c_e(\tau)}$. $\mathcal{P}$ shows that the extension is performed correctly by proving two divisibility relations: (1) there exists a polynomial $q_L(x)$ such that $c_e(x) - \tilde{b}_{e-1}(x) = Z_{\Omega_{e-1}}(x) q_L(x)$, and (2) there exists a polynomial $q_R(x)$ such that $c_e(x) = Z_{\Delta_e}(x) q_R(x)$. $\mathcal{P}$ sends the proofs $\pi_L := g_1^{q_L(\tau)}$ and $\pi_R := g_1^{q_R(\tau)}$ to $\mathcal{V}$, who verifies by checking $e(\pi_L, g_2^{Z_{\Omega_{e-1}}(\tau)}) \stackrel{?}{=} e(\mathsf{cm}_{c_e,1}, g_2)/e(g_1, \tilde{\mathsf{cm}}_{e-1})$, $e(\pi_R, g_2^{Z_{\Delta_e}(\tau)}) \stackrel{?}{=} e(\mathsf{cm}_{c_e,1}, g_2)$, and $e(\mathsf{cm}_{c_e,1}, g_2) \stackrel{?}{=} e(g_1, \mathsf{cm}_{c_e,2})$. This step certifies that the extended polynomial $c_e(x)$ agrees with $\tilde{b}_{e-1}(x)$ on Ω_{e-1} and vanishes on Δ_e.

Then $\mathcal{P}$ produces a disjointness proof showing that b_e is not overlapping with $c_e(x)$ when evaluated over Ω_e as in §5.1, with $c_e(x)$ replacing $\tilde{b}_{e-1}(x)$. Concretely, $\mathcal{P}$ computes $q_{D,e}(x)$ such that $\widehat{b}_e(x) c_e(x) = Z_{\Omega_e}(x) q_{D,e}(x)$, and $\mathcal{V}$ checks $e(g_1^{q_{D,e}(\tau)}, g_2^{Z_{\Omega_e}(\tau)}) \stackrel{?}{=} e(\mathsf{cm}_{b_e,1}, \mathsf{cm}_{c_e,2})$. Afterwards, $\mathcal{V}$ verifies the proofs. If all checks pass, $\mathcal{P}$ updates $\tilde{b}_e(x) := c_e(x) + \widehat{b}_e(x)$, and $\mathcal{V}$ updates $\tilde{\mathsf{cm}}_e := \mathsf{cm}_{c_e,2} \cdot \mathsf{cm}_{b_e,2}$ and $K := K + k_e$.

Efficiency. In epoch e, issuance and audit over the *current* domain Ω_e cost $O(N_e \log N_e)$ field operations and $O(N_e)$ group operations for $\mathcal{P}$, with $O(1)$ pairings and group operations for $\mathcal{V}$. In comparison, if the domain is fixed at

Audit (epoch e)

1) $\mathcal{V}$ sends an audit request to $\mathcal{P}$.
2) *Receipt commitments.* $\mathcal{P}$ computes $b_e(x)$, $\widehat{b}_e(x)$, $f_e(x)$, and $\widehat{f}_e(x)$ from R_e as in step 2 of Fig. 5, with interpolation set Ω_e, vanishing polynomial $Z_{\Omega_e}(x)$, and domain size N_e. In particular, $\widehat{b}_e(x) := b_e(x) + \rho_{b_e} x Z_{\Omega_e}(x)$ and $\widehat{f}_e(x) := f_e(x) + \rho_{f_e} Z_{\Omega_e}(x)$. $\mathcal{P}$ sends $(\mathsf{cm}_{b_e,1}, \mathsf{cm}_{b_e,2}, \mathsf{cm}_{f_e})$ and the number of received receipts k_e to $\mathcal{V}$.
3) *Seed reveal.* $\mathcal{V}$ sends seed_e to $\mathcal{P}$. $\mathcal{P}$ derives r_i and interpolates $r_e(x)$ over Ω_e.
4) *Correctness proofs.* $\mathcal{P}$ runs **(C1)**, **(C2)**, **(C3)**, and **(C4)** as in Fig. 5, with Ω_e replacing Ω, $Z_{\Omega_e}(x)$ replacing $Z(x)$, N_e^{-1} replacing n^{-1}, $h^{(i)}$ replacing h where $\Omega_e = \Omega^{(i)}$, and $\mathsf{u}_{Z,e} := g_1^{\tau Z_{\Omega_e}(\tau)}$ replacing u_Z.
5) *Extension proofs.* If $\Omega_e = \Omega_{e-1}$, set $c_e(x) := \tilde{b}_{e-1}(x)$ and $\mathsf{cm}_{c_e,2} := \tilde{\mathsf{cm}}_{e-1}$, and skip the extension proof. Otherwise, $\mathcal{P}$ interpolates $c_e(x)$ over Ω_e such that $c_e(x) = \tilde{b}_{e-1}(x)$ for $x \in \Omega_{e-1}$ and $c_e(x) = 0$ for $x \in \Delta_e$. $\mathcal{P}$ computes $\mathsf{cm}_{c_e,1} := g_1^{c_e(\tau)}$ and $\mathsf{cm}_{c_e,2} := g_2^{c_e(\tau)}$. It computes $q_L(x), q_R(x)$ such that $c_e(x) - \tilde{b}_{e-1}(x) = Z_{\Omega_{e-1}}(x) q_L(x)$ and $c_e(x) = Z_{\Delta_e}(x) q_R(x)$. Then $\mathcal{P}$ sends $\mathsf{cm}_{c_e,1}$, $\mathsf{cm}_{c_e,2}$, $\pi_L := g_1^{q_L(\tau)}$, and $\pi_R := g_1^{q_R(\tau)}$ to $\mathcal{V}$.
6) *Disjointness proof.* $\mathcal{P}$ runs **(D)** as in Fig. 5, with $c_e(x)$ replacing $\tilde{b}_{e-1}(x)$ and $Z_{\Omega_e}(x)$ replacing $Z(x)$. That is, $\mathcal{P}$ computes $q_{D,e}(x)$ such that $\widehat{b}_e(x) c_e(x) = Z_{\Omega_e}(x) q_{D,e}(x)$ and sends $\pi_{D,e} := g_1^{q_{D,e}(\tau)}$.
7) *Verifier checks.* $\mathcal{V}$ accepts the claimed count k_e in this epoch if and only if **(C1)**, **(C2)**, **(C3)**, **(C4)**, and **(D)** pass, and the following extension checks pass whenever $\Omega_e \neq \Omega_{e-1}$:
   - **(E) Extension**: $e(\pi_L, g_2^{Z_{\Omega_{e-1}}(\tau)}) \stackrel{?}{=} e(g_1, \mathsf{cm}_{c_e,2}/\tilde{\mathsf{cm}}_{e-1})$, $e(\pi_R, g_2^{Z_{\Delta_e}(\tau)}) \stackrel{?}{=} e(\mathsf{cm}_{c_e,1}, g_2)$, and $e(\mathsf{cm}_{c_e,1}, g_2) \stackrel{?}{=} e(g_1, \mathsf{cm}_{c_e,2})$.

   In all cases, $\mathcal{V}$ also checks **(D) Disjointness**: $e(\pi_{D,e}, g_2^{Z_{\Omega_e}(\tau)}) \stackrel{?}{=} e(\mathsf{cm}_{b_e,1}, \mathsf{cm}_{c_e,2})$.
8) *Updates.* $\mathcal{P}$ updates $\tilde{b}_e(x) := c_e(x) + \widehat{b}_e(x)$. $\mathcal{V}$ updates $\tilde{\mathsf{cm}}_e := \mathsf{cm}_{c_e,2} \cdot \mathsf{cm}_{b_e,2}$ and $K := K + k_e$. They both update the epoch number $e := e + 1$.

Figure 7: Audit procedure of P-VAR that supports a dynamic user set.

the global maximum N, every epoch costs $O(N \log N)$ field operations and $O(N)$ group operations for $\mathcal{P}$, regardless of how many users are active.

Savings of the dynamic user set extension are significant when the active user sets grow over time. Provisioned for $N = 10^6$ but auditing epochs with $N_e = 10^3$ active users yields a speedup of at least $\frac{N}{N_e} \approx \frac{10^6}{10^3} \approx 1{,}000\times$ in group operations, and approximately $\frac{N \log N}{N_e \log N_e}$ in field operations. Suppose that, over $T = 12$ epochs, N_e doubles each epoch from 10^3 up to 10^6. Then, with dynamic user sets, the total prover time is $\sum_{e=1}^{T} N_e \approx 2N$ group operations and $\sum_{e=1}^{T} N_e \log N_e \approx 2N \log N$ field operations; without dynamic user sets, the total prover time is $T \cdot N$ group operations and $T \cdot N \log N$ field operations. Thus, the total prover time is reduced by at least $6\times$.

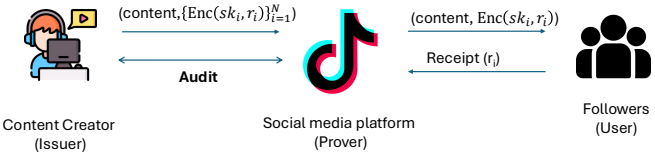


Figure 8: VAR for auditing social media platforms.

6. Applications

In this section, we discuss four applications of VAR to enable or enhance auditing of trusted platforms.

Auditing social media platforms. Content creators depend on social media platforms for their livelihood, but the lack of transparency and verifiability in how these platforms rank, deliver, and monetize content has led to well-documented frustration and discontent [18], [19], [20], [21], [22], [23]. Recent regulatory efforts, such as the EU Digital Services Act [24], highlight the need for greater scrutiny. VAR offers creators a means to cryptographically verify key aspects of platform behavior, including the fulfillment of paid boosts and the detection of stealth suppression.

For example, a creator may pay the platform for increased exposure through paid boosts and similar promotion mechanisms, but the platform can overreport views, underdeliver the post, or selectively suppress it. VAR provides a cryptographic audit mechanism for this setting: users who actually receive or view promoted content spend their receipts, and the platform proves the resulting count without revealing which users saw the post. We illustrate this flow in Fig. 8.

Outcome-based reimbursement. Generalizing beyond the example of company-sponsored benefits, this model of *outcome-based reimbursement* also captures many public-benefit programs that reimburse third-party providers based on how many eligible users they serve, such as discounted ride programs, job-training providers, or behavioral-health clinics. Funders need trustworthy aggregates while users expect privacy over their individual actions. VAR enables providers to prove the number of distinct beneficiaries served without exposing identities or usage histories.

Verifiable read receipts for broadcast messaging. In group or broadcast channels, a malicious or coerced platform can selectively suppress messages while showing inflated read receipts to the sender. Such silent interference is difficult to detect without independent verification. VAR enables senders to verify how many group members actually received a message while preserving the anonymity of broadcast.

Reputation in peer-to-peer file sharing. Peer-to-peer file-sharing systems such as BitTorrent [25] rely on reputation mechanisms, often based on upload-to-download ratios, to discourage free-riding and incentivize bandwidth contributions. Yet trackers that enforce these rules do not observe file transfers directly and must rely on self-reported statistics or mechanisms that reveal the traffic graph. VAR enables users to prove their contributions with succinct cryptographic evidence, allowing trackers to verify upload activity without exposing who exchanged data with whom.

7. Implementation and Evaluation

In this section, we report on implementation details and the performance evaluation of P-VAR and S-VAR. Our implementation artifacts are available at https://github.com/bunny-sleepy/Verifiable-Aggregate-Receipts.

7.1. Implementation

7.1.1. Cryptographic primitives. We implement all schemes (ours and baselines) in Rust (`rustc v1.91.1`) using `arkworks v0.5` as the main cryptography library. For P-VAR, the pairing-friendly groups $\mathbb{G}_1, \mathbb{G}_2$ and the scalar field $\mathbb{F}$ are instantiated by BLS12-381 from `ark-bls12-381 v0.5`. We use Ed25519 [26], [27] implemented with `ed25519-dalek v2.2.0` to instantiate digital signatures and SHA-256 to instantiate PRFs and hash functions. Our field and group choices across both schemes provide 128 bit-security.

For S-VAR, we implement a standalone multi-secret-sharing module with efficient sharing and reconstruction algorithms. The field we use is the scalar field of BN254 instantiated with `ark-bn254 v0.5`. To reconstruct, we use the *barycentric Lagrange formula* [16]:

$$f(0) = \frac{\sum_{i=1}^{n}(w_i f(x_i)/x_i)}{\sum_{i=1}^{n}(w_i/x_i)}.$$

Here $w_i = \frac{1}{M'(x_i)}$ are the *barycentric weights*, and $M(x) = \prod_{i=1}^{n}(x - x_i)$. To compute all $M'(x_i)$ efficiently, we build a *subproduct tree* [28] of the factors $(x - x_i)$ and form $M(x)$ at the root in $O(C(n)\log n)$ time – here $C(n)$ is the cost of multiplying two n-degree polynomials. We differentiate once, and perform multi-point evaluation [28] of the derivative $M'(x)$ at the x_i's using the same tree, again in $O(C(n)\log n)$ time. For modular reductions along the tree, we use the standard reversal and Newton series inversion trick [29] so that reduction modulo a degree-m node costs $O(C(m))$.

7.1.2. Baselines. We compare to two baselines: the PAS scheme [5] (discussed in §8) and a generic SNARK with aggregate signatures. Since the PAS paper did not provide an implementation, we implement the core inner-product arguments; for the generic SNARK, we implement only core zk-SNARK proving and verification, omitting signature aggregation and verification. Thus our benchmarks represent lower bounds on end-to-end runtime of both baselines. We implement the second baseline using Noir [30] `v1.0.0-beta.15` with Aztec's Barretenberg [31]. Note that neither baseline achieves all of VAR's security properties; we give more details in §D.1.

Choices of baselines. We consider alternative baselines such as hinTS [6]. While the proving time of hinTS is acceptable, it requires each user to run a preprocessing step with $O(n^2)$ group operations (n is the number of users), making it infeasible to scale to millions of users. Our benchmark shows that preprocessing for $n = 4{,}095$ takes more than 70 seconds for each user on a powerful machine (`c7i.16xlarge`). For $n = 2^{20}$ users, the preprocessing for each user is estimated to take more than 50 days. Therefore we do not consider hinTS a baseline for our protocols.

7.2. Experimental Setup

Here, we describe the setup used for our evaluation.

AWS environments. Depending on the settings, we run experiments on either a large server (`c7i.16xlarge`, denoted L) or a small server (`c7i.xlarge`, denoted S). The S server has 4 vCPUs and 8 GiB of memory, while L provides 64 vCPUs and 128 GiB of memory, both powered by 4th-generation Intel Xeon Scalable ("Sapphire Rapids") processors. All runtime measurements are averaged over 10 runs of the operation being measured, after 3 warmup runs.

Choice of server configurations. Our applications vary in the type of entities performing the issuance and proving steps and thus, we consider both *small* (S) and *large* (L) servers for these operations. For instance, a platform (prover) in the content creator application likely already maintains significant infrastructure to serve large user bases. In contrast, the broadcast receipts (§E.2) and the service reimbursement (§E.1) applications likely have fewer users and presumably less well-resourced provers, reflected in experiments on S. A similar dichotomy emerges in issuance/verification.

7.3. Issuance Performance

We first report the performance of the issuance procedure. Since our baselines do not include explicit issuance – requiring only that each user hold a pre-established key pair known in advance to the issuer or verifier – we exclude them from this evaluation. For our protocols, we consider two metrics: issuer bandwidth and runtime.

Bandwidth. The issuer bandwidth is the total amount of data sent by the issuer. Recall from §4, that the S-VAR issuer supplies each user with a single receipt (regardless of the number of thresholds) and sends auxiliary data aux to the prover. In P-VAR instead, each user gets as many receipts as there are supported epochs (no aux is sent). In both cases, each receipt is a random value and a signature.

Issuance time. The issuance time for both protocols includes generation and signing of receipts. S-VAR primarily consists of FFT-based polynomial interpolation and evaluation and for P-VAR, group operations dominate.

7.3.1. Comparison between S-VAR and P-VAR assuming the same number of supported audits. The two schemes have different tradeoffs: S-VAR's issuance overhead scales with the precision (the number of thresholds), while P-VAR's overhead scales with the number of support epochs. To fairly compare them, we choose a regime where the number of thresholds supported in S-VAR is the same as the supported epochs in P-VAR. The rationale is that this fixes the number of audits a verifier can perform, while illustrating the performance benefit of allowing inexact counting. To be precise, given n users, we configure P-VAR and S-VAR to

support $\log_2 n$ thresholds and epochs, respectively. We run S-VAR and P-VAR on small and large servers in the above comparison regime; the results are shown in Fig. 9.

Bandwidth. Figure 9a shows the bandwidth of S-VAR and P-VAR as a function of the number n of users. A trade-off emerges between the granularity of audits and the issuer's bandwidth. In particular, while S-VAR requires about $\frac{1}{3}$ of the issuer bandwidth consumed by P-VAR to support $\log n$ audits, P-VAR supports exact audit counts, while S-VAR only supports auditing at powers-of-2 thresholds.

Issuance time. Figure 9b shows the issuance time of S-VAR and P-VAR on large and small servers. Two trends can be observed in this figure. First, P-VAR performs worse than S-VAR, which is because S-VAR primarily consists of FFT-based polynomial interpolations and polynomial evaluations in fields, but P-VAR mainly consists of group operations that are relatively more expensive. Second, the issuance procedure in both schemes is embarrassingly parallelizable. Even though our algorithms cannot fully leverage the difference (16x) in the number of cores between S and L, we do observe an approximate $10\times$ speedup when running on L. Finally, similar to the tradeoff in the bandwidth comparison, P-VAR requires approximately an order-of-magnitude higher issuance time but provides more fine-grained audits.

Storage footprint on issuer and users. At the end of the issuance procedure, the issuer can discard the computed receipts and store a constant amount of data per supported audit, on the order of 100s of bytes for both schemes. Note that in S-VAR, the issuer must store the threshold schedule but does not need to store all secrets, as discussed in §4.2. In both schemes, each user simply stores his issued receipt. In this $\log n$-audits regime, and for 2^{20} users, the P-VAR user bandwidth is $2.08\,\mathrm{kB}$; in S-VAR, the user bandwidth is independent of the number of audits and under $100\,\mathrm{B}$.

7.4. Spend and Audit Performance

7.4.1. Spend. In both S-VAR and P-VAR, each user simply sends his constant-sized receipt ($96\,\mathrm{B}$ for S-VAR and $104\,\mathrm{B}$ for P-VAR); thus, the spend time and bandwidth are constant.

7.4.2. Proving time. We benchmark the proving time of a single audit for both schemes and plot the results in Fig. 10. To give our baselines the most favorable conditions, we report their benchmarks only on our larger VM (`c7i.16xlarge`). For all evaluated solutions, audit proving time is independent of the number of spent receipts (n denotes the total number of supported users). Some highlights:

- For 2^{10} users, S-VAR takes $0.066\,\mathrm{s}$, and P-VAR takes $0.18\,\mathrm{s}$ on S.
- For 2^{20} users, S-VAR takes $20.5\,\mathrm{s}$, and P-VAR takes $12.1\,\mathrm{s}$ on L.

Comparing against baselines. Our SNARK baseline numbers presented in Fig. 10 consist of the sum of the time taken for witness generation and the time for proving. We were able to benchmark the SNARK variant only for a narrow band of parameters. At larger parameter sizes, the proving server either exhausted memory or became unresponsive. Even within the parameter range where the SNARK baseline did complete, its proving time lagged behind both S-VAR and P-VAR by *at least an order of magnitude*.

The PAS construction exhibits severe performance limitations, with proving time orders of magnitude worse than any other method we evaluated. One reason is that PAS needs $O(n)$ pairing operations, which quickly becomes impractical. At the largest parameter size we tested, 2^{20}, the PAS solution's prover performs $48\times$ worse than the next-best construction S-VAR. Even on the smaller VM, S-VAR performs about $9\times$ better than PAS run on the large server!

Comparing our two solutions. Across all machine configurations we tested, P-VAR delivers lower proving time than the corresponding S-VAR implementation for a single audit. While P-VAR is slightly slower than S-VAR for very small parameters, the trend reverses quickly: beyond a small-parameter regime, P-VAR outperforms P-VAR across every machine and tested setting. The key reason is the absence of FFT-friendly structure in the S-VAR proving step. This transforms the polynomial-interpolation domain in S-VAR into an irregular subset, eliminating the ability to use FFTs and forcing interpolation to proceed in super-linear time.

Bandwidth. The bandwidth at audit time for both solutions is constant: $32\,\mathrm{B}$ in S-VAR and $648\,\mathrm{B}$ in P-VAR.

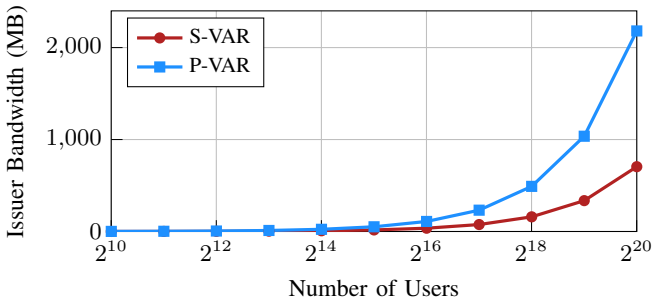
Round complexity and latency. For each solution, the latency from audits is the proving time, plus the network latency of any interactions. Estimating based on public TCP latency data [32], the network latency to audit in P-VAR (2 rounds) is 300ms. S-VAR, SNARK and PAS are each one round, so they incur 150ms latency.

7.4.3. Verification. Next, we consider the audit verification operations for each solution. The verification times for both P-VAR and S-VAR are constant and concretely small. The simplest verification of all the solutions presented here is for S-VAR, which is a simple equality check for smaller than $32\,\mathrm{B}$ strings, taking negligible time (less than $100\,\mu\mathrm{s}$ on both VMs). The P-VAR verifier runs in about $17\,\mathrm{ms}$ on the small VM and about $15\,\mathrm{ms}$ on the large VM.

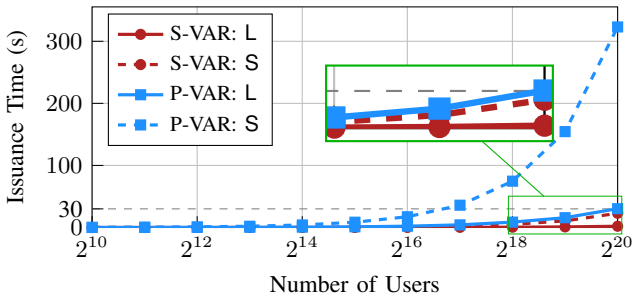
On the other hand, the verification time for PAS grows logarithmically in the number of users (again, as expected), with verification of an audit for 2^{20} on L taking on average $625\,\mathrm{ms}$. The SNARK baseline presented here causes verification time to increase linearly with the number of users, reaching $469\,\mathrm{ms}$ on L, at 2^{16} users. Note that this time can be shortened to a constant, approximately $28\,\mathrm{ms}$ by making the public keys a part of the witness but only at the expense of significantly higher proving time.

7.5. TrueReach: VAR on Bluesky

We further demonstrate the practical applicability of VAR by integrating it with Bluesky, a decentralized social network built on the AT protocol [8]. The AT protocol separates data hosting, content propagation, ranking, and



(a) Issuer bandwidth



(b) Issuance time. L = `c7i.16xlarge`, S = `c7i.xlarge`.

Figure 9: Issuer bandwidth and issuance time for S-VAR and P-VAR as a function of the number n of users, in a regime where S-VAR supports $\log n$ thresholds, and P-VAR supports $\log n$ epochs. Note that the number of permissible audits for both protocols and the particular thresholds for S-VAR are tunable.

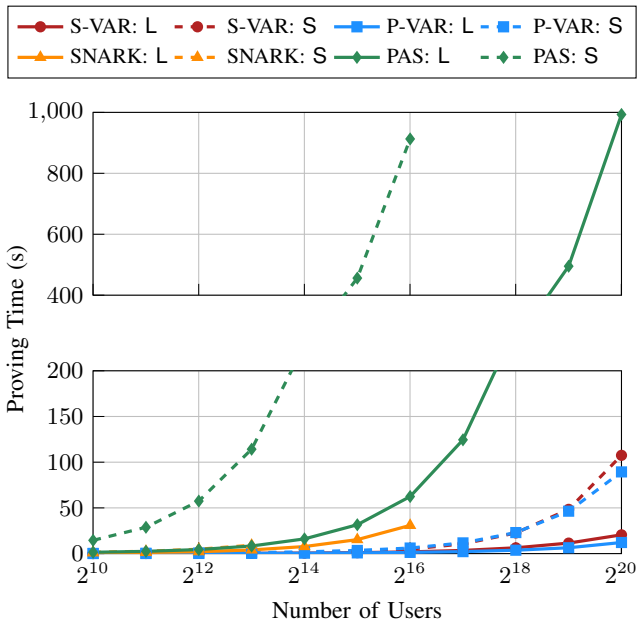


Figure 10: Audit prover time for S-VAR and P-VAR. **Note the discontinuity in the y-axis, and the change of scale.**

presentation into independent components. Creators publish content to Personal Data Servers (PDSs); relays propagate updates across the network; *feed generators* apply ranking logic and produce customized feeds; and app views present ranked results to end users. Feed generators act as recommendation algorithms that select and rank content, and creators may pay them to increase content visibility. However, creators currently have no way to verify whether a feed generator truly delivered the promised exposure – a natural fit for VAR.

Mapping VAR to Bluesky. We implement **TrueReach**, a system that extends Bluesky with verifiable view-count auditing using P-VAR. The system consists of a modified feed generator (playing the role of $\mathcal{P}$) and a custom app view that automates the protocol for creators ($\mathcal{V}$) and users (U). When users join TrueReach, they generate a key pair and register their public key and DID with the feed generator. To promote a post, a creator fetches registered public keys, runs P-VAR issuance to generate receipts, and encrypts each receipt under the corresponding user's public key, following the *prover-relay model* discussed in §9. When a user views the promoted post through the custom app view, the app view automatically fetches the encrypted receipt, decrypts it with the user's private key, and returns it to the feed generator, who updates its P-VAR state. At audit time, the creator and feed generator execute the two-round P-VAR audit procedure: the feed generator first commits to its current state, the creator reveals the epoch seed, and the feed generator produces a proof. The creator then verifies the proof, all handled automatically by the app view.

Implementation. We implement the cryptographic backend of TrueReach in Rust and compile it to WebAssembly (WASM), so that issuance, spending, and verification run in the browser. The feed generator runs a server-side Rust backend that handles receipt collection and proof generation.

Evaluation. We evaluate TrueReach on a MacBook with an Apple M1 Pro chip and 16 GB of memory, using 8 parallel threads to model a realistic creator-side device. The main creator-side costs in WASM are receipt issuance and encryption. For 2^{20} registered users, TrueReach issues all receipts in 53.3 s on average and encrypts them in 16.1 s. At the smaller scale of 2^{10} users, TrueReach issues all receipts in 0.13 s on average and encrypts them in 0.07 s. Audit verification remains lightweight: proof verification takes about 39 ms and is nearly independent of the user-set size in our experiments. These results show that TrueReach can support large audiences on commodity hardware, making it practical for real-world deployment. The artifact and benchmark are available at https://anonymous.4open.science/r/TrueReach-SP27.

8. Related Work

Signature aggregation. Aggregate signatures, most prominently those based on BLS [4], compress multiple signatures on (possibly) distinct messages into a single succinct signature for efficient verification. Plain aggregate signatures lack

an efficient method to produce a succinct proof of count, and thus fall short for VAR. Qiu and Tang propose Predicate Aggregate Signatures (PAS) [5], allowing users to sign messages individually, and a designated aggregator to produce a short aggregate signature certifying a public predicate (e.g., a minimum number of distinct signers signed a message) is satisfied. PAS matches some efficiency and security goals of VAR, but its inner-product pairing arguments make it expensive and thus intractable at our million-user scale.

More recent pairing-based threshold signature schemes [6], [33] offer succinct, weighted, and multi-threshold signing. Among hinTS [6] and the close variant of Das et al. [33], hinTS is most relevant for VAR, since it offers a blinded variant that hides the exact set of signers, which is why we consider it as a baseline in §7. As we explain in §7, our benchmarks and estimates show that these works do not scale to the number of users (one million) that VAR intends to support. Finally, all these schemes [4], [5], [6], [33] would require users to sign; as we have discussed in §9, this is undesirable, as user signatures are irrefutable proof that they received a specific service.

Privacy-preserving aggregate statistics. One line of work (see, e.g., [34], [35], [36], [37]) considers the problem of aggregating the data of many users for statistics, of which a simple count of how many users spent a token is an example. Some works such as Acorn [38] have considered checking the client data for certain invariants (such as ensuring integers are within a certain range). However, this line of work does not assume an issuer (who is also the verifier), as in our case, so it does not include any source of ground truth. Green et al. [39] propose a voting-style protocol for privacy-preserving ad reporting at web scale; Acharya et al. [40] introduce non-interactive verifiable aggregation (NIVA), a new primitive for secure computation of aggregate statistics over a large population of clients. However, both works treat the server as honest for integrity: a malicious server can inject inputs and thus inflate aggregate counts; guarding against such prover-side inflation is a central goal in VAR. Blomer et al. [41] build an anonymous reputation system where users can rate each product at most once; but, they rely on a trusted group manager who controls group public keys and can open signatures, whereas VAR aims to avoid such a central authority.

Anonymous tokens. The motivation for anonymous tokens is to enable access control, without tracking individual users. For instance, Heydt-Benjamin et al. [42] consider anonymous tokens as a solution to user tracking when users use longer-lived "metrocards" or similar methods to pay for public transport. Privacy pass [43] provides users with tokens upon successful completion of a CAPTCHA or similar challenge, allowing anonymous access to Internet services while protecting servers from DDoS attacks and bots. Anonymous counting tokens [44] focus on users being anonymously issued tokens, but with a "count" bounding how many tokens each user is able to obtain, not aggregating the total number of spent tokens. A line of work on anonymous tokens with a private metadata bit (e.g., [45], [46], [47]) includes a designated party that can decrypt the metadata bit of individual tokens; however, this work does not focus on aggregation or support proof of count.

Proof of Liability (PoL). Proof of Liability (PoL) protocols (e.g., [48], [49], [50]) are motivated by the context of a bank or cryptocurrency wallet provider showing to (1) *any* verifier its total (or aggregate) liability across all users, and (2) each user that their account balance is included in this liability. At first glance, it may seem as though the aggregation of total liability from PoL could solve the VAR problem. However, the proof of aggregate liability is not meant to show, cryptographically, that all the included liabilities correspond to some "valid" account. There is no central issuer and hence, the only available recourse is for each user to individually check that their liability is included correctly. Thus, this setting differs fundamentally from our problem.

Threshold primitives towards tiered auditing. Threshold signature schemes [6], [33], [51] can be used to construct VAR with tiered auditing, but the required signature and public key aggregation operations are heavier than necessary in our setting, where plain secret sharing suffices. Multi-secret sharing schemes (MSSS), starting from the information-theoretic construction of Blundo et al. [14], offer a more direct and lightweight approach. Kate et al. [7] formalized bottom-up secret sharing, first used by Baird et al. [52] to build multiverse threshold signatures. We also adopt the bottom-up paradigm to obtain an efficient MSSS that underpins our S-VAR construction. Dynamic threshold public key encryption (TPKE) [53] and more efficient related primitives [54] could be used to construct VAR with *flexible* tiered audits; however, TPKE is heavier than desired for a simple and lightweight tiered-audit solution. In fact, the threshold encryption scheme with silent setup of Garg et al. [54] relies on pairing-based cryptography similarly to our P-VAR construction, which in our case enables *exact* auditing.

9. Discussion

Streaming delivery of receipts. So far, we have presented our protocols as if any receipts are directly communicated from the issuer to a user. This assumes an out-of-band channel between the issuer and the users, which, in practice, must be opened every time a new instance of the protocol is run. However, our protocol can be deployed with receipts being relayed by the service provider in a streaming fashion, so that users and the issuer do not need to interact (beyond the initial key setup), and so that users do not need to obtain all receipts at the beginning of the protocol. Doing so requires the issuer $\mathcal{I}$ and each user $\mathcal{U}_i$ to establish a pairwise encryption key sk_i. Then, for each instance of the protocol, issuance can consist of the $\mathcal{I}$ computing receipts rx_i, which may consist of multiple sub-blocks b_i^1, ..., b_i^m (for instance, a different receipt for each epoch as described in P-VAR), followed by computing their encryptions under sk_i and sharing these with $\mathcal{P}$. When a user $\mathcal{U}_i$ wishes to spend, she retrieves the necessary ciphertext from $\mathcal{P}$, decrypts it using sk_i, and spends it as usual. The social media application is a

natural use case for streaming delivery (and it is presented as such), where users receive receipts as they download posts.

Semi-honest verifier assumption. In this work, our adversarial model assumes a *semi-honest* verifier, i.e., we assume $\mathcal{V}$ derives and issues receipts correctly. At the same time, we *do* require user privacy: $\mathcal{V}$ should only learn the number of spent receipts, and nothing else about the subset of users who spent. This assumption is justified by the incentives; in all applications we consider, the primary goal of the verifier $\mathcal{V}$ is to hold a more powerful platform $\mathcal{P}$ accountable, thus a malicious $\mathcal{V}$ who issues inconsistent or malformed receipts would undermine his *own* ability to receive an accurate engagement count, giving him little incentive to deviate from the protocol. However, in all three settings, $\mathcal{V}$ might still be tempted to analyze audit transcripts in an effort to deanonymize and profile users. Our threat model, therefore, focuses on treating $\mathcal{P}$ as fully malicious with regard to inflation soundness, while limiting what an otherwise semi-honest $\mathcal{V}$ can infer about user spending.

No transferable evidence of participation. An undesirable property of approaches based on user signatures – including those that can hide signer identities from $\mathcal{V}$ [5], [6] – is that a transcript containing $\mathcal{U}_i$'s signature can be shown to third parties later as publicly verifiable evidence that $\mathcal{U}_i$ participated in the spending process. In contrast, our solutions do not require users to sign receipts. Thus, the receipts themselves do not constitute transferable and irrefutable evidence that a user participated in a specific service event, such as viewing an advertisement, downloading a video, or visiting a clinic. This is a step in the direction of the property sometimes called *deniability*. This property is related to but distinct from user privacy captured by $\mathcal{F}_{\mathsf{VAR}}$.

10. Conclusion

Accurately measuring user engagement is central to decisions across online platforms, public services, and communication systems, yet self-reporting remains unreliable due to incentives to inflate. Motivated by this gap, we introduced Verifiable Aggregate Receipts (VAR), a primitive that enables privacy-preserving aggregation of issued receipts into a compact proof of count. We formalized VAR's security via an ideal functionality and presented two constructions, which we proved secure: S-VAR, a secret-sharing protocol that supports tiered "fuzzy" audits with constant-size receipts independent of the number of tiers, and P-VAR, a pairing-based protocol that enables exact audits and can be extended to handle dynamic user sets. We implemented and benchmarked our protocols, showing that they are concretely fast for one million users, while existing solutions either do not scale to this regime or are at least an order-of-magnitude slower. We showcased four diverse applications of VAR; auditing social media platforms, preventing fraudulent reimbursement, detecting censorship in broadcast channels, and verifying contributions in peer-to-peer file sharing. Finally, we implemented TrueReach, an end-to-end integration of VAR with the Bluesky social network.

Acknowledgments

Ioannis is partially supported by an IC3 grant.

References

[1] United States Attorney's Office, Southern District of New York. U.s. attorney announces $12.9 million settlement with the door for submitting fraudulent cost reports.

[2] "2024 national health care fraud enforcement action summary of criminal charges — criminal division DoJ, USA," Jun. 2024.

[3] "District of Arizona | Seven Charged in Arizona as Part of the Department of Justice's 2024 National Health Care Fraud Enforcement Action | United States Department of Justice," Jun. 2024.

[4] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *EUROCRYPT 2003*. Springer, 2003, pp. 416–432.

[5] T. Qiu and Q. Tang, "Predicate aggregate signatures and applications," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2023, pp. 279–312.

[6] S. Garg, A. Jain, P. Mukherjee, R. Sinha, M. Wang, and Y. Zhang, "hints: Threshold signatures with silent setup," in *2024 IEEE symposium on security and privacy (SP)*. IEEE, 2024, pp. 3034–3052.

[7] A. Kate, P. Mukherjee, H. Saleem, P. Sarkar, and B. Roberts, "ANARKey: A new approach to (socially) recover keys," Cryptology ePrint Archive, Paper 2025/551, 2025.

[8] M. Kleppmann, P. Frazee, J. Gold, J. Graber, D. Holmgren, D. Ivy, J. Johnson, B. Newbold, and J. Volpert, "Bluesky and the AT Protocol: Usable Decentralized Social Media," in *Proceedings of the ACM Conext-2024 Workshop on the Decentralization of the Internet*, 2024, pp. 1–7.

[9] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *IEEE Symposium on Security and Privacy 2014*, ser. SP '14. USA: IEEE Computer Society, 2014, p. 459–474.

[10] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," in *ACM CCS'93*, ser. CCS '93, New York, NY, USA, 1993, p. 62–73.

[11] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, 1988.

[12] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Asiacrypt 2010*. Springer, 2010, pp. 177–194.

[13] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[14] C. Blundo, A. De Santis, G. Di Crescenzo, A. G. Gaggia, and U. Vaccaro, "Multi-Secret Sharing Schemes," in *CRYPTO'94*, Y. G. Desmedt, Ed. Berlin, Heidelberg: Springer, 1994, pp. 150–163.

[15] A. Kiayias, M. Osmanoglu, A. Russell, and Q. Tang, "Space Efficient Computational Multi-Secret Sharing and Its Applications," Cryptology ePrint Archive, Paper 2018/1010, 2018.

[16] J.-P. Berrut and L. N. Trefethen, "Barycentric lagrange interpolation," *SIAM Review*, vol. 46, no. 3, pp. 501–517, 2004.

[17] A. Fiat and A. Shamir, "How To Prove Yourself: Practical Solutions to Identification and Signature Problems," in *Advances in Cryptology — CRYPTO' 86*, Berlin, Heidelberg, 1987, pp. 186–194.

[18] K. Conger, J. E. Bromwich, and T. Arango, "Woman opens fire at youtube headquarters, wounding 3 before killing herself," *The New York Times*, April 2018.

[19] P. Mozur, "A former youtube insider says the company failed to curb extremism," *The New York Times*, June 2019.

[20] R. Molla, "Youtube's lgbtq community says its videos keep getting restricted or demonetized," *Vox*, October 2019.

[21] V. Staff, "Youtube settles three-year-long racial bias lawsuit from black and hispanic creators," *Verdict*, February 2025.

[22] R. Caplan and T. Gillespie, "The algorithmic dance: Youtube's adpocalypse and the gatekeeping of cultural content in digital media," *Internet Policy Review*, vol. 9, no. 4, 2020.

[23] M. D. Staff, "Algorithms of favoritism: Power wins again," *Modern Diplomacy*, February 2025.

[24] "Digital Services Act (DSA) | Updates, Compliance, Training." [Online]. Available: https://www.eu-digital-services-act.com/

[25] BitTorrent Limited. (2025) BitTorrent — The World's Most Popular Torrent Client. [Online]. Available: https://www.bittorrent.com/

[26] S. Josefsson and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)," RFC 8032, Jan. 2017. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8032

[27] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, vol. 2, no. 2, pp. 77–89, 2012.

[28] A. Borodin and R. Moenck, "Fast modular transforms," *Journal of Computer and System Sciences*, vol. 8, no. 3, pp. 366–386, 1974.

[29] R. P. Brent and H. T. Kung, "Fast algorithms for manipulating formal power series," *J. ACM*, vol. 25, no. 4, p. 581–595, Oct. 1978.

[30] Noir Team. Noir documentation. [Online]. Available: https://noir-lang.org/docs/

[31] Aztec Team. Barretenberg documentation. [Online]. Available: https://barretenberg.aztec.network/docs/

[32] "Latency numbers everyone should know," https://static.googleusercontent.com/media/sre.google/en//static/pdf/rule-of-thumb-latency-numbers-letter.pdf.

[33] S. Das, P. Camacho, Z. Xiang, J. Nieto, B. Bünz, and L. Ren, "Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold," in *ACM CCS 2023*, 2023, pp. 356–370.

[34] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *NSDI 17*, 2017, pp. 259–282.

[35] S. Addanki, K. Garbe, E. Jaffe, R. Ostrovsky, and A. Polychroniadou, "Prio+: Privacy preserving aggregate statistics via boolean shares," in *International Conference on Security and Cryptography for Networks*. Springer, 2022, pp. 516–539.

[36] A. Bittau, Ú. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld, "Prochlo: Strong privacy for analytics in the crowd," in *SOSP '17*, 2017, pp. 441–459.

[37] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly) logarithmic overhead," in *ACM CCS '20*, 2020, pp. 1253–1269.

[38] J. Bell, A. Gascón, T. Lepoint, B. Li, S. Meiklejohn, M. Raykova, and C. Yun, "ACORN: input validation for secure aggregation," in *USENIX Security Symposium*, 2023, pp. 4805–4822.

[39] M. Green, W. Ladd, and I. Miers, "A Protocol for Privately Reporting Ad Impressions at Scale," in *ACM CCS '16*. ACM, Oct. 2016, pp. 1591–1601.

[40] O. Acharya, S. Biswas, W. Feng, A. O'Neill, and A. Yerukhimovich, "Non-interactive verifiable aggregation," *Proceedings on Privacy Enhancing Technologies*, vol. 2025, no. 4, pp. 1055–1074, 2025.

[41] J. Blömer, J. Juhnke, and C. Kolb, "Anonymous and Publicly Linkable Reputation Systems," in *Financial Cryptography and Data Security*. Berlin, Heidelberg: Springer, 2015, pp. 478–488.

[42] T. S. Heydt-Benjamin, H.-J. Chae, B. Defend, and K. Fu, "Privacy for public transportation," in *International Workshop on Privacy Enhancing Technologies*. Springer, 2006, pp. 1–19.

[43] A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda, "Privacy pass: Bypassing internet challenges anonymously," *PETS*, 2018.

[44] F. Benhamouda, M. Raykova, and K. Seth, "Anonymous counting tokens," in *Aisacrypt '23*. Springer, 2023, pp. 245–278.

[45] B. Kreuter, T. Lepoint, M. Orrù, and M. Raykova, "Anonymous tokens with private metadata bit," in *CRYPTO 2020*, vol. 12170. Springer, 2020, pp. 308–336.

[46] M. Chase, F. B. Durak, and S. Vaudenay, "Anonymous tokens with stronger metadata bit hiding from algebraic macs," in *Annual International Cryptology Conference*. Springer, 2023, pp. 418–449.

[47] F. Baldimtsi, L. Hanzlik, Q. Nguyen, and A. Yadav, "Non-interactive anonymous tokens with private metadata bit," *Cryptology ePrint Archive*, 2025.

[48] Y. Ji and K. Chalkias, "Generalized proof of liabilities," in *CCS '21*, Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds. ACM, 2021, pp. 3465–3486.

[49] J. Xin, A. Haghighi, X. Tian, and D. Papadopoulos, "Notus: Dynamic proofs of liabilities from zero-knowledge RSA accumulators," in *USENIX Security 24*, Aug. 2024, pp. 1453–1470.

[50] F. Falzon, K. Elkhiyaoui, Y. Manevich, and A. D. Caro, "Short privacy-preserving proofs of liabilities," in *ACM CCS 2023*. ACM, 2023, pp. 1805–1819.

[51] S. Das and L. Ren, "Adaptively Secure BLS Threshold Signatures from DDH and co-CDH," in *Advances in Cryptology – CRYPTO 2024*, L. Reyzin and D. Stebila, Eds. Cham: Springer Nature Switzerland, 2024, pp. 251–284.

[52] L. Baird, S. Garg, A. Jain, P. Mukherjee, R. Sinha, M. Wang, and Y. Zhang, " Threshold Signatures in the Multiverse ," in *IEEE Symposium on Security 2023*. Los Alamitos, CA, USA: IEEE Computer Society, May 2023, pp. 1454–1470.

[53] C. Delerablée and D. Pointcheval, "Dynamic Threshold Public-Key Encryption," in *Advances in Cryptology – CRYPTO 2008*, D. Wagner, Ed. Berlin, Heidelberg: Springer, 2008, pp. 317–334.

[54] S. Garg, D. Kolonelos, G.-V. Policharla, and M. Wang, "Threshold Encryption with Silent Setup," in *CRYPTO 2024*, 2024, pp. 352–386.

[55] R. L. Rivest, A. Shamir, and Y. Tauman, "How to Leak a Secret," in *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, Ed. Berlin, Heidelberg: Springer, 2001, pp. 552–565.

[56] A. Bender, J. Katz, and R. Morselli, "Ring signatures: Stronger definitions, and constructions without random oracles," in *Theory of Cryptography Conference*. Springer, 2006, pp. 60–79.

[57] J. Groth, "On the Size of Pairing-Based Non-interactive Arguments," in *Advances in Cryptology – EUROCRYPT 2016*, M. Fischlin and J.-S. Coron, Eds. Berlin, Heidelberg: Springer, 2016, pp. 305–326.

[58] J. Thaler, "A Note on the GKR Protocol."

[59] Federal Transit Administration. (2016) Shared mobility FAQs: Eligibility under FTA grant programs.

[60] Learn & Work Ecosystem Library. (2018) How does WIOA provide/pay for training?

[61] Connecticut Department of Social Services. (2013) Regulation concerning payment of behavioral health clinic services.

[62] F.-X. Wicht, Z. Tong, S. Zhou, H. Yin, and A. Yaish, "Persistent BitTorrent trackers," Cryptology ePrint Archive, Paper 2025/2131, 2025.

Appendix A. Strawman and Inefficient Solutions

In this section, we go over two related primitives and explain why they fail to solve the VAR problem.

Ring signatures. An alternative approach is to use *ring signatures* [55], [56], which inherently satisfy privacy. However, most practical ring signatures typically incur a cost linear in the size of the anonymity set (the entire user set), as they include the public keys of all users in the set; thus, the resultant spending process would become highly inefficient in terms of both time and bandwidth. Moreover, it is unclear how to efficiently aggregate all these ring signatures at audit time towards a sublinear audit proof and verification time.

Aggregate message authentication codes. Another approach that inherently satisfies deniability is to rely on message authentication code (MAC) schemes. In a MAC-based VAR protocol, $\mathcal{V}$ shares a symmetric MAC key with each user; to spend, a user produces a MAC tag on the session id and sends the tag to $\mathcal{P}$. Since MAC schemes are *symmetric-key* primitives, user spending satisfies deniability. However, MAC cannot be verified by $\mathcal{P}$ (which means a malicious user can corrupt the entire aggregate), and moreover, $\mathcal{V}$ needs to know the user identities to verify MACs, violating privacy.

Appendix B. Threshold Schedules

In this section, we discuss details related to the threshold schedules in our S-VAR construction.

Choice of threshold schedule. The threshold schedule in S-VAR is chosen by $\mathcal{V}$ and can be tuned to the application. An *arithmetic* schedule fixes a step size Δ and sets $t_j = \min\{n, j\Delta\}$ for $j \in [\lceil n/\Delta \rceil]$. This gives additive accuracy: the largest provable threshold is within Δ of the true count, but the number of tiers grows as $\Theta(n/\Delta)$. A *geometric* schedule instead fixes $\varepsilon > 0$ and sets $t_1 = 1$ and $t_{j+1} = \min\{n, \lceil (1+\varepsilon)t_j \rceil\}$. This gives multiplicative accuracy: the largest provable threshold is within a factor $(1+\varepsilon)$ of the true count, using only $O(\log_{1+\varepsilon} n)$ tiers. For example, powers of two correspond to $\varepsilon = 1$ and require about $\log_2 n$ tiers.

Applications may also use a *hybrid* schedule, with arithmetic spacing for small counts and geometric spacing beyond a switching point $n' < n$. This is useful when small counts require fine additive precision, while large counts only need multiplicative precision. For instance, with $n = 10^6$ and $\varepsilon = 0.01$, a pure geometric schedule uses roughly 1,400 tiers; using arithmetic spacing $\Delta = 10$ up to $n' = 10^3$ and geometric spacing afterward reduces this to about 800 tiers.

Toward dynamic thresholds. In S-VAR, thresholds are fixed at issuance. This can be restrictive when engagement is lower than expected: $\mathcal{P}$ cannot meet the next threshold even though a smaller meaningful count was reached. A simple modification gives downward-adjustable thresholds without changing issuance or spending. At audit time, if $\mathcal{P}$ wants to prove threshold t_{j^*} but has only $t_{j^*} - d$ user receipts, it sends d to $\mathcal{V}$. Then $\mathcal{V}$ returns d additional evaluation points of f_{j^*}, disjoint from both user points and the existing auxiliary points. If these points let $\mathcal{P}$ reconstruct s_{j^*}, then $\mathcal{V}$ is convinced that at least $t_{j^*} - d$ users spent. Thus, public auxiliary points can stand in for missing users and let $\mathcal{V}$ relax thresholds at audit time while preserving the soundness and privacy guarantees of S-VAR.

Appendix C. Additional Details on P-VAR

Example construction of nested interpolation domains in §5.2. Here we show one way to construct such interpolation domains in practice. For instance, with BLS12-381, a pairing-friendly elliptic curve, the scalar field is $\mathbb{F}_p$ with p being a 255-bit prime, and $2^{32} \mid p-1$, so a 2^{32}-th primitive root of unity ω exists. We can therefore construct $\Omega^{(1)} \subset \cdots \subset \Omega^{(32)}$ with $N^{(i)} = 2^i$ as follows: $\Omega^{(2)} = \{1, \omega^{2^{31}}\}$, $\Omega^{(3)} = \Omega^{(2)} \cup \{\omega^{2^{30}}, \omega^{3\cdot 2^{30}}\}$, $\Omega^{(4)} = \Omega^{(3)} \cup \{\omega^{2^{29}}, \omega^{3\cdot 2^{29}}, \omega^{5\cdot 2^{29}}, \omega^{7\cdot 2^{29}}\}$, and so on. Each user $\mathcal{U}_i$ is associated with the i-th interpolation point (e.g., user 1 with 1, user 2 with $\omega^{2^{31}}$, and so on.) With this nested subset structure, when the system migrates to a bigger set of users, the existing users do not need to be re-associated with a different interpolation point.

Appendix D. Implementation and Evaluation Details

D.1. Baselines

Here, we describe in more detail the two baselines that we compare our constructions against in §7.

Baseline 1: Predicate aggregate signatures. For the PAS scheme of Qiu et al. [5], since the original paper did not provide an implementation, we implement the core inner-product arguments to obtain a lower bound on the runtime.

Baseline 2: Generic SNARK with aggregate signatures. Our second baseline achieves all security properties (with some weakening) using a digital signature scheme with efficient aggregation and generic SNARK, which $\mathcal{P}$ will use to attest that he has received the claimed number of receipts. We describe this baseline solution in more detail here. Each user $\mathcal{U}_i$ has a signature key pair (sk_i, pk_i); let $\boldsymbol{pk} = (pk_i)_{i\in[n]}$ be the vector of all user public keys. Each receipt is of the form $\mathsf{rx}_i = (\mathsf{sid}, \sigma_i)$, where σ_i is user $\mathcal{U}_i$'s signature on message sid (using sk_i). Note that the issuance process is simply a local signing operation. As users spend, $\mathcal{P}$ maintains a bit vector $\boldsymbol{b} = (b_i)_{i\in[n]}$ where $b_i = 1$ if and only if $\mathcal{U}_i$ has spent. At audit time, $\mathcal{P}$ computes the count $\mathsf{claimed} = \sum_{i\in[n]} b_i$, an aggregate signature $\tilde{\sigma}$ over the received user signatures, the corresponding aggregate public key apk, and finally a zk-SNARK π for the relation

$$\mathcal{R} := \left\{ (x, w) \;\middle|\; \begin{array}{l} x = (\boldsymbol{pk}, \mathsf{claimed}, \mathsf{apk}),\ w = \boldsymbol{b} : \\ b_i \in \{0,1\}\ \forall i \in [n] \wedge \\ \mathsf{apk} = \mathsf{AggPK}\big((pk_i)_{i\in[n];\, b_i=1}\big) \wedge \\ \sum_{i\in[n]} b_i = \mathsf{claimed} \end{array} \right\},$$

where AggPK is the algorithm for public key aggregation. $\mathcal{P}$ sends $(\mathsf{claimed}, \tilde{\sigma}, \pi)$ to $\mathcal{V}$, who verifies π against public

inputs, verifies $\tilde{\sigma}$ against apk, and checks that $\boldsymbol{pk}$ is indeed the vector of all user public keys. We note that apk might not hide the spend set and hurt user privacy; indeed, BLS aggregate public keys are deterministic, so a semi-honest $\mathcal{V}$ can de-anonymize users by brute-forcing apk.

We implement this baseline using Noir [30] `v1.0.0-beta.15`, Aztec's Barretenberg [31] as the proving backend, and the UltraHonk proving system. Our implementation includes the core zk-SNARK proof generation and verification, i.e., we omit the signature aggregation and verification. Thus our benchmarks provide lower bound proving and verification times for this baseline.

Brief background: zk-SNARKs. We use the term zk-SNARK to refer to a zero-knowledge succinct non-interactive argument of knowledge for an NP relation R. Such a system consists of algorithms $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ satisfying completeness, knowledge soundness, zero knowledge, and succinctness: proofs are short and verification is efficient, typically poly-logarithmic in the size of the circuit computing R [57], [58].

D.2. Additional Evaluation for Issuance

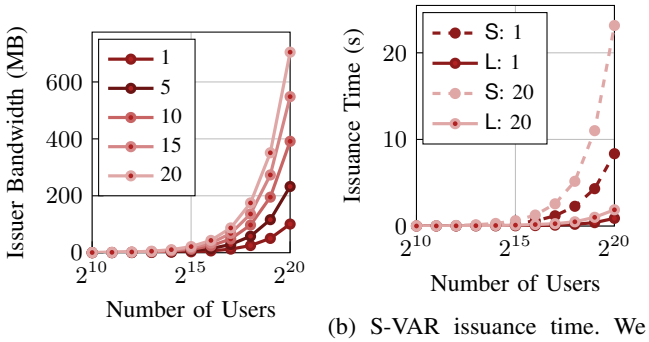
Here, we provide further evaluation for the issuance procedure in P-VAR and S-VAR.

Evaluation methods. For S-VAR, we measure issuer bandwidth and time for a varying number of thresholds $k \in \{1, 5, 10, 15, 20\}$. For issuer bandwidth, our calculations assume a geometric schedule, as described in §B; i.e., given n and k, we calculate $r = n^{1/k}$ and set thresholds as $t_{i+1} = t_i \cdot r$, with $t_k = n$. Note that a geometric schedule incurs a higher bandwidth compared to an arithmetic schedule. For issuer time, we set $t_i = 0$ for all $i \in [k]$, which requires generating the maximum number of public points (n) for each t_i, and thus incurs the maximum size for aux; as such, our benchmarks provide an upper bound on the issuance time incurred by the geometric schedule. For P-VAR, we measure issuer bandwidth and time in the regime where 1 audit is supported.

Evaluation results. Figs. 11 and 12 show the results.

S-VAR. Fig. 11a shows the issuer bandwidth of S-VAR as a function of the number n of users, with a varying number of thresholds $k \in \{1, 5, 10, 15, 20\}$. Concretely, for small scale, such as 2^{10} users, issuance in S-VAR consumes 0.098 MB in bandwidth for 1 threshold, and 0.642 MB for 20 thresholds. For large scale, such as 2^{20} (around one million) users, issuance in S-VAR consumes 100.6 MB in bandwidth for 1 threshold, and 704.6 MB for 20 thresholds. With the chosen geometric schedule, issuer bandwidth in S-VAR grows approximately linearly in n, since the user receipts dominate. Similarly, bandwidth grows approximately *linearly* in the number of thresholds k; for instance, each additional threshold contributes about 32 MB at $n = 2^{20}$.

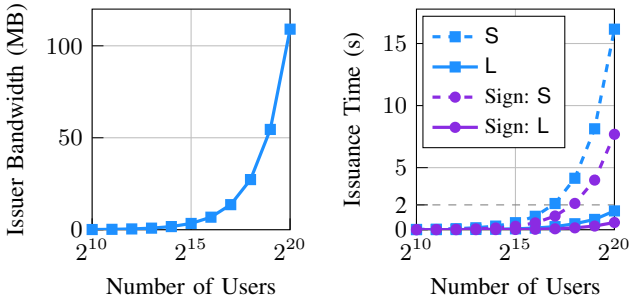
Figure 11b shows the issuance time of S-VAR on large and small servers, in the 1- and 20-threshold regimes (to reduce clutter, we skipped threshold numbers in between). Issuance time in S-VAR scales roughly linearly with n, and also with the number of supported thresholds k. The runtime



(a) S-VAR issuer bandwidth with varying numbers of thresholds.

(b) S-VAR issuance time. We only show 1 and 20 thresholds to reduce clutter.

Figure 11: Issuer bandwidth and time for S-VAR as a function of the number n of users, with a varying number of supported thresholds. L = `c7i.16xlarge`, S = `c7i.xlarge`.



(a) P-VAR issuer bandwidth

(b) P-VAR issuance time.

Figure 12: Issuance bandwidth and time for P-VAR as a function of the number n of users, in the regime where 1 audit is supported. Fig. 12b also shows the signing time, which is part of issuance. L = `c7i.16xlarge`, S = `c7i.xlarge`.

is concretely efficient; in the largest setting we benchmarked – one million users and 20 thresholds – it takes only 23.15 s (S) and 1.876 s (L). We found that EdDSA signing (plotted in Fig. 12b) dominates the issuer runtime on the small VM – about 90% of total issuance time – while on the large VM it contributes only around 50% of the runtime.

P-VAR. Both issuance bandwidth and time in P-VAR scale linearly with n. Fig. 12a shows the issuer bandwidth of P-VAR for one epoch. Concretely, issuance in P-VAR requires bandwidth 0.106 MB for 2^{10} users and 109.05 MB for 2^{20} users. Figure 12b shows the issuance time. For 2^{10} users, issuance takes 0.023 s on S and 0.005 s on L; for 2^{20} users, issuance takes 16.156 s on S and 1.516 s on L. For reference, Fig. 12b shows the time it takes to produce n Ed25519 signatures for different n. Signing contributes around 50% (S) and 33% (L) to the total issuance time.

Appendix E. Applications

E.1. Outcome-based Reimbursement

This class of applications extends the motivating example of company-sponsored benefits in §1 and models scenarios in which a funder (e.g., an employer) reimburses a third-party service provider based on usage by beneficiaries (e.g., employees), a practice we call *outcome-based reimbursement*.

This model captures many public-benefit programs that reimburse independent organizations for services to eligible users. For example, public transit agencies contract with taxi or shared-mobility operators and reimburse them for providing discounted or voucher-based rides to eligible riders (e.g., seniors or riders with disabilities) [59]. Workforce agencies under programs such as WIOA [60] pay approved training providers on a per-participant or per-hour basis for job-training and work-based learning activities. Health and social-services agencies similarly reimburse community behavioral health clinics on a per-session basis for psychotherapy and related services delivered to eligible clients [61].

Mapping to VAR. In each scenario, the issuer chooses a set of eligible users and can run VAR to audit the outcome. The reimbursing entity (e.g., employer, transit authority) acts as the issuer and verifier, $\mathcal{V}$. Each provider (counselor, transit operator) runs a prover $\mathcal{P}$, and eligible beneficiaries form the user set U. During issuance, $\mathcal{V}$ mints service-specific receipts, making them available to each user in U. When users avail a service, they spend their receipts at the corresponding prover $\mathcal{P}$, which adds them to its local VAR instance. At audit time, $\mathcal{V}$ and $\mathcal{P}$ execute the VAR audit procedure: $\mathcal{P}$ proves that the number of receipts it holds matches the requested reimbursement for a given service and time window and $\mathcal{V}$ verifies this claim while learning only the aggregate count.

E.2. Read Receipts for Broadcast Messages

When broadcast channels (e.g., Telegram groups, WeChat groups) are used in sensitive contexts, such as organizing activist events and whistleblowing, a malicious or coerced broadcast platform can disrupt these activities by dropping messages to selected recipients (e.g., users in a specific country or region). In the absence of a verifiable read-receipt mechanism, such suppression can happen silently, with the broadcaster unalerted. VAR enables the sender to gain cryptographic assurance that her message has been delivered, and if not, switch to another platform. Note that privacy and anonymity are especially important here in these contexts: recipients should not be linked to a sensitive message by the receipt, which VAR ensures.

Mapping to VAR. The mapping from the group messaging application to VAR is quite similar to the social media setting. In this case, the message sender acts as the VAR issuer and verifier, the remaining group chat members act as VAR users, and the messaging platform acts as the VAR prover. The performance profile is different, though, as group chats tend to be smaller than content creators' follower sets, and the protocol takes place more frequently.

E.3. Reputation in BitTorrent

Peer-to-peer file-sharing protocols such as BitTorrent [25] enforce minimal upload-to-download ratios to prevent free-riding and align users' incentives. However, the party enforcing such rules does not participate in file transfers and therefore lacks direct visibility into actual downloads and uploads. In practice, they accept self-reported upload ratios, which are trivial to inflate.

A recent work [62] suggests using aggregate signatures to replace self-reported statistics. At a high level, downloaders sign acknowledgements for pieces received from an uploader. The uploader collects signatures from downloaders as proof of their contributions and sends the aggregated signatures to the tracker, who verifies and records the uploads accordingly.

However, using signatures leaks the traffic graph (who exchanged data with whom and when) to the tracker. This leakage is not only unnecessary for the tracker's functionality but also enables various attacks, such as inference of user relationships, identification of high-value participants, and targeted enforcement or censorship. VAR can replace aggregate signatures and avoid leaking the traffic graph, while still allowing succinct proofs.

Mapping to VAR. In this application, the tracker plays the role of the issuer and verifier. When setting up a file, the tracker issues receipts to potential downloaders. When downloading data pieces, a downloader sends the corresponding receipt to the uploader. Later, the uploader aggregates receipts collected from all downloaders and reports the total count of downloads to the tracker.

Appendix F. Security Proofs

F.1. S-VAR: Security Proof

In this section, we prove security of S-VAR. Concretely, S-VAR ensures that a malicious prover cannot convince an honest verifier of an audit count larger than the number of received receipts, and the audit proof does not reveal anything – beyond the count – to a semi-honest verifier.

Theorem 1. *The S-VAR protocol presented in Fig. 3, where* GMSS *is instantiated bottom-up as described in §4.3, realizes the ideal functionality $\mathcal{F}_{\mathsf{VAR}}$, under static corruptions, a semi-honest verifier, and malicious prover and users.*

Proof. Let $\mathcal{A}$ denote the adversary, and $\mathcal{S}$ denote the simulator. If all parties are honest, then $\mathcal{S}$ does not need to simulate anything. On the other hand, if both $\mathcal{P}$ and $\mathcal{V}$ are corrupted, then this case is out of scope per our adversarial model (see §3.2). We thus consider three corruption cases,

disjoint from the two cases above. Let sid be the session identifier and let $\mathcal{V}_{\text{S-VAR}}$ denote the real-world verifier, who follows the S-VAR protocol honestly.

Case 1. $\mathcal{P}$ and $\mathcal{V}$ are honest, but a set $U_{\mathcal{C}}$ of users are corrupted. In this case, $\mathcal{A}$ (controlling $U_{\mathcal{C}}$) is malicious. We construct the simulator $\mathcal{S}$ as follows.

Initialize. First, $\mathcal{S}$ gets the public parameters pp from the environment and gives pp to $\mathcal{A}$. Then $\mathcal{S}$ simulates $\mathcal{A}$ to get the set $U_{\mathcal{C}}$ of corrupted users and sends message $(\mathsf{sid}, \textsc{Init}, U_{\mathcal{C}})$ to $\mathcal{F}_{\mathsf{VAR}}$. Then $\mathcal{S}$ samples $(sk, vk) \leftarrow \mathsf{DS.Gen}(1^\lambda)$. Finally, $\mathcal{S}$ initializes the set of received receipts $\mathrm{R} = \emptyset$.

Issue. Upon receiving $(\mathsf{sid}, \textsc{Issue}, T, e_{\max})$ from $\mathcal{F}_{\mathsf{VAR}}$:

- $\mathcal{S}$ sets $k = |T|$ and $(t_j)_{j\in[k]} := T$. Using the previously sampled sk, $\mathcal{S}$ runs step 1 of the issue procedure to compute secrets $\boldsymbol{s} = (s_j)_{j\in k}$, receipts $\{\mathsf{rx}_i = (\mathsf{sh}_i, \sigma_i)\}_{i\in[n]}$ and auxiliary data aux.
- Sends rx_i to each $\mathcal{U}_i \in U_{\mathcal{C}}$ (i.e., to $\mathcal{A}$).

Spend. $\mathcal{S}$ handles spending from corrupted users. Upon receiving receipt $\mathsf{rx}^* = (\mathsf{sh}^*, \sigma^*)$ from $\mathcal{U}_i \in U_{\mathcal{C}}$ (i.e., from $\mathcal{A}$ on behalf of $\mathcal{U}_i$), first check that $(i, \cdot) \notin \mathrm{R}$ and $(\cdot, \mathsf{rx}^*) \notin \mathrm{R}$. Then check $\mathsf{DS.Verify}(vk, \mathsf{sh}^*, \sigma^*) = 1$. If any of these checks fails, do nothing. If all checks pass, send $(\mathsf{sid}, \textsc{Spend})$ to $\mathcal{F}_{\mathsf{VAR}}$ as if it came from $\mathcal{U}_i$ and update $\mathrm{R} = \mathrm{R} \cup \{(i, \mathsf{rx}^*)\}$.

Audit. $\mathcal{S}$ does nothing during the audit procedure.

We argue indistinguishability between the two worlds.

Initialize. The public parameters pp are given by the environment, so they are distributed identically in the two worlds. Whenever $\mathcal{A}$ initializes in the real world, $\mathcal{S}$ sends an initialize message to $\mathcal{F}_{\mathsf{VAR}}$ in the ideal world which includes the same set of corrupted users $U_{\mathcal{C}}$. Also, the key pair (sk, vk) sampled by $\mathcal{S}$ in the ideal world is distributed identically to the key pair sampled by $\mathcal{V}$ in the real world.

Issue. The issue procedure is triggered by honest $\mathcal{V}$ and thus occurs at the same time in both worlds. In the ideal world, $\mathcal{S}$ uses the same thresholds as $\mathcal{V}_{\text{S-VAR}}$ and then runs $\mathcal{V}_{\text{S-VAR}}$ to produce $(\boldsymbol{s}, \{\mathsf{rx}_i\}_{i\in[n]}, \mathsf{aux})$, which are identically distributed between the two worlds.

Spend. First, we prove statement (1): for all users $\mathcal{U}_i$, $\mathcal{P}$ adds (i, rx^*) to his set R in the real world if and only if $\mathcal{F}_{\mathsf{VAR}}$ adds $\mathcal{U}_i$ to R in the ideal world. If $\mathcal{U}_i$ is *honest*, then he spends (at most) once in both worlds; also, in the real world, when spending, he spends his valid, issued receipt. Thus the statement holds in this case. If $\mathcal{U}_i$ is *corrupted*, during issuance, $\mathcal{U}_i$ receives an identically distributed receipt in both worlds. Also, at spend time, $\mathcal{S}$'s checks in the ideal world are exactly the same as honest $\mathcal{P}$'s checks in the real world: (a) $(i, \cdot) \notin \mathrm{R}$, (b) $(\cdot, \mathsf{rx}^*) \notin \mathrm{R}$, and (c) the signature in rx^* verifies. Thus the statement holds in this case, too.

Second, we prove statement (2): in both worlds, if $\mathcal{U}_i$ spends a receipt rx^* and is added to the spend set, then rx^* is among the issued receipts. It suffices to prove the statement for the real world, since $\mathcal{S}$ performs the same checks in the ideal world as honest $\mathcal{P}$ does in the real world. If $\mathcal{U}_i$ is honest, the statement holds since honest users only spend their issued receipt. If $\mathcal{U}_i$ is corrupted, then assume for contradiction that $\mathsf{rx}^* = (\mathsf{sh}^*, \sigma^*)$ is not equal to the receipt $\mathsf{rx}_i = (\mathsf{sh}_i, \sigma_i)$ issued by $\mathcal{V}$ to $\mathcal{U}_i$. Then we have $\sigma^* \neq \sigma$ or $\mathsf{sh}^* \neq \mathsf{sh}_i$; in either case, $\mathcal{A}$ is able to break unforgeability of the DS scheme, which is a contradiction.

Audit. Consider one audit run, and let $\mathsf{claimed} = t_{j^*}$ be the claimed count in the real world. Let s_{j^*} be the secret sampled by honest $\mathcal{V}$ for threshold t_{j^*}. Given statement (2), honest $\mathcal{P}$ uses exactly the shares issued by honest $\mathcal{V}$ and is able to reconstruct the original secret s_{j^*}; therefore, $\mathcal{V}$ outputs $(\mathsf{accept}, \mathsf{claimed})$. In the ideal world, given statement (1), the count proven by $\mathcal{P}$ is also claimed (i.e., the same as in the real world). Thus $\mathcal{F}_{\mathsf{VAR}}$ sends $(\mathsf{sid}, \textsc{AuditOK}, \mathsf{claimed})$ to $\mathcal{V}$, who outputs $(\mathsf{accept}, \mathsf{claimed})$.

Case 2. $\mathcal{V}$ is honest, but $\mathcal{P}$ and a set $U_{\mathcal{C}}$ of users are corrupted. In this case, $\mathcal{A}$ (controlling $\mathcal{P}$ and $U_{\mathcal{C}}$) is malicious. We construct the simulator $\mathcal{S}$ as follows.

Initialize. First, $\mathcal{S}$ gets the public parameters pp from the environment and gives pp to $\mathcal{A}$. Then $\mathcal{S}$ simulates $\mathcal{A}$ to get the set $U_{\mathcal{C}}$ of corrupted users and sends message $(\mathsf{sid}, \textsc{Init}, U_{\mathcal{C}})$ to $\mathcal{F}_{\mathsf{VAR}}$. Then $\mathcal{S}$ samples $(sk, vk) \leftarrow \mathsf{DS.Gen}(1^\lambda)$ and sends vk to $\mathcal{A}$, as if it came from $\mathcal{V}$. Finally, $\mathcal{S}$ initializes the spend sets $R = \emptyset$ and $U'_{\mathcal{C}} = \emptyset$.

Issue. Upon receiving $(\mathsf{sid}, \textsc{Issue}, T, e_{\max})$ from $\mathcal{F}_{\mathsf{VAR}}$:

- $\mathcal{S}$ sets $k = |T|$ and $\boldsymbol{t} = (t_j)_{j\in k} = T$. Using the previously sampled sk, $\mathcal{S}$ runs step 1 of the issue procedure in the real world to compute secrets $\boldsymbol{s} = (s_j)_{j\in k}$, receipts $\{\mathsf{rx}_i = (\mathsf{sh}_i, \sigma_i)\}_{i\in[n]}$ and auxiliary data aux.
- Sends $(\boldsymbol{t}, \mathsf{aux})$ to $\mathcal{A}$.
- Sends rx_i to each $\mathcal{U}_i \in U_{\mathcal{C}}$ (i.e., to $\mathcal{A}$).

Spend. Upon receiving $(\mathsf{sid}, \textsc{Spend}, \mathcal{U}_i)$ from $\mathcal{F}_{\mathsf{VAR}}$ where $\mathcal{U}_i \notin U_{\mathcal{C}}$, $\mathcal{S}$ sends rx_i to $\mathcal{A}$ as if it came from $\mathcal{U}_i$. Then $\mathcal{S}$ updates $R = R \cup \{\mathcal{U}_i\}$.

Audit. Upon receiving $(\mathsf{sid}, \textsc{AuditReq})$ from $\mathcal{F}_{\mathsf{VAR}}$, $\mathcal{S}$ sets $\mathsf{spent} = |R|$ and sends an audit request to $\mathcal{A}$ as if it came from $\mathcal{V}$. Then $\mathcal{S}$ simulates the audit procedure between $\mathcal{V}_{\text{S-VAR}}$ and $\mathcal{A}$. Let $\mathsf{claimed} = t_{j^*}$ be the threshold value output by $\mathcal{A}$ after step 2 of the audit procedure (see Fig. 3). We consider two cases:

- Case (A): $\mathcal{V}_{\text{S-VAR}}$ outputs $(\mathsf{accept}, \mathsf{claimed})$. If $\mathsf{claimed} > \mathsf{spent}$, $\mathcal{S}$ chooses an *arbitrary* set $U''_{\mathcal{C}} \subseteq U_{\mathcal{C}}$ such that $U''_{\mathcal{C}} \cap U'_{\mathcal{C}} = \emptyset$ and $|U''_{\mathcal{C}}| = \min\{|U_{\mathcal{C}} \setminus U'_{\mathcal{C}}|, \mathsf{claimed} - \mathsf{spent}\}$. Then $\mathcal{S}$ updates $U'_{\mathcal{C}} = U'_{\mathcal{C}} \cup U''_{\mathcal{C}}$, $R' = R \cup U''_{\mathcal{C}}$, and sends $(\mathsf{sid}, \textsc{SpendExtra}, U''_{\mathcal{C}})$ to $\mathcal{F}_{\mathsf{VAR}}$. If $\mathsf{claimed} \leq \mathsf{spent}$, $\mathcal{S}$ does not simulate any extra spending. In either case, $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{AuditResp}, 1, \mathsf{claimed})$ to $\mathcal{F}_{\mathsf{VAR}}$.
- Case (B): $\mathcal{V}_{\text{S-VAR}}$ outputs (reject). In this case, $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{AuditResp}, 0, \mathsf{claimed})$ to $\mathcal{F}_{\mathsf{VAR}}$.

Clarifying notation. We write R and $\mathsf{spent} = |R|$ to denote the spend set and its cardinality *before* $\mathcal{S}$ (potentially) invokes $\textsc{SpendExtra}$; we write R' and $\mathsf{spent}' = |R'|$ denote the spend set and its cardinality *after* $\mathcal{S}$ (potentially) invokes $\textsc{SpendExtra}$.

We argue indistinguishability between the two worlds.

Initialize. The public parameters pp are given by the environment, so they are distributed identically in the two

worlds. Whenever $\mathcal{A}$ initializes in the real world, $\mathcal{S}$ sends an initialize message to $\mathcal{F}_{\mathsf{VAR}}$ in the ideal world which includes the same set of corrupted users $U_{\mathcal{C}}$. Also, the verification key vk sampled by $\mathcal{S}$ and sent to $\mathcal{A}$ is distributed identically to the key in the real world.

Issue. The issue procedure occurs at the same time in both worlds, since it is triggered by the honest $\mathcal{V}$. In the ideal world, $\mathcal{S}$ uses the same thresholds as $\mathcal{V}_{\text{S-VAR}}$ and then runs $\mathcal{V}_{\text{S-VAR}}$ to produce $(\boldsymbol{s}, \{\mathsf{rx}_i\}_{i\in[n]}, \mathsf{aux})$. Thus all secrets, receipts and auxiliary data are identically distributed between the two worlds, from the point of view of $\mathcal{P}$ and $U_{\mathcal{C}}$.

Spend. Whenever an honest user spends in the real world, $\mathcal{S}$ simulates this spending in the ideal world. Concretely, in the ideal world, when $\mathcal{F}_{\mathsf{VAR}}$ notifies $\mathcal{S}$ of an honest user $\mathcal{U}_i$ spending, $\mathcal{S}$ sends rx_i to $\mathcal{A}$ which is identically distributed as the receipt he would have received in the real world. Note that at this point, $\mathcal{S}$ does not simulate any corrupted user spending; looking ahead, $\mathcal{S}$ will instead send extra spend messages to $\mathcal{F}_{\mathsf{VAR}}$ as needed during the audit simulation.

Audit. We prove that $\mathcal{V}$ outputs $(\mathsf{accept}, \mathsf{claimed})$ in the ideal world if and only if $\mathcal{V}_{\text{S-VAR}}$ outputs $(\mathsf{accept}, \mathsf{claimed})$ during $\mathcal{S}$'s simulation of the audit procedure. For the "if" direction, assume that $\mathcal{V}_{\text{S-VAR}}$ outputs (reject) at the end of $\mathcal{S}$'s simulation of the audit procedure between $\mathcal{V}_{\text{S-VAR}}$ and $\mathcal{A}$, on the claimed count $\mathsf{claimed}$. Then $\mathcal{S}$ enters case (B), sends $(\mathsf{sid}, \textsc{AuditResp}, 0, \mathsf{claimed})$ to $\mathcal{F}_{\mathsf{VAR}}$, which in turn sends $(\mathsf{sid}, \textsc{AuditBad})$ to $\mathcal{V}$. Thus $\mathcal{V}$ outputs (reject).

For the "only if" direction, assume $\mathcal{V}_{\text{S-VAR}}$ outputs $(\mathsf{accept}, \mathsf{claimed})$. For the sake of contradiction, assume $\mathcal{V}$ outputs (reject) in the ideal world. This means that $\mathcal{F}_{\mathsf{VAR}}$ sends $(\mathsf{sid}, \textsc{AuditBad})$ to $\mathcal{V}$, which happens if at least one of (I) or (II) below has occured:

- (I): $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{AuditResp}, 0, \mathsf{claimed})$ to $\mathcal{F}_{\mathsf{VAR}}$.
- (II): $\mathcal{F}_{\mathsf{VAR}}$'s check that $\mathsf{claimed} \in T$ and $\mathsf{claimed} \leq \mathsf{spent}'$ *fails*. Recall that spent' is the cardinality of the spend set (R') *after* potential invocations to SpendExtra.

Note that (I) cannot occur, since by case (A) of $\mathcal{S}$'s audit simulation, $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{AuditResp}, 1, \mathsf{claimed})$ whenever $\mathcal{V}_{\text{S-VAR}}$ outputs $(\mathsf{accept}, \mathsf{claimed})$. Thus (II) occurred; since $\mathcal{V}_{\text{S-VAR}}$ already checks that $\mathsf{claimed} \in T$, we must have

$$\mathsf{claimed} > \mathsf{spent}'.$$

Recall that R and $\mathsf{spent} = |R|$ denote the spend set and its cardinality *before* $\mathcal{S}$ (potentially) invokes SpendExtra. Note that $\mathsf{spent}' = \mathsf{spent} + |U''_{\mathcal{C}}|$; this holds even if SpendExtra is not invoked, since in that case we can set $|U''_{\mathcal{C}}| = 0$. Since $\mathcal{V}_{\text{S-VAR}}$ outputs $(\mathsf{accept}, \mathsf{claimed})$, we are in case (A) of of the audit simulation, and in particular, in the subcase where $\mathsf{claimed} > \mathsf{spent}$ (if not, then $\mathcal{S}$ does not simulate any extra spending, and $\mathsf{spent}' = \mathsf{spent}$, which would mean $\mathsf{claimed} \leq \mathsf{spent}'$). Recall that $\mathcal{S}$ constructs $U''_{\mathcal{C}}$ such that $|U''_{\mathcal{C}}| = \min\{|U_{\mathcal{C}} \setminus U'_{\mathcal{C}}|, \mathsf{claimed} - \mathsf{spent}\}$. We consider two cases:

1) $|U''_{\mathcal{C}}| = \mathsf{claimed} - \mathsf{spent}$. In this case, we have $\mathsf{spent}' = \mathsf{spent} + |U''_{\mathcal{C}}| = \mathsf{claimed}$, which contradicts the assumption that $\mathsf{claimed} > \mathsf{spent}'$.
2) $|U''_{\mathcal{C}}| = |U_{\mathcal{C}} \setminus U'_{\mathcal{C}}|$.

In case (2), since $U''_{\mathcal{C}} \cap U'_{\mathcal{C}} = \emptyset$, we have $|U_{\mathcal{C}}| = |U'_{\mathcal{C}}| + |U''_{\mathcal{C}}|$. Then we can express

$$\begin{aligned}\mathsf{spent}' = |R'| &= |R| + |U''_{\mathcal{C}}| \\ &= |R \setminus U'_{\mathcal{C}}| + |U'_{\mathcal{C}}| + |U''_{\mathcal{C}}| \\ &= |U'_{\mathcal{H}}| + |U_{\mathcal{C}}|\,,\end{aligned}$$

where $U'_{\mathcal{H}} := \{\mathcal{U} \in R \mid \mathcal{U} \notin U_{\mathcal{C}}\}$ is the set of honest users who have spent. Since $\mathsf{claimed} > \mathsf{spent}'$, we have $\mathsf{claimed} > |U'_{\mathcal{H}}| + |U_{\mathcal{C}}|$. Since $\mathcal{A}$ knows at most $|U_{\mathcal{C}}|$ shares (from corrupted users) on top of the $|U'_{\mathcal{H}}|$ shares from honest users, and since $\mathcal{V}_{\text{S-VAR}}$ outputs $(\mathsf{accept}, \mathsf{claimed})$, $\mathcal{A}$ must be able to reconstruct the secret s_{j^*} for threshold $t_{j^*} = \mathsf{claimed}$ with less than t_{j^*} shares. Thus $\mathcal{A}$ must be able to either derive the missing shares, which violates the security of the hash function in the random oracle model, or recover the n-degree polynomial f_j with less than $(n - t_{j^*}) + t_{j^*} = n$ evaluation points, which is information-theoretically impossible.

Case 3. $\mathcal{P}$ is honest, but $\mathcal{V}$ and a set $U_{\mathcal{C}}$ of users are corrupted. Recall that our threat model assumes $\mathcal{V}$ is semi-honest, thus, in this case, $\mathcal{A}$ (controlling $\mathcal{V}$ and $U_{\mathcal{C}}$) is semi-honest. We construct the simulator $\mathcal{S}$ as follows.

Initialize. First, $\mathcal{S}$ gets the public parameters pp and gives pp to $\mathcal{A}$. Then $\mathcal{S}$ simulates $\mathcal{A}$ to get the set $U_{\mathcal{C}}$ of corrupted users and the verification key vk meant for $\mathcal{P}$. $\mathcal{S}$ stores vk and sends message $(\mathsf{sid}, \textsc{Init}, U_{\mathcal{C}})$ to $\mathcal{F}_{\mathsf{VAR}}$.

Issue. $\mathcal{S}$ simulates $\mathcal{A}$ to get the issued receipts $\{\mathsf{rx}_i = (\mathsf{sh}_i, \sigma_i)\}_{i\in[n]}$, the allowed counts $\boldsymbol{t} = (t_j)_{j\in[k]}$. Then $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{Issue}, \boldsymbol{t}, e_{\max})$ to $\mathcal{F}_{\mathsf{VAR}}$, as if it came from $\mathcal{V}$. $\mathcal{S}$ stores the receipts $\{\mathsf{rx}_i\}_{i\in[n]}$. Note that since $\mathcal{A}$ controls $\mathcal{V}$ and $U_{\mathcal{C}}$, he can implicitly share rx_i (and other receipts he knows) with corrupted user $\mathcal{U}_i$.

Spend. Upon receiving rx_i from $\mathcal{U}_i \in U_{\mathcal{C}}$, store rx_i and send $(\mathsf{sid}, \textsc{Spend})$ to $\mathcal{F}_{\mathsf{VAR}}$ as if it came from $\mathcal{U}_i$.

Audit. $\mathcal{S}$ simulates $\mathcal{A}$ and upon an audit request, $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{Audit})$ to $\mathcal{F}_{\mathsf{VAR}}$. Upon receiving $(\mathsf{sid}, \textsc{AuditOK}, \mathsf{claimed})$ from $\mathcal{F}_{\mathsf{VAR}}$, $\mathcal{S}$ chooses an arbitrary subset $I \subseteq [n]$ of size $|I| = \mathsf{claimed}$. Then $\mathcal{S}$ runs audit step 2 in Fig. 3 with $\mathrm{R} = \{\mathsf{sh}_i\}_{i\in I}$ to compute (j^*, t_{j^*}, s'), which it sends to $\mathcal{A}$ as if it came from $\mathcal{P}$.

We argue indistinguishability between the two worlds.

Initialize. The public parameters pp are given by the environment, so they are distributed identically in the two worlds. Whenever $\mathcal{A}$ initializes in the real world, $\mathcal{S}$ sends an initialize message to $\mathcal{F}_{\mathsf{VAR}}$ in the ideal world which includes the same set of corrupted users $U_{\mathcal{C}}$. Also, the key pair (sk, vk) sampled by $\mathcal{S}$ in the ideal world is identically distributed as the key pair sampled by $\mathcal{V}$ in the real world.

Issue. Whenever $\mathcal{A}$ issues in the real world, $\mathcal{F}_{\mathsf{VAR}}$ sends an Issue message to the prover and all users in the ideal world. In the real world, before the issue procedure, an honest user may not spend. After the issue procedure, if the environment triggers an honest user $\mathcal{U}$ to spend, she may do so once. The simulator causes the exact same behavior in the ideal world by triggering issuance when $\mathcal{A}$ issues. Since $\mathcal{A}$ is semi-honest

in this case, $\mathcal{S}$ does not need to check the well-formedness of the receipts issued by $\mathcal{A}$.

Spend. Whenever $\mathcal{A}$ spends on behalf of a corrupted user in the real world, $\mathcal{S}$ simulates this spending in the ideal world. Again, since $\mathcal{A}$ is semi-honest, $\mathcal{S}$ does not need to check the well-formedness of the receipts spent by corrupted users.

Audit. In the ideal world, since $\mathcal{P}$ is honest, we know that $\mathcal{F}_{\mathsf{VAR}}$ will only ever send a message $(\mathsf{sid}, \textsc{AuditOK}, \mathsf{claimed})$ to $\mathcal{V}$ (i.e., no bad audits). We also know $\mathsf{claimed} = t_{j^*}$ where t_{j^*} is the threshold proven by the honest $\mathcal{P}$ in the real world. In the real world, $\mathcal{A}$ performs step 1 of the issue procedure (in Fig. 3) honestly. Thus, any reconstruction subset I' of size t_{j^*} can be used to reconstruct the original secret s_{j^*}, which is precisely what $\mathcal{S}$ does without knowing the spend set. Also, all secrets and shares computed by $\mathcal{S}$ at issuance in the ideal world are distributed *identically* to the ones computed by $\mathcal{A}$ in the real world; the thresholds are in fact *equal* between the two worlds. Thus, the tuple (j^*, t_{j^*}, s') computed by $\mathcal{S}$ and received by $\mathcal{A}$ in the ideal world audit is distributed *identically* to the tuple received by $\mathcal{A}$ in the real world audit. □

F.2. P-VAR: Security Proof

In this section, we prove security of P-VAR. Concretely, P-VAR ensures that a malicious prover cannot convince an honest verifier of an audit count larger than the number of received receipts, and that the audit proof does not reveal anything beyond the count to a semi-honest verifier.

Theorem 2. *The P-VAR protocol presented in Figs. 4 and 5 realizes the ideal functionality $\mathcal{F}_{\mathsf{VAR}}$ under static corruptions, assuming a semi-honest verifier, malicious prover and users, PRF security, EUF-CMA security of* DS*, KZG/PCS soundness for the quotient checks, and knowledge soundness and zero-knowledge of the Fiat-Shamir structure proof in the random oracle model.*

Proof. Let $\mathcal{A}$ denote the adversary and $\mathcal{S}$ denote the simulator. If all parties are honest, the real and ideal executions are identical by correctness of the protocol. If both $\mathcal{P}$ and $\mathcal{V}$ are corrupted, the case is out of scope per our adversarial model (see §3.2). We therefore consider the remaining three cases. Let sid be the session identifier. We write k_e for the count proved in epoch e, and K for the cumulative count accepted so far.

Case 1. $\mathcal{P}$ and $\mathcal{V}$ are honest, but a set $U_{\mathcal{C}}$ of users are corrupted. In this case, $\mathcal{A}$ controls $U_{\mathcal{C}}$ maliciously. We construct $\mathcal{S}$ as follows.

Initialize. $\mathcal{S}$ receives the public parameters pp from the environment and gives pp to $\mathcal{A}$. It obtains the corrupted set $U_{\mathcal{C}}$ and sends $(\mathsf{sid}, \textsc{Init}, U_{\mathcal{C}})$ to $\mathcal{F}_{\mathsf{VAR}}$. It samples $(sk, vk) \leftarrow \mathsf{DS.Gen}(1^\lambda)$ and gives vk to $\mathcal{A}$. It initializes $U'_{\mathcal{C}} := \emptyset$.

Issue. Upon $(\mathsf{sid}, \textsc{Issue}, T, e_{\max})$ from $\mathcal{F}_{\mathsf{VAR}}$, $\mathcal{S}$ samples seeds $\mathsf{seed}_1, \ldots, \mathsf{seed}_{e_{\max}}$. For each epoch $e \in [e_{\max}]$ and each corrupted user $\mathcal{U}_i \in U_{\mathcal{C}}$, $\mathcal{S}$ computes $r_{i,e} := \mathsf{PRF}(\mathsf{seed}_e; i)$ and $\sigma_{i,e} := \mathsf{DS.Sign}(sk, i \| r_{i,e} \| e)$, and gives $\mathsf{rx}_{i,e} := (i, r_{i,e}, e, \sigma_{i,e})$ to $\mathcal{A}$.

Spend. Whenever $\mathcal{A}$ instructs a corrupted user $\mathcal{U}_i$ to spend a receipt $\mathsf{rx}_{i,e}$, $\mathcal{S}$ verifies the signature. If it verifies and $\mathcal{U}_i \notin U'_{\mathcal{C}}$, then $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{Spend}, \mathcal{U}_i)$ to $\mathcal{F}_{\mathsf{VAR}}$ and updates $U'_{\mathcal{C}} := U'_{\mathcal{C}} \cup \{\mathcal{U}_i\}$. Duplicate spends are ignored, matching the real protocol and the ideal functionality.

Audit. Since both $\mathcal{P}$ and $\mathcal{V}$ are honest, $\mathcal{S}$ does not need to simulate adversarial audit messages.

We now compare the two executions. By EUF-CMA security of DS, a corrupted user can only produce a verifying receipt that was honestly issued to that user for some epoch, except with negligible probability. Hence the set of corrupted users whose receipts are accepted by the honest prover is exactly the set reported by $\mathcal{S}$ to $\mathcal{F}_{\mathsf{VAR}}$. Honest parties run the same deterministic audit algorithm in both worlds, and by PRF security the issued receipt values are indistinguishable from the real execution. Thus the view of $\mathcal{A}$ and the honest outputs are indistinguishable in the two worlds.

Case 2. $\mathcal{V}$ is honest, but $\mathcal{P}$ and a set $U_{\mathcal{C}}$ of users is corrupted. In this case, $\mathcal{A}$ controls $\mathcal{P}$ and $U_{\mathcal{C}}$ maliciously. We construct $\mathcal{S}$ as follows.

Initialize. $\mathcal{S}$ receives pp from the environment and gives it to $\mathcal{A}$. It obtains $U_{\mathcal{C}}$ and sends $(\mathsf{sid}, \textsc{Init}, U_{\mathcal{C}})$ to $\mathcal{F}_{\mathsf{VAR}}$. It samples $(sk, vk) \leftarrow \mathsf{DS.Gen}(1^\lambda)$ and gives vk to $\mathcal{A}$. It initializes $U'_{\mathcal{H}} := \emptyset$, $U'_{\mathcal{C}} := \emptyset$, local epoch counter $e := 1$, and cumulative count $K := 0$.

Issue. Upon $(\mathsf{sid}, \textsc{Issue}, T, e_{\max})$ from $\mathcal{F}_{\mathsf{VAR}}$, $\mathcal{S}$ samples seeds $\mathsf{seed}_1, \ldots, \mathsf{seed}_{e_{\max}}$. For each epoch $e \in [e_{\max}]$ and each corrupted user $\mathcal{U}_i \in U_{\mathcal{C}}$, $\mathcal{S}$ computes $r_{i,e} := \mathsf{PRF}(\mathsf{seed}_e; i)$ and $\sigma_{i,e} := \mathsf{DS.Sign}(sk, i \| r_{i,e} \| e)$, and gives $\mathsf{rx}_{i,e} := (i, r_{i,e}, e, \sigma_{i,e})$ to $\mathcal{A}$.

Spend. Upon receiving $(\mathsf{sid}, \textsc{Spend}, \mathcal{U}_i)$ from $\mathcal{F}_{\mathsf{VAR}}$ for an honest user $\mathcal{U}_i \notin U_{\mathcal{C}}$, $\mathcal{S}$ computes $r_{i,e} := \mathsf{PRF}(\mathsf{seed}_e; i)$ and $\sigma_{i,e} := \mathsf{DS.Sign}(sk, i \| r_{i,e} \| e)$, and sends $\mathsf{rx}_{i,e} := (i, r_{i,e}, e, \sigma_{i,e})$ to $\mathcal{A}$ as if it came from $\mathcal{U}_i$. It updates $U'_{\mathcal{H}} := U'_{\mathcal{H}} \cup \{\mathcal{U}_i\}$.

If $\mathcal{A}$ makes a corrupted user spend a valid receipt and that user is not already in $U'_{\mathcal{C}}$, $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{Spend}, \mathcal{U}_i)$ to $\mathcal{F}_{\mathsf{VAR}}$ and updates $U'_{\mathcal{C}} := U'_{\mathcal{C}} \cup \{\mathcal{U}_i\}$.

Audit. Upon receiving $(\mathsf{sid}, \textsc{AuditReq}, \mathcal{V})$ from $\mathcal{F}_{\mathsf{VAR}}$, $\mathcal{S}$ sends an audit request to $\mathcal{A}$ as if it came from $\mathcal{V}$. It then runs the real audit verification algorithm honestly using the stored seed seed_e.

If $e > e_{\max}$ or any step of the the audit verification fails, $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{AuditResp}, 0, 0)$ to $\mathcal{F}_{\mathsf{VAR}}$, increments e, and continues.

If the audit verification accepts an epoch count k_e, let $K^\star := K + k_e$. If $K^\star \notin T$, then $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{AuditResp}, 0, K^\star)$ to $\mathcal{F}_{\mathsf{VAR}}$, increments e, and continues. Otherwise, let $\mathsf{spent}^\star := |U'_{\mathcal{H}}| + |U'_{\mathcal{C}}|$. If $K^\star > |\mathsf{spent}^\star|$, $\mathcal{S}$ tries to choose a set $U''_{\mathcal{C}} \subseteq U_{\mathcal{C}} \setminus U'_{\mathcal{C}}$ of size $K^\star - \mathsf{spent}^\star$. If no such set exists, $\mathcal{S}$ sets $\mathsf{Bad} := 1$, sends $(\mathsf{sid}, \textsc{AuditResp}, 0, K^\star)$ to $\mathcal{F}_{\mathsf{VAR}}$, increments e, and continues. If such a set exists, $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{SpendExtra}, U''_{\mathcal{C}})$ to $\mathcal{F}_{\mathsf{VAR}}$ and updates $U'_{\mathcal{C}} := U'_{\mathcal{C}} \cup U''_{\mathcal{C}}$. Then $\mathcal{S}$ sends

$(\mathsf{sid}, \textsc{AuditResp}, 1, K^\star)$ to $\mathcal{F}_{\mathsf{VAR}}$, sets $K := K^\star$, increments e, and continues.

We now argue that the real and ideal executions are indistinguishable. During Issue and Spend, the receipts shown to $\mathcal{A}$ are distributed exactly as in the real protocol, by construction and PRF security. Since $\mathcal{S}$ internally runs a copy of the P-VAR verifier from the real world, and only relays its messages to $\mathcal{A}$, the view of $\mathcal{A}$ is also identically distributed in the real and ideal world. Now we claim that for a given execution of $\mathcal{A}$, the output of $\mathcal{V}$ at the end of Audit is the same in the real and ideal worlds.

We claim that the ideal world $\mathcal{V}$ outputs $(\mathsf{sid}, \mathsf{accept}, \mathsf{claimed})$ iff the real-world verifier outputs $(\mathsf{sid}, \mathsf{accept}, \mathsf{claimed}_{\text{real}})$.

As in Case 2 of the proof for Theorem 1, we argue that (1) if the real world verifier outputs $(\mathsf{sid}, \mathsf{accept}, \mathsf{claimed}_{\text{real}})$ and ideal world verifier outputs $(\mathsf{sid}, \mathsf{accept}, \mathsf{claimed}_{\text{ideal}})$, then $\mathsf{claimed}_{\text{real}} = \mathsf{claimed}_{\text{ideal}}$ must be equal, since $\mathcal{S}$ only sends a message of the form $(\mathsf{sid}, \textsc{AuditResp}, 1, \mathsf{claimed}_{\text{ideal}})$ if its copy of the real-world verifier outputs $(\mathsf{sid}, \mathsf{accept}, \mathsf{claimed}_{\text{ideal}})$. (2) The only way for the real and ideal world to diverge is if in one world the verifier outputs $(\mathsf{sid}, \mathsf{accept}, \cdot)$ and the other (reject). In the "if" direction, we again note that if the real-world verifier outputs (reject), then, $\mathcal{S}$, internally running the honest verifier code must also have its audit fail and send $(\mathsf{sid}, \textsc{AuditResp}, 0, 0)$ to $\mathcal{F}_{\mathsf{VAR}}$, triggering $\mathcal{F}_{\mathsf{VAR}}$ to send $(\mathsf{sid}, \textsc{AuditBad})$ to the ideal-world $\mathcal{V}$, who outputs (reject).

Next, we consider the "only if" case suppose the real-world verifier outputs $(\mathsf{accept}, \mathsf{claimed})$ and the ideal-world verifier outputs (reject). By the same reasoning as in Theorem 1 (Case 2), this means $\mathsf{claimed} > \mathsf{spent}$, where $\mathsf{spent} = |R|$, the number of users who have spent in the ideal functionality. We now show how the probability of this scenario is negligible, based on the fact that the real-world verifier checks the conditions **C1-C4** and **D** as outlined in Fig. 5.

Assume the real verifier accepts an audit in epoch e. By KZG/PCS soundness and knowledge soundness of the structure proof, except with negligible probability, the accepted proof implies the following polynomial relations.

We wish to show that if the proof accepts in the real world, this is equivalent to $\mathcal{A}$ knowing a total of at least $\mathsf{claimed}$ number of receipts.

This implies that $\mathcal{A}$ can construct a polynomial $\hat{f}_e(x)$ which is the result of interpolating the following vector over Ω:

$$\mathbf{v}[i] = \begin{cases} r_i & \text{if } \mathcal{P} \text{ knows } r_i \\ 0 & \text{otherwise.} \end{cases}$$

First we show that a valid structure proof (C3), implies that the commitment $\mathsf{cm}_{b_e,1}$ is indeed a commitment to a polynomial of the following form: $\hat{b}(x) = b_e(x) + a_e x Z(x)$, where $b_e(x)$ has degree at most $n-1$, and a_e is a constant. To this end, we first spell out the knowledge extraction for the structure proof. Let $C := \mathsf{cm}_{b_e,1}$ and $\mathsf{u}_Z := g_1^{\tau Z(\tau)}$. The structure proof is a Fiat-Shamir proof of knowledge of a scalar a_e and a degree-$(n-1)$ polynomial $B_e(x)$ such that $C = g_1^{B_e(\tau)} \cdot \mathsf{u}_Z^{a_e}$. In the corresponding interactive protocol, the prover first sends T, then receives a random challenge c, and responds with $(a^\star, B^\star, H^\star)$ where $B^\star = g_1^{B^\star_{\text{poly}}(\tau)}$ and $H^\star = h^{B^\star_{\text{poly}}(\tau)}$ for some degree-$(n-1)$ response polynomial $B^\star_{\text{poly}}(x)$. The verifier checks $B^\star \cdot \mathsf{u}_Z^{a^\star} = T \cdot C^c$ and $e(B^\star, h) = e(g_1, H^\star)$.

By the forking lemma in the random oracle model, if a malicious prover produces an accepting Fiat-Shamir proof with non-negligible probability, then an extractor can rewind it to the same first message T and obtain two accepting transcripts with distinct challenges $c \neq c'$. Let their responses be $(a^\star, B^\star, H^\star)$ and $(a^{\star\prime}, B^{\star\prime}, H^{\star\prime})$. By the low-degree knowledge soundness of the cross-group certificate, the extractor obtains degree-$(n-1)$ polynomials $B^\star_{\text{poly}}(x)$ and $B^{\star\prime}_{\text{poly}}(x)$ such that $B^\star = g_1^{B^\star_{\text{poly}}(\tau)}$, $H^\star = h^{B^\star_{\text{poly}}(\tau)}$, $B^{\star\prime} = g_1^{B^{\star\prime}_{\text{poly}}(\tau)}$, and $H^{\star\prime} = h^{B^{\star\prime}_{\text{poly}}(\tau)}$. Dividing the two accepting equations gives $g_1^{B^\star_{\text{poly}}(\tau) - B^{\star\prime}_{\text{poly}}(\tau)} \cdot \mathsf{u}_Z^{a^\star - a^{\star\prime}} = C^{c-c'}$. Therefore, defining $a_e := (a^\star - a^{\star\prime})/(c - c')$ and $B_e(x) := (B^\star_{\text{poly}}(x) - B^{\star\prime}_{\text{poly}}(x))/(c - c')$, we obtain $\deg(B_e) \leq n-1$ and $C = g_1^{B_e(\tau)} \cdot \mathsf{u}_Z^{a_e}$. Thus the accepted structure proof implies that the committed polynomial has the form $\widehat{B}_e(x) := B_e(x) + a_e x Z(x)$. Since $xZ(x)$ vanishes on Ω, the evaluations of $\widehat{B}_e$ on Ω equal those of B_e.

Bitness gives $\widehat{B}_e(x)(1 - \widehat{B}_e(x)) = Z(x) q_{2,e}(x)$, so the values $b_i := B_e(\omega^{i-1}) = \widehat{B}_e(\omega^{i-1})$ are bits. The cardinality opening gives $\widehat{B}_e(0) = k_e n^{-1}$. Since $xZ(x)$ also vanishes at 0, we have $\widehat{B}_e(0) = B_e(0)$. Because B_e has degree at most $n-1$ and is defined over the roots-of-unity domain, the roots-of-unity averaging identity gives $B_e(0) = n^{-1} \sum_i B_e(\omega^{i-1}) = n^{-1} \sum_i b_i$. Therefore $k_e = \sum_i b_i$.

Second, consistency gives $\widehat{B}_e(x) r_e(x) - \widehat{f}_e(x) = Z(x) q_{1,e}(x)$, so for every selected index i with $b_i = 1$, the committed value satisfies $f_i = r_{i,e}$. Third, disjointness gives $\widehat{B}_e(x) \tilde{b}_{e-1}(x) = Z(x) q_{D,e}(x)$, so the accepted current bitmap is disjoint from all previously accepted bitmaps.

It remains to argue that every selected honest index corresponds to an honestly spent receipt. The commitments to b_e and f_e are sent before seed_e is revealed. For any honest user whose receipt was not delivered to the corrupted prover before this commitment step, the value $r_{i,e} = \mathsf{PRF}(\mathsf{seed}_e; i)$ is computationally indistinguishable from a fresh uniform field element to $\mathcal{A}$. Thus, except with probability at most $1/|\mathbb{F}|$ per guessed index, $\mathcal{A}$ cannot commit to $f_i = r_{i,e}$ before seeing the seed unless it has received that honest user's receipt. Therefore, except with negligible probability, every honest user counted by an accepting proof is in $U'_{\mathcal{H}}$, and every remaining counted user can be charged to a corrupted user through the SpendExtra interface. Hence $\mathsf{Bad} = 1$ only with negligible probability, and $\mathcal{F}_{\mathsf{VAR}}$ produces the same verifier output as in the real world except with negligible probability.

Case 3. $\mathcal{P}$ is honest, but $\mathcal{V}$ and a set $U_{\mathcal{C}}$ of users are corrupted. By our threat model, the corrupted verifier is semi-honest. We construct $\mathcal{S}$ as follows.

Initialize. $\mathcal{S}$ receives pp and gives it to $\mathcal{A}$. It obtains $U_{\mathcal{C}}$ and the verification key vk output by the semi-honest verifier. It sends $(\mathsf{sid}, \textsc{Init}, U_{\mathcal{C}})$ to $\mathcal{F}_{\mathsf{VAR}}$. It initializes local epoch counter $e := 1$, cumulative accepted count $K := 0$, dummy spent set $U_D := \emptyset$, and corrupted-user spend record $U'_{\mathcal{C}} := \emptyset$.

Issue. Upon $(\mathsf{sid}, \textsc{Issue}, T, e_{\max})$ from $\mathcal{F}_{\mathsf{VAR}}$, $\mathcal{S}$ receives from $\mathcal{A}$ the seeds $\mathsf{seed}_1, \ldots, \mathsf{seed}_{e_{\max}}$ generated by the semi-honest verifier. Since the verifier is semi-honest, these seeds are distributed exactly as in the real protocol.

Spend. When a corrupted user spends a receipt, $\mathcal{S}$ verifies it using vk. If it verifies and the user has not spent before, $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{Spend}, \mathcal{U})$ to $\mathcal{F}_{\mathsf{VAR}}$ and updates $U'_{\mathcal{C}} := U'_{\mathcal{C}} \cup \{\mathcal{U}\}$. The simulator records the corrupted users whose spending behavior is known to $\mathcal{A}$.

Audit. When $\mathcal{A}$ initiates an audit and $\mathcal{F}_{\mathsf{VAR}}$ outputs $(\mathsf{sid}, \textsc{AuditOK}, K^\star)$ to the corrupted verifier, $\mathcal{S}$ sets $k_e := K^\star - K$. It then chooses a dummy set $S_e^\star$ of size k_e that contains exactly the corrupted users whose spending is known to $\mathcal{A}$ in this epoch and is otherwise filled with arbitrary honest users not previously used in the dummy execution. The set is chosen disjoint from the previous dummy accumulator. The simulator then runs the honest prover algorithm using $S_e^\star$ and the seed seed_e, producing commitments and proofs exactly as an honest prover would. It sends the resulting audit transcript to $\mathcal{A}$, updates $K := K^\star$, updates $U_D := U_D \cup S_e^\star$, and increments e.

If $\mathcal{F}_{\mathsf{VAR}}$ outputs $(\mathsf{sid}, \textsc{AuditBad})$, $\mathcal{S}$ simulates the corresponding rejecting audit outcome.

We now argue that the semi-honest verifier's view is indistinguishable in the real and ideal executions. Fix any epoch e. Let $s_e := \widehat{b}_e(\tau)$ and $t_e := \widehat{f}_e(\tau)$. Since $\widehat{b}_e(x) = b_e(x) + \rho_{b_e} x Z(x)$ and $\tau Z(\tau) \neq 0$, where, in both the real and ideal worlds ρ_{b_e} is chosen uniformly at random, the value s_e is uniform over $\mathbb{F}$ and independent of the support of b_e. Similarly, since $\widehat{f}_e(x) = f_e(x) + \rho_{f_e} Z(x)$ and $Z(\tau) \neq 0$, and ρ_{f_e} is chosen uniformly at random in both worlds, the value t_e is uniform over $\mathbb{F}$ and independent of the support of f_e. Thus $(\mathsf{cm}_{b_e,1}, \mathsf{cm}_{b_e,2}, \mathsf{cm}_{f_e})$ is distributed as $(g_1^{s_e}, g_2^{s_e}, g_1^{t_e})$, in both the real and ideal worlds, where s_e, t_e are fresh uniform scalars. Conditioned on the public count and on the verification equations being satisfied, this distribution is independent of the support of b_e.

The remaining proof elements do not reveal the support either. For checks C1, C2 and C4, observe that only the randomized values s_e and t_e determine their exponents. The consistency proof satisfies $q_{1,e}(\tau) = (s_e r_e(\tau) - t_e)/Z(\tau)$, so its evaluation is determined by s_e, t_e and public $r_e(\tau)$. The bitness proof satisfies $q_{2,e}(\tau) = s_e(1 - s_e)/Z(\tau)$, so it is determined by s_e. The cardinality opening proof satisfies $q_{\mathsf{card},e}(\tau) = (s_e - k_e n^{-1})/\tau$, so it is determined by s_e and the public count.

For checks C3 and D, we show that any unblinded expression which is deterministically computed using $b_e(x)$, is blinded with fresh randomness when used to compute the proofs in the real world and simulated in the ideal world. The structure proof is zero-knowledge by the Schnorr/Fiat-Shamir argument: the first message T is computed from fresh random masks $(a_0, b_0(x))$, and the response $(a^\star, B^\star, H^\star)$ is masked by those fresh values, revealing no information about the decomposition of $\mathsf{cm}_{b_e,1}$. Finally, the disjointness proof depends on $\tilde{b}_{e-1}(\tau)$, which is uniform by induction because it is a sum of independently blinded previous audit polynomials.

Therefore, conditioned on the public cumulative count sequence and the corrupted users' known spending behavior, the transcript distribution is independent of which honest users are selected. The dummy transcripts produced by $\mathcal{S}$ are therefore indistinguishable from real transcripts. This proves privacy against a semi-honest verifier.

Combining the three cases, P-VAR realizes $\mathcal{F}_{\mathsf{VAR}}$ under the stated assumptions. □