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

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 involving 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. For a million users, issuance takes less than 2 seconds for either scheme, and for audit proving time, P-VAR requires less than 10 seconds and S-VAR requires less than 35 seconds. Compared to our schemes, baseline and existing solutions are either at least an order of magnitude slower in proving and verification time, or they do not scale to one million users. Our benchmarks demonstrate that our VAR protocols can be used to enable verifiable and privacy-preserving user engagement auditing at scale. Finally, we showcase how VAR can be integrated with the aforementioned applications.

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 corporate contracts 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.

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 C to reimburse P based on the self-reported number of visits. However, self-reporting is prone to fraud because P has an incentive to *inflate* the report, which often is 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, causing 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 cryptographic solutions to 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 supposed 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 any time, the principal can request and verify a succinct cryptographic proof for the number of receipts possessed 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 example illustrates the need for *user privacy*: employees do not wish to reveal whether/when

*. These authors contributed equally to this work.

they use such services to their employer because this may disclose sensitive medical information. Thus, VAR aims to guarantee that $\mathcal{V}$ learns only the count, not individual identities. However, hiding user identities from the service provider is often infeasible (e.g., the service provider may need user identities for verification). VAR instead aims for what seems to be the second best guarantee – *deniability* – that the protocol does not enable $\mathcal{P}$ or $\mathcal{V}$ to *prove* to a third party that a given user has been serviced (e.g., viewed a particular ad, downloaded a certain video, or visited a specific clinic).

Finally, a VAR system should be performant to support large-scale applications involving millions of users. Moreover, we require that the audit cost of the principals and users does not scale with the number of users. Since the $\mathcal{P}$ serves users, it incurs a cost at least linear in the number of users.

Strawman solutions that do not work. It is informative to explain why closely related primitives do not solve the problem of VAR. Mostly related are aggregate signatures, such as BLS [4]. One strawman idea is to ask users for their signatures when receiving the service, and use the aggregate signature (of constant size) as audit evidence. However, this simple approach violates both privacy and deniability. First, to verify an aggregate signature, $\mathcal{V}$ needs to know the set of public keys used in the aggregation, which violates privacy. Even though this can be mitigated using zero-knowledge proof techniques [5], requiring users to sign violates deniability, as user signatures are irrefutable proof that they received a specific service. Later in §3.3, we explain how other related primitives, such as ring signatures and Aggregate MAC, do not meet performance or security requirements.

1.1. Two Constructions

We present two constructions of VAR using different techniques.

S-VAR: VAR from multi-secret sharing. The first construction S-VAR is a novel use of a threshold secret sharing scheme [6], or SS for short. This construction supports what we call *tiered* auditing. The intuition can be conveyed with a single-secret SS, though the challenges stem from reducing the bandwidth and computation overhead with a suitable multi-secret SS. To construct VAR, the key observation is that in an (n, t)-SS scheme, the ability to reconstruct the original secret implies the possession (or knowledge) of t shares or more. In a strawman construction, $\mathcal{V}$ first determines a series of count thresholds $t_1, \cdots, t_k$ that she would like to use in future audits (e.g., $t_i = 2^i$). Then $\mathcal{V}$ samples k secret values $s_1, \cdots, s_k$ and each user is issued signed shares of s_i for threshold t_i, for all $i \in [k]$. When a user receives service from $\mathcal{P}$, she reveals all k secret shares. To audit, $\mathcal{P}$ identifies the highest threshold for which she has enough shares for reconstruction, say t_j, then sends (t_j, s'_j) to $\mathcal{V}$, where s'_j is the secret reconstructed from users' shares. Verification simply requires checking that s'_j matches the original secret with threshold t_j.

Starting from this idea, we applied several optimizations. First, in the above protocol, the receipt size for each user grows with the number of tiers. In the optimized version (see §4.3), we reduce this to a constant by giving each user a seed that generates all of her shares, while still allowing threshold reconstruction of all secrets using Bottom-Up Secret Sharing [7]. Second, applying FFT to speed up secret reconstruction can improve the performance drastically, but direct implementation of FFT requires the entire set of user secrets to be available, which does not work for threshold reconstruction. To efficiently reconstruct from any subset, we use the *barycentric Lagrange* formula [8] and compute the necessary weights in near-linear time via a subproduct tree and multi-point evaluation [9]. So far, a natural limitation of S-VAR is that thresholds are fixed at issuance time and cannot adapt to realized user spending. In §4.4, we propose a minimal modification to the audit procedure in S-VAR, which yields *dynamic* thresholds while leaving issuance and spending unchanged.

Overall, S-VAR is a novel use of secret sharing, featuring fast issuance and audit, as well as a constant receipt size. Moreover, it does not depend on pairings (in contrast to the next solution), which have only limited support in major certification and evaluation frameworks (e.g., SOG-IS).

P-VAR: VAR from pairings. Our second construction P-VAR removes the fixed tiers, allowing for "accurate" auditing as opposed to fuzzy. P-VAR relies on a one-time trusted setup, and has better performance than S-VAR during auditing, but its security is shown in the AGM model.

To convey intuition, we present a simplified version of P-VAR that only supports one-time audit, that is, $\mathcal{V}$ is limited to invoke only one instance of the audit procedure for each issuance. At a high level, receipts are signed field elements from $\mathcal{V}$. At audit time, the prover $\mathcal{P}$ proves that it counted a hidden subset of receipts *correctly* while revealing only the total count k using pairing-based polynomial commitments and proofs.

Setup. Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be pairing-friendly groups with a bilinear mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and an associated scalar field $\mathbb{F}$, and $g_1 \in \mathbb{G}_1$ and $h, g_2 \in \mathbb{G}_2$ be random generators. The setup process publishes *powers-of-τ* vectors $\boldsymbol{g_1} := (g_1, g_1^{\tau}, \ldots, g_1^{\tau^N})$, $\boldsymbol{g_2} := (g_2, g_2^{\tau}, \ldots, g_2^{\tau^N})$, $\boldsymbol{h} := (h, h^{\tau}, \ldots, h^{\tau^{N-2}})$ for a secret trapdoor $\tau \leftarrow \mathbb{F}$. We pick N to be an integer larger than the number of users such that an N-th primitive root of unity ω exists. Let $\Omega = \{1, \omega, \ldots, \omega^{N-1}\}$ be the set of N-th roots of unity. These values form the public parameters of P-VAR $\mathsf{pp} := \big(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbb{F}, e(\cdot,\cdot), \boldsymbol{g_1}, \boldsymbol{g_2}, \boldsymbol{h}, \Omega\big)$.

Issuance. First, $\mathcal{V}$ samples a random value seed to derive all receipts for the current protocol instance. For user $\mathcal{U}_i$, $\mathcal{V}$ computes $r_i = \mathsf{PRF}(\mathsf{seed}; i)$ and issues a receipt $\mathsf{rx}_i = (r_i, \sigma_i)$ where σ_i is a signature. Let $r(x)$ be a polynomial of degree $(N-1)$ with $r(\omega^{i-1}) = \mathsf{PRF}(\mathsf{seed}; i)$ for $i \in [N]$. $\mathcal{V}$ computes the commitment $\mathsf{cm}_r := g^{r(\tau)}$.

Spending. To spend her receipt, user $\mathcal{U}_i$ submits $\mathsf{rx}_i = (r_i, \sigma_i)$ to $\mathcal{P}$, who verifies the signature and stores (r, i).

Audit. Suppose k users have spent their receipts. Then $\mathcal{P}$ convinces $\mathcal{V}$ of the count k as follows.

- *Receipt commitments:* $\mathcal{P}$ commits to the collected receipts. Let $\boldsymbol{b} = (b_1, \ldots, b_N)$ be a bit vector with $b_i = 1$ if $\mathcal{P}$ has a receipt from user $\mathcal{U}_i$. $\mathcal{P}$ computes $\boldsymbol{f} = (f_i, \ldots, f_N)$ such that $f_i = r_i$ if $b_i = 1$, and $f_i = 0$ otherwise. Then $\mathcal{P}$ commits to $(\boldsymbol{b}, \boldsymbol{f})$ using KZG polynomial commitments (§2), and let $b(x), f(x)$ denote the committed polynomials.
- *Seed reveal:* $\mathcal{V}$ reveals seed to $\mathcal{P}$, who computes all receipts and interpolates $r(x)$ over Ω with $r(\omega^{i-1}) = \mathsf{PRF}(\mathsf{seed}; i)$ for $i \in [N]$.
- *Correctness proofs:* $\mathcal{P}$ shows the correctness of the claimed count k with the following proofs.
  - First, $\mathcal{P}$ shows that f is consistent with r, i.e., $f_i = b_i \cdot r_i$ for all $i \in [N]$. To do so, $\mathcal{P}$ computes $q_1(x)$ such that $b(x) \cdot r(x) - f(x) = q_1(x) \cdot Z(x)$ and $\pi_1 = g_1^{q_1(\tau)}$.
  - Second, $\mathcal{P}$ shows that $b(x)$ evaluates to $\{0, 1\}$ over Ω with the proof $\pi_2 = g_1^{q_2(\tau)}$, where $b(x)(1 - b(x)) = Z(x)q_2(x)$.
  - Finally, $\mathcal{P}$ shows that there are exactly k non-zero elements among the b_is via univariate sumcheck [10].

Supporting multi-shot audit. Note that in the above strawman, $\mathcal{V}$ reveals the seed, which enables $\mathcal{P}$ to generate succinct correctness proofs but also arbitrarily inflate claims in future proofs. This is why the protocol is only secure if the audit is run only once for the issued receipts. In §5, we show how to support multi-shot verification. The main idea is to issue new receipts after each audit, yet prevent a user from spending multiple receipts during an audit.

Supporting dynamic user sets. So far, we have assumed a fixed upper bound n on the number of users chosen at setup, and the audit cost scales with n. In many applications, however, the user population is *dynamic*: the system starts with a small set of users and new users may join over time. Running the multi-shot audit procedure with a large n in this setting is wasteful, since the first few audits may involve only a tiny fraction of the eventual user base. In §5.3, we show how to extend our construction to handle dynamic user sets more efficiently, so that the prover and verifier costs scale with the *current* active population (or newly added users) instead of the worst-case bound fixed at setup.

1.2. Applications of VAR

To showcase the power of VAR, we discuss three socially important applications in §6: auditing social media platforms, preventing fraudulent reimbursement claims, and detecting censorship in broadcast. We present a brief overview below.

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 [11], [12], [13], [14], [15], [16]. Recent regulatory efforts, such as the EU Digital Services Act [17], highlight the need for greater scrutiny, and 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.

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.

1.3. Implementation and Evaluation

We implement both S-VAR and P-VAR in Rust, using Ed25519 for signatures and SHA-256 for PRFs and hash functions. For S-VAR, we use the BN254 scalar field and for P-VAR, the pairing-friendly groups and the scalar field are instantiated by BLS12-381. For comparison, we implement two baselines: predicate aggregate signatures (PAS) [5] in Rust and a generic SNARK baseline in Noir [18]. We benchmark on two AWS configurations ("small" and "large"), varying the number of users $n \in \{2^{10}, \ldots, 2^{20}\}$; for S-VAR, we also vary the number of supported thresholds k and use a geometric (i.e., powers-of-2) threshold schedule. We measure issuance bandwidth and time, as well as audit proving and verification time.

Issuance time and bandwidth in both S-VAR and P-VAR scales approximately linearly in the number of users and the number of supported audits. Concretely, on the large VM and for 2^{20} users, S-VAR's issuance takes $0.89\,\mathrm{s}$ for $k = 1$ and $1.88\,\mathrm{s}$ for $k = 20$; in the same regime, P-VAR (with 1 epoch) takes $1.52\,\mathrm{s}$. The issuance costs above are not intended for direct comparison of the two schemes, because they support different auditing styles; in §7.3.1 we propose a comparison regime where both schemes support the same number of audits; in this regime, S-VAR's issuance time outperforms that of P-VAR by an order of magnitude across all values of n.

At audit, P-VAR yields lower proving time than S-VAR beyond the small-n regime; concretely, on the large VM and for 2^{20} users, S-VAR's proving time is $34\,\mathrm{s}$ and P-VAR's proving time is $9.7\,\mathrm{s}$. Verification time is constant and concretely small for both schemes – less than $100\,\mu\mathrm{s}$ for S-VAR and less than $19\,\mathrm{ms}$ for P-VAR. Both PAS and the SNARK baseline are *at least an order of magnitude* slower in proving and verification time compared to our solutions.

The key takeaway is that both proposed protocols are concretely practical at million-user scale and significantly

outperform heavyweight baselines. Among the two protocols, S-VAR achieves lower issuer time and bandwidth when coarse, thresholded audits suffice; P-VAR trades higher issuance cost for faster audits with exact counts. The choice of which to deploy depends on the target audit granularity and which entity (issuer vs. prover) is more powerful; large-issuer/smaller-prover settings favor P-VAR, while lightweight issuer or resource-constrained deployments favor S-VAR.

Contributions

In summary, this paper makes the following contributions:

- We introduce VAR, a new cryptographic primitive with applications in user engagement auditing. We model the security properties of VAR via an ideal functionality (§3).
- We propose two VAR constructions: S-VAR (§4) is a secret-sharing-based protocol which supports tiered auditing, and P-VAR (§5) is a pairing-based protocol which supports exact auditing. We prove both protocols secure with respect to our ideal functionality. We also propose an extension to P-VAR to support dynamic user sets.
- We showcase three diverse applications of VAR (§6).
- We implement and benchmark our protocols. Our experiments show orders of magnitude performance improvement over baseline and existing solutions (§7).

2. Preliminaries

In this section, we go over notations and preliminaries. We refer the reader to §A for additional preliminaries.

Notations. 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., $\boldsymbol{x} = (x_1, \ldots, x_n)$.

Digital signatures. A digital signature scheme is a tuple of PPT algorithms $\mathsf{DS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$. We rely the standard correctness and unforgeability definitions [19].

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 [20], 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 $e(\mathsf{cm}_f, g_2) \stackrel{?}{=} e(\pi, g_2^{Z(\tau)})$.

Hash functions and random oracle model. We model hash functions in the random oracle model (ROM) [21]. We write $\mathsf{H}(x)$ to denote hash function H evaluated on input x. The domain and range of H will always be clear when used.

Pseudo-random function. A family of polynomial-time functions are pseudorandom functions (PRFs) if a randomly selected function from the family cannot be distinguished from a truly random function by a PPT adversary. We write $\mathsf{PRF}(\mathsf{seed}; x)$ to denote the evaluation of pseudorandom function PRF with seed seed on input x. We use $\mathsf{SHA256}(\mathsf{seed} \| x)$ to instantiate PRFs [22, Section V.A].

Secret sharing. An (n, t)-*single*-secret sharing scheme [6] is a protocol for the distribution of a secret s among n parties such that the recovery of the secret is possible out of t shares where $1 \leq t \leq n$, while $t - 1$ shares give no information on the secret s. In a $(t_1, \ldots, t_k)$-*multi*-secret sharing scheme [23], [24], k independent secrets $s_1, \ldots, s_k$ are distributed among n parties. Each secret s_i can be recovered only if at least t_i parties combine their shares, while any fewer than t_i shares reveal nothing about s_i.

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, 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

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 $\mathcal{F}_{\mathsf{VAR}}$ in Fig. 1. To separate different instances of $\mathcal{F}_{\mathsf{VAR}}$, all messages are tagged with a unique *session id* denoted sid. We refer readers to the original proposal of the Universal Composability (UC) framework [25] for details on ideal protocol execution.

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 sending his claimed count. 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.

Adverserial model. We consider a static, computationally bounded adversary $\mathcal{A}$. Corrupted parties (except for $\mathcal{V}$) may deviate arbitrarily from the protocol and reveal their states to $\mathcal{A}$. Concretely, $\mathcal{P}$ and any subset of users $U_{\mathcal{C}} \subseteq U$ can be *malicious*, but we assume that $\mathcal{V}$ is *semi-honest*. To model corrupted users, at initialization, $\mathcal{A}$ fixes and sends a corrupted user set $U_{\mathcal{C}} \subseteq U$ to $\mathcal{F}_{\mathsf{VAR}}$, and $\mathcal{F}_{\mathsf{VAR}}$ records $R := U_{\mathcal{C}}$.

Below we discuss the correctness, security, and privacy guarantees required of a VAR system, reflected by $\mathcal{F}_{\mathsf{VAR}}$.

- *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.* At a high level, an honest $\mathcal{V}$ wishes to verify the number of issued receipts spent by users, even if $\mathcal{P}$ and a subset of users are corrupted. Informally, if an adversary has corrupted $\mathcal{P}$ and $|U_{\mathcal{C}}|$ users, and if ℓ honest parties spend their receipts, then, $\mathcal{V}$ must only accept an audit count up to $\ell + |U_{\mathcal{C}}|$. Since $\mathcal{P}$ is the only party that interacts with users during the spend procedure, it may always censor some of the receipts it receives; however, recall that 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 spending users as any other $|U'|$-sized subset of honest users.

Deniability. We focus our study on VAR constructions that enable *deniability* on the user side. Specifically, we require that at no point during a VAR protocol can any party obtain *publicly verifiable evidence* linking a user to the spending process (e.g., viewed a particular ad, downloaded a particular video, or visited a specific clinic). For instance, protocols which rely on each user sending a traditional digital signature to $\mathcal{P}$ do *not* satisfy deniability, since the user signature constitutes publicly verifiable evidence that $\mathcal{U}_i$ engaged in the spending process. Deniability and privacy are related but incomparable properties (neither implies the other). Indeed, the non-deniable solution above can still ensure privacy, as long as $\mathcal{P}$'s audit proof hides the identity of the individual signers from $\mathcal{V}$. For simplicity, we do not model the deniability property in our functionality $\mathcal{F}_{\mathsf{VAR}}$.

Efficiency. Broadly, we are interested in VAR protocols that are efficient in terms of *computation time* and *bandwidth* across all parties and procedures. In what follows, 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 bandiwdth 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 $\mathcal{P}$ and of each $\mathcal{U} \in U$; at the same time, we wish to preserve $\mathcal{P}$'s time and and bandwidth at reasonable levels towards practical deployment at scale.

Accuracy. We also consider *accuracy*, which measures how close $\mathsf{claimed}$ is to the actual number of received receipts. We say that a VAR protocol is β-accurate if there exists $\beta \in (0, 1]$ such that, for all honest provers and all successful executions of the audit procedure, we have that $\frac{\mathsf{claimed}}{|R|} \geq \beta$. For instance, if a VAR protocol enables the prover to prove the exact number of spent receipts, then that protocol is 1-accurate. Note that a VAR protocol can satisfy inflation soundness and at the same time not be 1-accurate; that is, inflation soundness and accuracy are different properties.

3.3. Strawman Solutions

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

Aggregate signatures. The simplest solution is to use *digital signatures* that enable *aggregation*, such as BLS signatures [4]. To spend, each user $\mathcal{U}_i$ signs the session id with their signing key, and sends their signature σ_i to $\mathcal{P}$. Upon an audit request, $\mathcal{P}$ aggregates the received signatures into an aggregate signature $\tilde{\sigma}$ and the corresponding public keys pk_i into an aggregate public key apk. First, having users sign violates deniability. Second, to verify $\tilde{\sigma}$, $\mathcal{V}$ needs to know the set of public keys used to form apk, which violates privacy. To mitigate this, $\mathcal{P}$ could employ a zk-SNARK

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 1: Ideal Functionality for VAR with epochs.

to hide the spend set; however, generic SNARKs do not scale, and the deniability issue remains. We provide a short background on zk-SNARKs in §A.

The recent work of Qiu and Tang [5] introduces *Predicate Aggregate Signatures* (PAS), a powerful generalization of aggregate signatures that supports succinctness, signer anonymity, and predicate-based verification. Nonetheless, PAS does not satisfy *deniability*, nor does it scale to the number of users VAR intend to support. (Looking ahead, we report performance of these baseline protocols in §7.)

Ring signatures. An alternative approach is to use *ring signatures* [26], [27], which at first glance seem to solve both the privacy and deniability problem. 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 towards 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.

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 predetermined count values, which form tiers. In other words, this VAR construction supports "fuzzy" auditing, rounded to the closest pre-determined threshold.

The high-level idea is as follows. In *multi-secret sharing* [23], multiple secrets are shared simultaneously, each associated with its own threshold. To construct VAR from multi-secret sharing, the verifier $\mathcal{V}$ chooses distinct thresholds, each corresponding to a distinct audit tier. Then he issues a share for each user, which holds enough information to help towards reconstruction of *all* secrets. These shares serve as the users' receipts: a user spends their receipt simply by sending their share to the prover $\mathcal{P}$. During an audit, $\mathcal{P}$ uses the received shares to reconstruct the secret associated with the highest threshold that has been met. The ability to reconstruct this secret serves as proof to $\mathcal{V}$ that the claimed threshold of receipts was spent.

After introducing an abstraction for multi-secret sharing in §4.1, we present a generic construction for S-VAR in §4.2. In §4.3, we present S-VAR, our optimized instantiation based on a bottom-up secret-sharing scheme. In §4.4, we discuss the efficiency and accuracy implications of different threshold schedules, and present a modification to S-VAR that enables dynamic thresholds. Then in §4.5, we prove S-VAR secure with respect to our ideal functionality.

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 [6] 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. It 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 the random oracle model). 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 [23] 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 specified in Fig. 2.

Setup. Given the security parameter λ and the number of users n, we fix a scalar field $\mathbb{F}$ with order at least $|\mathbb{F}| \ge 2^\lambda$. The protocol uses two building blocks: 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 then chooses a t_{j^*}-sized reconstruction set $I \subseteq [n]$ from R and reconstructs $s' = \mathsf{GMSS.Recon}(\boldsymbol{t}, I, \{\mathsf{sh}_i\}_{i\in I}, \mathsf{aux})$. $\mathcal{P}$ sends (j^*, t_{j^*}, s') to $\mathcal{V}$, who checks if t_{j^*} is one of the thresholds chosen at issuance and whether $s' \stackrel{?}{=} s_{j^*}$. If all checks pass, $\mathcal{V}$ accepts; otherwise, 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. 2, $\mathcal{V}$'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 seed and use it to derive all k secrets. This optimization is orthogonal to how GMSS is instantiated, and thus $\mathcal{V}$'s storage overhead does not inform the choice of said instantiation.

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

We now present two instantiations of GMSS and discuss the performance tradeoffs in the resultant VAR constructions.

A top-down instantiation of GMSS. GMSS can be instantiated by running k independent top-down (e.g., Shamir [6]) single-secret-sharing instances, one per threshold. Under this instantiation, each GMSS share sh_i consists of k Shamir 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 $O(k)$ receipt per user.

Bottom-up secret sharing for $O(1)$ user receipts. Ideally, we would like users to have constant-sized shares, i.e., $|\mathsf{sh}_i| = O(1)$ independent of k. The challenge is enabling reconstruction of different secrets at different thresholds while keeping each user's receipt fixed-size. The *Bottom-Up Secret Sharing (BUSS)* [7] paradigm, which inverts the usual top-down secret sharing process, can help toward this constant-sized-receipt goal. In bottom-up secret sharing, rather than

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 $\mathsf{R} := \emptyset$ 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 $(i, z') \notin \mathsf{R}$ for any z' and $(i', z_i) \notin \mathsf{R}$ for any i'. Then, if $\mathsf{DS.Verify}(vk, i\|z_i, \sigma_i) = 1$, he updates $\mathsf{R} = \mathsf{R} \cup \{(i, z_i)\}$.

Audit

1) $\mathcal{V}$ sends an audit request to $\mathcal{P}$.
2) On receiving an audit request, $\mathcal{P}$ proceeds as follows:
   - Sets $j^* = \max\{j \in [k] \mid t_j \leq |\mathsf{R}|\}$ and chooses a set $I \subseteq [n]$ such that $\{(i, z_i)\}_{i\in I} \subseteq \mathsf{R}$ 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 2: VAR based on GMSS

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 bottom-up 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 is the collection of BUSS public points for all tiers. To share, for each tier j, the dealer fixes an n-degree polynomial f_j whose user evaluations are set to $f_j(i) = \mathsf{H}(j, x_i)$ and whose constant term is the tier's secret, where H is a hash function; 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 as a random oracle. 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 $\big(f_j(-1), \ldots, f_j(-(n-t+1))\big)$. Concatenate these $(n - t + 1)$ 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 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 the prover's storage in some cases. For instance, under an arithmetic threshold schedule (see §4.4 below) with step size 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 storage by a factor of 2. We note that bottom-up GMSS always incurs the $|\mathsf{aux}|$-sized upfront prover storage, which is not the case for top-down GMSS.

Accuracy. S-VAR's accuracy depends on the choice of threshold schedule, discussed in §4.4. Under a geometric schedule with accuracy parameter ϵ, the resulting S-VAR protocol is ϵ-accurate, per the definition of accuracy in §3.

4.4. On Threshold Schedules

Choosing a threshold schedule. S-VAR leaves the threshold schedule up to the $\mathcal{V}$'s choice; in practice, this choice will be application-specific. Nevertheless, the core tradeoff is simple: adding more thresholds yields finer-grained auditing – i.e., the prover can prove a count closer to the true number of spent receipts – but it also increases system overhead.

Two natural schedule choices are: *arithmetic* and *geometric*. In an arithmetic schedule, thresholds are spaced by a fixed increment, giving uniform resolution across the range. In a geometric schedule, thresholds grow by a constant

multiplicative factor, which concentrates tiers near small counts and becomes sparser at large counts.

Arithmetic schedule. Fix a step $\Delta \in \mathbb{N}$ and set $k \leftarrow \lceil \frac{n}{\Delta} \rceil$. Define $\boldsymbol{t} = (t_j)_{j\in[k]}$ by $t_j = \min\{n,\ j\Delta\}$ for $j \in [k]$. For example, for $n = 10^6$ users, one can choose $\Delta = 10^3$ to yield $k = 1{,}000$ thresholds at multiples of 1,000 up to n.

With an arithmetic schedule, there always exists a threshold that is within (additive) distance Δ of the true number of spent receipts, However, if Δ is much smaller than n, then the number of thresholds k essentially grows with n. In general, an arithmetic schedule is well-suited for applications that either require high precision at the expense of $k = \Theta(n)$ thresholds, or for applications that can work with coarsely chosen thresholds by using large Δ (e.g., $\Delta = \Theta(n)$).

Geometric schedule. Fix an accuracy parameter $\varepsilon > 0$ and set $k \leftarrow \lceil \log_{1+\varepsilon} n \rceil + 1$. Define $\boldsymbol{t} = (t_j)_{j\in[k]}$ by $t_1 = 1$ and

$$t_{j+1} = \min\{n,\ \lceil (1+\varepsilon)\, t_j \rceil\} \quad \text{for } j \in [k].$$

For example, for $n = 10^6$ users, one can choose $\varepsilon = 1$ to yield $\log_2(10^6) \approx 20$ thresholds at powers of 2 up to n.

With a geometric schedule, there always exists a threshold that is within (multiplicative) distance ε of the true number of spent receipts. Unlike the arithmetic schedule, the number of thresholds k in this case grows with $\log_{1+\varepsilon} n$. However, as the target accuracy ε approaches 1, k can grow very large. For instance, for $n = 10^6$ users and accuracy $\varepsilon = 0.01$, we have $k \approx 1{,}400$ thresholds. In general, a geomteric schedule is well-suited for applications that wish to have few ($O(\log n)$) thresholds (assuming ε is not too small) and are willing to settle for multiplicative precision.

Hybrid schedule. Applications can also choose a *hybrid* schedule; that is, they can use an arithmetic schedule up until a threshold $n' < n$, and then *switch* to a geometric schedule between n' and n. Such a hybrid approach can be useful if the application's desired accuracy ε is small. Back to the $n = 10^6$ and $\varepsilon = 0.01$ example, suppose we set the switching threshold at $n' = 10^3$, and use an arithmetic schedule with $\Delta = 10$ up until n'. Then this hybrid schedule will have approximately 100 thresholds in $[0, n']$ and 700 thresholds in $[n', n]$, for a total of $k \approx 800$, which is significantly less than the 1,400 thresholds used by a pure geometric schedule. The reason behind this improvement is that out of the 1,400 thresholds, approximately 700 were concentrated between 0 and 1,000; these are precisely the thresholds that the hybrid schedule gets rid of, and replaces them with only 100 arithmetic thresholds.

Toward dynamic thresholds. A natural limitation of S-VAR is that thresholds are fixed at issuance time and cannot adapt to realized user spending. This can be restrictive in applications where $\mathcal{V}$ expects a fresh proof of count at regular intervals; if user engagement is low during a given period, $\mathcal{P}$ may be unable to produce an audit proof for the scheduled threshold. In such cases, a smaller but still meaningful count might have been attained, and it is desirable that $\mathcal{P}$ be able to prove that smaller count.

We propose a minimal modification to the audit procedure in S-VAR, which yields *dynamic*, downward-adjustable thresholds while leaving issuance and spending unchanged. At audit time, suppose $\mathcal{P}$ wishes to prove t_{j^*} as the next threshold, but he only holds $t_{j^*} - d$ user receipts for some $d > 0$. To fix this, $\mathcal{P}$ can send the difference d to $\mathcal{V}$, who then sends back d additional evaluation points of f_{j^*}; these d points are disjoint from the points already in aux and the user points. If $\mathcal{P}$ can then reconstruct the target secret s_{j^*}, $\mathcal{V}$ is convinced that at least $t_{j^*} - d$ users spent their receipts. Intuitively, the extra public points "stand in" for the missing users, allowing $\mathcal{V}$ to relax thresholds on the fly depending on user engagement, while preserving the soundness and privacy guarantees of the base S-VAR protocol.

4.5. Security

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. 2, 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 $\mathsf{R} = \emptyset$.

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 $(i, \cdot) \notin \mathsf{R}$ and $(\cdot, \mathsf{rx}^*) \notin \mathsf{R}$. 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 $\mathsf{R} = \mathsf{R} \cup \{(i, \mathsf{rx}^*)\}$.

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) $(i, \cdot) \notin$ R, (b) $(\cdot, \mathsf{rx}^*) \notin$ 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 $(\mathsf{sid}, \text{AUDITOK}, \mathsf{claimed})$ 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 $(\mathsf{sid}, \text{INIT}, U_{\mathcal{C}})$ 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 $(\mathsf{sid}, \text{ISSUE}, T, e_{\max})$ 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 $(\mathsf{sid}, \text{SPEND}, \mathcal{U}_i)$ 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 $(\mathsf{sid}, \text{AUDITREQ})$ 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. 2). 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 $(\mathsf{sid}, \text{SPENDEXTRA}, U''_{\mathcal{C}})$ 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 $(\mathsf{sid}, \text{AUDITRESP}, 1, \mathsf{claimed})$ to $\mathcal{F}_{\mathsf{VAR}}$.
- Case (B): $\mathcal{V}_{\text{S-VAR}}$ outputs (reject). In this case, $\mathcal{S}$ sends $(\mathsf{sid}, \text{AUDITRESP}, 0, \mathsf{claimed})$ to $\mathcal{F}_{\mathsf{VAR}}$.

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 $(\mathsf{sid}, \text{AUDITRESP}, 0, \mathsf{claimed})$ to $\mathcal{F}_{\mathsf{VAR}}$, which in turn sends $(\mathsf{sid}, \text{AUDITBAD})$ 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 $(\mathsf{sid}, \text{AUDITBAD})$ 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*. Here spent is the cardinality of the R set *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 $\mathsf{claimed} > \mathsf{spent}$. Let 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 $\mathsf{claimed} < \mathsf{spent}$). 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) $|U''_{\mathcal{C}}| = \mathsf{claimed} - \mathsf{spent}'$.
2) $|U''_{\mathcal{C}}| = |U_{\mathcal{C}} \setminus U'_{\mathcal{C}}|$.

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

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 can 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. 2 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. 2) 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. □

5. P-VAR: VAR Based on Pairings

In this section, we present P-VAR, a pairing-based construction for VAR. As a warm-up, one can build a simple pairing-based protocol (as sketched in §1.1) directly from prior work on polynomial commitments and inner-product arguments [28], [29]. This construction, however, fundamentally supports only *one-shot* verification: after a run of the audit procedure, the verifier has revealed enough information (e.g., seeds or committed randomness) that the prover effectively learns all minted receipts, so any subsequent audit would no longer be sound. This limits

the usefulness of the construction and falls short of our ideal functionality, which allows the verifier to make audit requests multiple times.

In §5.1, we present a construction that supports *multi-epoch* auditing and realizes our ideal functionality. Beyond the techniques of [28], [29], we introduce (i) audits with a rolling accumulator across epochs and succinct disjointness proofs to show that no users are counted twice, and (ii) a *consistency* check that binds the counted users so that a malicious prover cannot inflate the count. In §5.2, we prove P-VAR secure with respect to our ideal functionality. Then in §5.3, we present an optimization called "dynamic user sets", so that the overhead of the protocol only scales with the *actual* number of receipts, instead of the maximum number of users determined at setup time.

5.1. Protocol Details

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, which is set to be 1 initially and has an upper bound $e_{\max}$. To issue receipts, $\mathcal{V}$ samples random field elements $\{r_i\}$ and sends each user $\mathcal{U}_i$ the tuple (r_i, σ_i) where σ_i is the signature on r_i under $\mathcal{V}$'s key. When users spend, the prover $\mathcal{P}$ verifies the signature and collects the receipts. At audit time, it proves that it counted a hidden subset of the receipts correctly while revealing only the total count of collected receipts, denoted k, in this epoch, and proves that no user is counted twice across epochs. After an audit, both $\mathcal{P}$ and $\mathcal{V}$ increase the epoch number by 1.

This protocol realizes the ideal functionality $\mathcal{F}_{\mathsf{VAR}}$: it enables an honest $\mathcal{V}$ to get the exact number of spent receipts, and hides the exact users who spent the receipts from any $\mathcal{V}$, and only the count k is revealed. The main cryptographic challenge is to guarantee that no user will be counted more than once across all epochs, even if a malicious user submits receipts across multiple epochs, in an attempt to inflate the count. We achieve this by accumulating a commitment to the bit vector and performing disjointness checks per epoch.

Setup. P-VAR requires a trusted setup to generate the common reference string (CRS) for the KZG polynomial commitment scheme. A trusted party runs the Setup algorithm $\mathsf{Setup}(1^\lambda, n)$, where λ is the security parameter and n is the number of users. The algorithm produces public parameters for the digital signature scheme DS and the CRS for the KZG commitment scheme. 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 $\mathbb{F}$. Fix uniformly random generators $g_1 \in \mathbb{G}_1$ and $h, g_2 \in \mathbb{G}_2$. Then it chooses a trapdoor $\tau \leftarrow \mathbb{F}$ and publish *powers-of-τ* vectors $\boldsymbol{g_1} := (g_1, g_1^\tau, \ldots, g_1^{\tau^n})$, $\boldsymbol{g_2} := (g_2, g_2^\tau, \ldots, g_2^{\tau^n})$, $\boldsymbol{h} := (h, h^\tau, \ldots, h^{\tau^{n-2}})$. The trapdoor τ itself is *not* published. The algorithm also outputs the n-th roots of unity $\Omega = \{1, \omega, \ldots, \omega^{n-1}\} \subset \mathbb{F}^\times$ with vanishing polynomial $Z(x) := Z_\Omega(x) = \prod_{\alpha \in \Omega}(x - \alpha) = x^n - 1$. For a set of elements A in a field $\mathbb{F}$, the *vanishing polynomial* over A is defined as $Z_A(x) := \prod_{a \in A}(x - a)$. The public CRS is $\mathsf{pp} := \big(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbb{F}, e(\cdot,\cdot), \boldsymbol{g_1}, \boldsymbol{g_2}, \boldsymbol{h}, \Omega\big)$, which is given to both $\mathcal{P}$ and $\mathcal{V}$. In addition, $\mathcal{V}$ runs DS key generation to obtain $(sk, vk) \leftarrow \mathsf{DS.Gen}(1^\lambda)$ and publishes vk.

$\mathcal{P}$ maintains a cumulative polynomial $\tilde{b}_e(x)$, whose evaluation at ω^{i-1} denotes whether or not $\mathcal{U}_i$ has spent in epochs $1, \ldots, e$, i.e., $\tilde{b}_e(\omega^{i-1}) = 1$ if $\mathcal{U}_i$ has spent a valid receipt in epoch e or 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 be the 0 polynomial, and $\tilde{\mathsf{cm}}_0$ is initialized to be g_1^0. $\mathcal{V}$ chooses and publishes the maximum number of epochs $e_{\max}$, and the epoch number is initialized to be $e = 1$.

Issuance. For each $e \in [e_{\max}]$, $\mathcal{V}$ samples a new random seed seed_e. For each user $i \in [n]$, $\mathcal{V}$ computes $r_i \leftarrow \mathsf{PRF}(\mathsf{seed}_e; i)$ and issues a receipt $\mathsf{rx}_{i,e} = (r_i, e, \sigma_i)$ to $\mathcal{U}_i$ where $\sigma_i = \mathsf{DS.Sign}(sk, r_i \| e)$. Let $r_e(x)$ be a polynomial

---

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) Sample $\tau \xleftarrow{\$} \mathbb{F}$.
3) Compute $\boldsymbol{g_1} := (g_1, g_1^\tau, \ldots, g_1^{\tau^n})$, $\boldsymbol{g_2} := (g_2, g_2^\tau, \ldots, g_2^{\tau^n})$, and $\boldsymbol{h} := (h, h^\tau, \ldots, h^{\tau^{n-2}})$.
4) Let $\Omega = \{1, \omega, \ldots, \omega^{n-1}\}$ be the n-th roots of unity of $\mathbb{F}$.

The trusted party outputs public parameters $\mathsf{pp} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbb{F}, e(\cdot,\cdot), \boldsymbol{g_1}, \boldsymbol{g_2}, \boldsymbol{h}, \Omega)$ 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 a commitment $\tilde{\mathsf{cm}}_0 := g_1^0$, 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 $\mathsf{rx}_{i,e} := (r_i, e, \sigma_i)$ 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 $\mathsf{cm}_{r_e} := g_1^{r_e(\tau)}$.
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 any epoch $t' \neq t$, and receipts from users who have spent receipts in previous epochs.
3) On receiving a receipt $\mathsf{rx}_{i,e} = (r_i, e, \sigma_i)$ 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 3: Setup, Issue, and Spend procedures of P-VAR.

Audit (epoch e)

1) $\mathcal{V}$ sends an audit request to $\mathcal{P}$ for epoch e. If $e > e_{\max}$, $\mathcal{P}$ and $\mathcal{V}$ halt.
2) *Receipt commitments.* Define $(b_i)_{i=1}^n$ where $b_i = 1$ if $(r_i, i) \in \mathrm{R}$ for some r_i, otherwise $b_i = 0$. Also define $(f_i)_{i=1}^n$ where $f_i = r_i$ if $(r_i, i) \in \mathrm{R}$ 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.
$$\widehat{b}_e(x) := b_e(x) + \rho_{b_e} Z(x), \qquad \widehat{f}_e(x) := f_e(x) + \rho_{f_e} Z(x).$$
$\mathcal{P}$ 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) *Correctness 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 $\widehat{b}_e(x)(1 - \widehat{b}_e(x)) = Z(x) q_{2,e}(x)$. Then $\mathcal{P}$ sends $\pi_{2,e} := g_1^{q_{2,e}(\tau)}$ to $\mathcal{V}$.
   - **(C3) Cardinality**: $\mathcal{P}$ computes $q_{3,e}(x), q_{4,e}(x)$ with $\deg(q_{4,e}) \le n - 2$ such that $\widehat{b}_e(x) = q_{3,e}(x) Z(x) + q_{4,e}(x) x + k_e \cdot n^{-1}$. Then $\mathcal{P}$ sends $\pi_{3,e} := g_1^{q_{3,e}(\tau)}$, $\pi_{4,e} := g_1^{q_{4,e}(\tau)}$, $\pi_{5,e} := h^{q_{4,e}(\tau)}$.
5) *Disjointness proof.*
   - **(D) Disjointness**: $\mathcal{P}$ computes $q_{D,e}$ 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 $\mathcal{V}$.
6) *Verifier checks.* $\mathcal{V}$ accepts the claimed count k_e in this epoch if and only if the following checks pass:
   - **(C1) Consistency**: $e(\pi_{1,e}, g_2^{Z(\tau)}) \cdot e(\mathsf{cm}_{f_e}, g_2) \stackrel{?}{=} e(\mathsf{cm}_{b_e,1}, g_2^{r_e(\tau)})$.
   - **(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) Cardinality**: $e(\pi_{3,e}, g_2^{Z(\tau)}) \cdot e(\pi_{4,e}, g_2^{\tau}) \cdot e(g_1^{k_e \cdot n^{-1}}, g_2) \stackrel{?}{=} e(\mathsf{cm}_{b_e,1}, g_2)$, and $e(\pi_{4,e}, h) \stackrel{?}{=} e(g_1, \pi_{5,e})$.
   - **(D) Disjointness**: $e(\pi_{D,e}, g_2^{Z(\tau)}) \stackrel{?}{=} e(\mathsf{cm}_{b_e}, \tilde{\mathsf{cm}}_{e-1})$
7) *Updates.* $\mathcal{P}$ updates $\tilde{b}_e(x) := \tilde{b}_{e-1}(x) + \widehat{b}_e(x)$. $\mathcal{V}$ updates $\tilde{\mathsf{cm}}_e := \tilde{\mathsf{cm}}_{e-1} \cdot \mathsf{cm}_{b_e}$ and $k := k + k_e$. They both update the epoch number $e := e + 1$.

Figure 4: Audit procedure of P-VAR.

of degree $(n-1)$ with $r_e(\omega^{i-1}) = \mathsf{PRF}(\mathsf{seed}_e; i)$ for $i \in [n]$. $\mathcal{V}$ computes $\mathsf{cm}_{r_e} := g_1^{r_e(\tau)} \in \mathbb{G}_1$, and $\mathcal{P}$ initializes the set of received receipts for this epoch to be $\mathrm{R}_e := \emptyset$.

Deployment note. For simplicity of exposition, the specification above generates all epoch receipts up front (e.g., distributing $\{\mathsf{rx}_{i,e}\}_{i \in [n],\, e \in [e_{\max}]}$ at onboarding). In practice, the same security and privacy guarantees hold if receipts are *streamed on demand*: at the start of epoch e, $\mathcal{V}$ sends the epoch's receipt batch to $\mathcal{P}$, who relays receipts only to users who have not yet submitted in prior epochs. This reduces user storage and issuance bandwidth without changing any audit proofs or verification checks. This optimization is suitable for applications where users receive receipts from the service provider (e.g., see §6.1). We discuss streaming delivery of receipts in §9.

Spending. Before spending, user $\mathcal{U}_i$ interacts with $\mathcal{P}$ to learn the current epoch number e. To spend for this epoch, $\mathcal{U}_i$ submits her receipt $\mathsf{rx}_{i,e} = (r_i, e, \sigma_i)$ to $\mathcal{P}$, who in turn checks if $\mathsf{DS.Verify}(vk, i \| r_i \| e, \sigma_i) \stackrel{?}{=} 1$. If the check passes, $\mathcal{P}$ adds the receipt (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 $e > e_{\max}$, both $\mathcal{P}$ and $\mathcal{V}$ halt. Otherwise, $\mathcal{P}$ and $\mathcal{V}$ 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., $b_i = 1$ if and only if $(r_i, i) \in \mathrm{R}_e$. Define $f_1, \ldots, f_N$ such at $f_i = r_i$ 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 $\mathcal{P}$ commits blindly to $b_e(x), f_e(x)$ by sampling the blinding factors $\rho_{b_e}, \rho_{f_e} \xleftarrow{\$} \mathbb{F}$ and computing $\mathsf{cm}_{b_e,1} := g_1^{\widehat{b}_e(\tau)}$, $\mathsf{cm}_{b_e,2} := g_2^{\widehat{b}_e(\tau)}$, and $\mathsf{cm}_{f_e} := g_1^{\widehat{f}_e(\tau)}$ where $\widehat{b}_e(x) := b_e(x) + \rho_{b_e} Z(x)$ and $\widehat{f}_e(x) := f_e(x) + \rho_{f_e} Z(x)$. That is, $\mathsf{cm}_{b_e,1}, \mathsf{cm}_{b_e,2}, \mathsf{cm}_{f_e}$ are *blinded commitments* to $b_e(x)$ and $f_e(x)$ respectively. Then $\mathcal{P}$ computes the received receipt count $k := \sum_{i=1}^n b_i$ and sends $(\mathsf{cm}_{b,1}, \mathsf{cm}_{b,2}, \mathsf{cm}_f, k)$ to $\mathcal{V}$.

Seed reveal. The verifier stores the received commitments locally and then reveals seed_e to $\mathcal{P}$. From 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 $(\pi_{1,e}, \pi_{2,e}, \pi_{3,e}, \pi_{D,e})$, and sends them to $\mathcal{V}$.

- **(C1) Consistency of $f_e(x)$ with $b_e(x)$ and $r_e(x)$.** First, $\mathcal{P}$ needs to show that f is consistent with r, i.e., $f_i = b_i \cdot r_i$ for all $i \in [n]$. This is implied by the existance of a polynomial $q_{1,e}(x)$ such that $\widehat{b}_e(x) \cdot r_e(x) - \widehat{f}_e(x) = q_{1,e}(x) \cdot Z(x)$. $\mathcal{P}$ sends the proof $\pi_{1,e} = g_1^{q_{1,e}(\tau)}$ to $\mathcal{V}$, who verifies by $e(\pi_{1,e}, g_2^{Z(\tau)}) \cdot e(\mathsf{cm}_{f_e}, g_2) \stackrel{?}{=} e(\mathsf{cm}_{b_e,1}, g_2^{r_e(\tau)})$.
- **(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 $\widehat{b}_e(x)(1 - b_e(x)) = Z(x) q_{2,e}(x)$. 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 $\mathsf{cm}_{b,1}, \mathsf{cm}_{b,2}$ are consistent.
- **(C3) Cardinality.** Next, $\mathcal{P}$ shows that there are exactly k_e non-zero elements among the b_is via univariate sum-check [10]. $\mathcal{P}$ constructs polynomials $q_{3,e}$ and $q_{4,e}$ with $\deg(q_{4,e}) \le n - 2$, such that $\widehat{b}_e(x) = q_{3,e}(x) Z(x) +$

$xq_{4,e}(x) + k_e \cdot n^{-1}$. $\mathcal{P}$ sends the proofs $\pi_{3,e} = g_1^{q_{3,e}(\tau)}, \pi_{4,e} = g_1^{q_{4,e}(\tau)}, \pi_{5,e} = h^{q_{5,e}(\tau)}$ to $\mathcal{V}$, who verifies by checking $e(\pi_{3,e}, g_2^{Z(\tau)}) \cdot e(\pi_{4,e}, g_2^{\tau}) \cdot e(g_1^{k_e \cdot n^{-1}}, g_2) \stackrel{?}{=} e(\mathsf{cm}_{b_e,1}, g_2)$ and $e(\pi_{4,e}, h) \stackrel{?}{=} e(g_1, \pi_{5,e})$. The first equation binds $\widehat{b}_e(x)$ to k_e through univariate sumcheck; the second certifies that $\deg(q_4) \leq n-2$.

- **(D) Disjointness proof.** To make sure no users who have already been counted in previous epochs are double-counted in this epoch, $\mathcal{P}$ computes a disjointness proof showing that $\tilde{b}_e$ is not overlapping with $\tilde{b}_{e-1}(x)$ when evaluated over Ω. Specifically, $\mathcal{P}$ computes $q_{D,e}(x)$ such that $\widehat{b}_{e-1}(x)\tilde{b}_e(x) = Z(x)q_{D,e}(x)$. The prover sends the proof $\pi_{D,e} := g_1^{q_{D,e}(\tau)}$ to $\mathcal{V}$, which is verified with $e(\pi_D, g_2^{Z(\tau)}) \stackrel{?}{=} e(\mathsf{cm}_{b_e}, \tilde{\mathsf{cm}}_{e-1})$. If there exists a user $\mathcal{U}_i$ who is counted twice by $\mathcal{P}$ in epoch e and in previous epochs, we have $\widehat{b}_{e-1}(\omega^{i-1}) = 1$ and $\tilde{b}_e(\omega^{i-1}) = 1$, and $\widehat{b}_{e-1}(x)\tilde{b}_e(x)$ is not divisible by $Z(x)$.

Verification and updates. $\mathcal{V}$ accepts the count k 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 $\tilde{\mathsf{cm}}_e := \tilde{\mathsf{cm}}_{e-1} \cdot \mathsf{cm}_{b_e}$ and $k := k + k_e$, and 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 $O(n)$ group operations. 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.

Accuracy. P-VAR enables the prover to prove the exact number of spent receipts. Therefore, P-VAR is 1-accurate, per the definition of accuracy in §3.

5.2. Security

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 the audit proof does not reveal anything – beyond the count – to a semi-honest verifier.

Theorem 2. *The P-VAR protocol presented in Figs. 3 and 4 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.

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 $(sk, vk) \leftarrow \mathsf{DS.Gen}(1^\lambda)$. $\mathcal{S}$ initializes a set of corrupted users who have spent their receipts, R'.

Issue. During the issue procedure, for all $\mathcal{U}_i \in U_{\mathcal{C}}$, $\mathcal{S}$ sends $r_i := \mathsf{PRF}(\mathsf{seed}_e; i)$ and the corresponding signature σ_i under vk to $\mathcal{A}$ for all $e \in [e_{\max}]$.

Spend. During the spend procedure, for all $\mathcal{U} \in U_{\mathcal{C}}$, $\mathcal{S}$ verifies the spent receipt (if any). If the receipt verifies and $\mathcal{U} \notin R$, $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{Spend}, \mathcal{U})$ to $\mathcal{F}_{\mathsf{VAR}}$ and updates $R' := R' \cup \{\mathcal{U}\}$.

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

We argue that the receipt verifies if and only if the r_i and σ_i are exactly what $\mathcal{S}$ distributes to a malicious user $\mathcal{U}_i$. This is because $\sigma_i = \mathsf{DS.Sign}(sk, i \| r_i \| e)$, and by the unforgeability of the DS scheme, σ_i will verify if and only if (r_i, σ_i) is what is sent by $\mathcal{S}$ to $\mathcal{U}_i$. Therefore, R is the same in the ideal world and in the real world. Since $\mathcal{P}$ and $\mathcal{V}$ are honest, we have that the output of honest parties is the same in both worlds. Finally, since $\mathcal{S}$ exactly performs the issuance procedure of P-VAR, and using the security of PRFs, the view of $\mathcal{A}$ is the same in both worlds for the r_i and σ_i that each corrupted user receives.

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 $\mathcal{V}$. $\mathcal{S}$ initializes a set of honest users who spent their receipts $R_{\mathcal{H}} = \emptyset$, and a set of corrupted users who spent their receipts $R' = \emptyset$. Finally, $\mathcal{S}$ initializes a variable $e = 1$, denoting the current epoch.

Issue. Upon receiving $(\mathsf{sid}, \textsc{Issue}, T, e_{\max})$ from $\mathcal{F}_{\mathsf{VAR}}$, for each $e \in [e_{\max}]$, $\mathcal{S}$ samples a random seed seed_e and stores it, and for all $\mathcal{U}_i \in U_{\mathcal{C}}$, $\mathcal{S}$ sends $r_i := \mathsf{PRF}(\mathsf{seed}_e; i)$ and the corresponding signature σ_i under vk to $\mathcal{A}$.

Spend. Upon receiving $(\mathsf{sid}, \textsc{Spend}, \mathcal{U}_i)$ from $\mathcal{F}_{\mathsf{VAR}}$, $\mathcal{S}$ computes $r_i = \mathsf{PRF}(\mathsf{seed}_e; i)$ and $\sigma_i = \mathsf{DS.Sign}(sk_{\mathcal{V}}, i \| r_i \| e)$, and sends (r_i, σ_i) to $\mathcal{A}$ as if it came from $\mathcal{U}_i$. $\mathcal{S}$ updates $R_{\mathcal{H}} := R_{\mathcal{H}} \cup \{\mathcal{U}_i\}$.

Audit. Upon receiving message $(\mathsf{sid}, \textsc{AuditReq})$ from $\mathcal{F}_{\mathsf{VAR}}$, $\mathcal{S}$ sends an audit request to $\mathcal{A}$ as if it came from $\mathcal{V}$. $\mathcal{S}$ runs the audit procedure as the verifier with $\mathcal{A}$. If $e \leq e_{\max}$ and the verification passes, and k_e is output of the simulated verifier, $\mathcal{S}$ updates $k := k + k_e$ sends $(\mathsf{sid}, \textsc{AuditResp}, 1, k)$ to $\mathcal{F}_{\mathsf{VAR}}$; otherwise, $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{AuditResp}, 0, k)$ to $\mathcal{F}_{\mathsf{VAR}}$, where k is whatever message $\mathcal{A}$ sends in step 2 in Fig. 4 (0 if no message/invalid message is received). If the proof verifies, we discuss two cases. In the first case, if $k > |R_{\mathcal{H}}| + |R'|$, $\mathcal{S}$ selects $k - (|R_{\mathcal{H}}| + |R'|)$ users in $U_{\mathcal{C}} \setminus R'$ as R''. For each of such $\mathcal{U}_i \in R''$, $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{SpendExtra}, \mathcal{U}_i)$ to $\mathcal{F}_{\mathsf{VAR}}$. Then $\mathcal{S}$ updates $R' := R' \cup R''$. In the second case, i.e., $k \leq |R_{\mathcal{H}}| + |R'|$, $\mathcal{S}$ does nothing. At the end, $\mathcal{S}$ increments e by 1, and updates $\tilde{\mathsf{cm}}_e$ accordingly.

Now we argue that the view of $\mathcal{A}$ is the same in the ideal world and in the real world. During Issue, for each $\mathcal{U}_i \in U_\mathcal{C}$, using security of PRFs and DS, the view of $\mathcal{A}$ of the corrupted users is the same in both worlds for r_i and σ_i. During Spend, using security of PRFs and DS, the view of $\mathcal{A}$ is the same in both worlds for r_i and σ_i of the received receipts from honest users by the corrupted $\mathcal{P}$. During Audit, since seed is uniformly randomly sampled, the view of $\mathcal{A}$ is the same in both worlds.

Finally, we argue that the output of $\mathcal{V}$ is the same in the real world and in the ideal world. In the ideal world, the output of $\mathcal{V}$ is $(\mathsf{sid}, \textsc{AuditOK}, k)$ if and only if the output count k_e of $\mathcal{P}$ in this epoch is not greater than the total number of remaining corrupted users and the number of honest users who spent their receipts in this epoch, and $b = 1$. It suffices to show that in no case can $\mathcal{A}$ interact with $\mathcal{S}$ such that $\mathcal{S}$ outputs $(\mathsf{sid}, \textsc{AuditResp}, 1, k)$ and violates this predicate. Assuming the hardness of q-SDH, in the AGM model, by Theorem 5.1 in the work of Das et al. [28], if $\mathcal{V}$ accepts the proof, except with negligible probability we must have (1) $\widehat{b}_e(x)(1-\widehat{b}_e(x)) = Z(x) \cdot q_{2,e}(x)$ for some polynomial $q_{2,e}(x)$, (2) $\widehat{f}_e(x) + q_{1,e}(x)Z(x) = \widehat{b}_e(x)r_e(x)$ for some polynomial $q_{1,e}(x)$, (3) $\widehat{b}_e(x) = q_{3,e}(x)Z(x) + q_{4,e}(x)x + k_e \cdot n^{-1}$ for some polynomial $q_{3,e}(x)$ and $q_{4,e}(x)$, (4) the degree of $q_{4,e}$ is at most $n-2$, and (5) $\widehat{b}_{e-1}(x)\widehat{b}_e(x) = Z(x)q_{D,e}(x)$ for some polynomial $q_{D,e}(x)$. (3) and (4) imply that $\sum_{a\in\Omega}\widehat{b}(a) = k_e$ according to the correctness of the univariate sumcheck [10]. (1) implies that the vector $\boldsymbol{b} := (b_1, \ldots, b_n)$ has all entries being 0 or 1. Then (2) implies that the vector $\boldsymbol{f} := (f_1, \ldots, f_n)$ satisfies $\boldsymbol{b} \circ \boldsymbol{f} = \boldsymbol{r}_e := (r_1, \ldots, r_n)$. Since r_i are sampled from $\mathbb{F}$, only with negligible probability $\mathcal{A}$ can have $f_i = r_i$ without receiving a receipt from $\mathcal{U}_i$ from $\mathcal{S}$. Therefore, we must have except with negligible probability that $\mathcal{A}$ cannot interact with $\mathcal{S}$ such that $\mathcal{S}$ outputs $(\mathsf{sid}, \textsc{AuditResp}, 1, k)$ with $k > |R|$.

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 $(\mathsf{sid}, \textsc{Init}, U_\mathcal{C}, e_{\max})$ to $\mathcal{F}_{\mathsf{VAR}}$. $\mathcal{S}$ initializes a set of users who spent their receipts $R = \emptyset$, and an epoch number $e = 1$. $\mathcal{S}$ keeps track of $s_{\tilde{b}}$, initialized to be 0.

Issue. Upon message $(\mathsf{sid}, \textsc{Issue}, T, e_{\max})$ from $\mathcal{F}_{\mathsf{VAR}}$, $\mathcal{S}$ gets from $\mathcal{A}$ the seed seed_e for all $e \in [e_{\max}]$.

Spend. For $\mathcal{U} \in U_\mathcal{C}$, $\mathcal{S}$ verifies the spent receipt (if any). If the receipt verifies, $\mathcal{S}$ sends $(\mathsf{sid}, \textsc{Spend}, \mathcal{U})$ to $\mathcal{F}_{\mathsf{VAR}}$.

Audit. Upon message $(\mathsf{sid}, \textsc{AuditOK}, k)$ from $\mathcal{F}_{\mathsf{VAR}}$, $\mathcal{S}$ simulates the audit procedure. If $e > e_{\max}$, $\mathcal{S}$ does nothing and halts. Otherwise, $\mathcal{S}$ gets k_e, the claimed count in this epoch. $\mathcal{S}$ selects a set of random users U_e of size k_e that does not overlap with the previous epoch's simulated users, and proceeds with the audit protocol with $\mathcal{A}$ as the prover.

Define the masked scalars $s_e := \widehat{b}_e(\tau), t_e := \widehat{f}_e(\tau)$. Since the verifier controlled by $\mathcal{A}$ is semi-honest, the outputs of honest parties are the same in the ideal world and in the real world. We now argue that the view of $\mathcal{A}$ is the same in both worlds. Because $\rho_{b,e}$ and $\rho_{f,e}$ are uniform and $Z(\tau) \neq 0$, both s_e, t_e are uniform in $\mathbb{F}$ and independent of the underlying witness (b_e, f_e). The commitments $(\mathsf{cm}_{b_e,1}, \mathsf{cm}_{b_e,2}, \mathsf{cm}_{f_e})$ are therefore distributed as $(g_1^{s_e}, g_2^{s_e}, g_1^{t_e})$, independent of the support of b_e.

Moreover, each quotient proof is a commitment to a polynomial whose evaluation at τ is a deterministic function of (s_e, t_e) and public values (and of the accumulator value $\tilde{b}_{e-1}(\tau)$ for disjointness). More specifically, (C1) enforces $\widehat{b}_e(\tau)r_e(\tau) - \widehat{f}_e(\tau) = q_{1,e}(\tau)Z(\tau)$, so $q_{1,e}(\tau)$ is determined by (s_e, t_e) and public $r_e(\tau)$; (C2) enforces $\widehat{b}_e(\tau)(1-\widehat{b}_e(\tau)) = q_{2,e}(\tau)Z(\tau)$, so $q_{2,e}(\tau)$ is determined by s_e; (C3) and the low-degree certificate constrain $(q_{3,e}(\tau), q_{4,e}(\tau))$ only through the linear relation $s_e = q_{3,e}(\tau)Z(\tau) + q_{4,e}(\tau)\tau + k_e n^{-1}$; since s_e is uniform, the resulting group elements leak no additional information about b_e's support; (D) depends on $\tilde{b}_{e-1}(\tau)$, which is uniform by induction because $\tilde{b}_{e-1}(\tau)$ is a sum of independent uniforms s_j from earlier epochs. Therefore, the joint distribution of all prover messages in epoch e depends only on (k_e, seed_e) and fresh uniform masks, not on which indices are selected. □

5.3. Supporting Dynamic User Sets

In §5.1, the runtime of $\mathcal{P}$ and $\mathcal{V}$ depends on the number of users determined at setup, instead of the actual number of possible users during audits, which could be way smaller. This is unnecessarily wasteful since in many applications, the number of spent receipts grows gradually. It would be ideal if the efficiency of the audit procedure depended on the number of actual users. We call this feature "dynamic user sets".

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. We assume both parties know U_e. 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 of degree N_e, rather than N, while preserving the same soundness and privacy guarantees. For example, if the system is provisioned for $N = 10^6$ users but early epochs have only 1,000 active users, auditing using 1,000-degree polynomials yields roughly a $10^3\times$ reduction in computation.

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

is, we need a sequence of points $(\omega_1, \ldots, \omega_N) \subset \mathbb{F}$ and a series of $\{N^{(i)}\}$ such that prefixes $\Omega^{(i)} = \{\omega_1, \ldots, \omega_{N^{(i)}}\}$ of length $N^{(i)}$ form a multiplicative subgroup of size $N^{(i)}$. In epoch e, we use the *smallest* $\Omega^{(i)}$ with size larger than N_e to perform all polynomial operations. Another property that follows from the nested structure is that if a user $\mathcal{U}_i$ (assigned to interpolation point ω_i) is counted in any epoch, her point ω_i is included in the domains of all future epochs.

We show one way to construct such sets. 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 $2^{32}|p-1$, 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: $\Omega^{(2)} = \{1, \omega^{2^{31}}\}$, $\Omega^{(3)} = \Omega^{(2)} \cup \{\omega^{2^{30}}, \omega^{3\cdot 2^{30}}\}$, $\Omega^{(4)} = \Omega^{(4)} \cup \{\omega^{2^{29}}, \omega^{3\cdot 2^{29}}, \omega^{5\cdot 2^{29}}, \omega^{7\cdot 2^{29}}\}$, 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.

Now we are ready to present the protocol.

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 $\Omega^{(i)} \subset \Omega^{(i+1)}$. This ensures the cardinality identity in **(C3)** holds over $Z_{\Omega^{(i)}}(x)$ and enables FFT interpolation on every $\Omega^{(i)}$.)

In addition to the standard KZG SRS vectors in $\mathbb{G}_1$ and $\mathbb{G}_2$ for degree N, the CRS publishes, for each $i \in [\ell]$, a short $\mathbb{G}_2$ vector $\boldsymbol{h}^{(i)} := \left(h^{(i)}, (h^{(i)})^{\tau}, \ldots, (h^{(i)})^{\tau^{N^{(i)}-2}}\right)$, with $h^{(i)} \xleftarrow{\$} \mathbb{G}_2$, to certify degree bounds relative to $\Omega^{(i)}$ when needed in the audit procedure. Similar to the Setup procedure in Fig. 3, $\mathcal{P}$ initializes $\tilde{b}_0(x) := 0$, $\mathcal{V}$ initializes $\tilde{\mathsf{cm}}_0 := g_1^0$, and both $\mathcal{P}$ and $\mathcal{V}$ set epoch number $e = 1$.

Issuance. When a new epoch with epoch number e starts, $\mathcal{V}$ issues new receipts to the users in set U_e. The interpolation domain to use 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 $r_i \leftarrow \mathsf{PRF}(\mathsf{seed}_e; i)$ and issues a receipt $\mathsf{rx}_{i,e} = (r_i, e, \sigma_i)$ to $\mathcal{U}_i$ where $\sigma_i = \mathsf{DS.Sign}(sk, i \| r_i \| e)$. Let $r_e(x)$ be a polynomial of degree $(N_e - 1)$ with $r_e(\omega_{i-1}) = \mathsf{PRF}(\mathsf{seed}_e; i)$ for $i \in [N_e]$. $\mathcal{V}$ computes $\mathsf{cm}_{r_e} := g_1^{r_e(\tau)} \in \mathbb{G}_1$. $\mathcal{P}$ initializes the received receipt set for this epoch to be $\mathrm{R}_e := \emptyset$.

Spending. Identical to the one in §5.1.

Audit. The audit procedure for epoch e starts when $\mathcal{V}$ sends an audit request to $\mathcal{P}$. $\mathcal{P}$ computes $b_e(x)$, $\widehat{b}_e(x)$, and $f_e(x)$ with R_e in the same way as step 2 in Fig. 4, with the interpolation set Ω_e replacing Ω, and 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-C3) as step 4 in Fig. 4, which are verified in the same way.

Setup$(1^\lambda, N)$ Runs all Setup steps in Fig. 3, 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 $\mathbb{F}$ and $N^{(i)} < N^{(i+1)}$ for $i \leq \ell$.
- Let $h^{(1)}, h^{(2)}, \ldots, h^{(\ell)}$ be uniformly random generators of $\mathbb{G}_2$. Compute and publish $\boldsymbol{h}^{(i)} := (h^{(i)}, \ldots, (h^{(i)})^{\tau^{N^{(i)}-2}})$ 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 this epoch. Note that if a user has received a receipt in epoch $(t-1)$, they must be issued receipt in epoch e. Let $\Omega_e := \Omega^{(i)}$ and $N_e := |\Omega^{(i)}|$ such that i is the smallest index with $N^{(i)} \geq N_e$. Note that $\Omega_e = \{\omega_1, \ldots, \omega_{N_e}\}$. If $\Omega_e = \Omega_{e-1}$, the rest of the protocol is the same as Fig. 3; otherwise, let $\Delta_e := \Omega_e \setminus \Omega_{e-1}$.
2) $\mathcal{V}$ samples seed_e. For each user $i \in [n_t]$:
   - $\mathcal{V}$ 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 $\mathsf{rx}_{i,e} := (r_i, e, \sigma_i)$ 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 $\mathsf{cm}_{r_e} := g_1^{r_e(\tau)}$.
4) $\mathcal{P}$ initializes the received receipt set $\mathrm{R}_e := \emptyset$.

Spend (epoch e) Identical to the Spend procedure in Fig. 3.

Audit (epoch e)

1) $\mathcal{V}$ sends an audit request to $\mathcal{P}$.
2) *Receipt commitments.* $\mathcal{P}$ computes $b_e(x), \widehat{b}_e(x), f_e(x)$ with R_e as step 2 in Fig. 4, with the interpolation points being Ω_e instead of Ω. $\mathcal{P}$ then 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 to $\mathcal{P}$. $\mathcal{P}$ derives r_i and interpolates $r_e(x)$ over Ω_e.
4) *Correctness proofs.* $\mathcal{P}$ runs **(C1)**, **(C2)** and **(C3)** as step 4 in Fig. 3.
5) *Extension proofs.*
   - **(E) Extension**: $\mathcal{P}$ interpolates $c_e(x)$ over Ω_t such that $c_e(x)|_{x \in \Omega_{e-1}} = \tilde{b}_e(x)$, and $c(x)|_{x \in \Delta_e} = 0$. The prover computes $q_L(x), q_R(x)$ such that $c(x) - \tilde{b}_e(x) = Z_{\Omega_{e-1}}(x) q_L(x)$ and $c(x) = Z_{\Delta_e}(x) q_R(x)$. Then $\mathcal{P}$ sends $\mathsf{cm}_c = g_1^{c(\tau)}$, $\pi_L := g_1^{q_L(\tau)}$, and $\pi_R := g_1^{q_R(\tau)}$ to $\mathcal{V}$.
6) *Disjointness proofs.* $\mathcal{P}$ runs **(D)** with $c(x)$ replacing $\tilde{b}_{e-1}$ as in Fig. 4.
7) *Verifier checks.* $\mathcal{V}$ accepts the claimed count k in this epoch if and only if **(C1)**, **(C2)**, **(C3)**, and **(D)** pass, and the following check passes:
   - **(E) Extension**: $e(\pi_L, g_2^{Z_{\Omega_{e-1}}(\tau)}) \stackrel{?}{=} e(\mathsf{cm}_c / \tilde{\mathsf{cm}}_{e-1}, g_2)$ and $e(\pi_R, g_2^{Z_{\Delta_e}(\tau)}) \stackrel{?}{=} e(\mathsf{cm}_c, g_2)$.
8) *Updates.* $\mathcal{P}$ updates $\tilde{b}_e(x) := c_e(x) + \widehat{b}_e(x)$. $\mathcal{V}$ updates $\tilde{\mathsf{cm}}_e := \mathsf{cm}_{c_e} \cdot \mathsf{cm}_{b_e}$ and $k := k + k_e$. They both update the epoch number $e := e + 1$.

Figure 5: P-VAR that supports a dynamic user set.

The challenge arises with the disjointness proof (D). The problem is that $\tilde{b}_{e-1}(x)$ is defined over Ω_{e-1}, so it and does not necessarily evaluates to 0 over Δ_e. Before running the disjointness proofs the prover needs extend $\tilde{b}_{e-1}(x)$ from Ω_{e-1} to Ω_e, with an extension proof: The prover interpolates $c_e(x)$ of degree N_e such that $c_e(x)|_{x\in\Omega_{e-1}} = \tilde{b}_{e-1}$ and $c_e(x)|_{x\in\Delta_e} = 0$, and commits $\mathsf{cm}_{c_e} = g_1^{c_e(\tau)}$. $\mathcal{P}$ needs to prove 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 $\mathcal{V}$, which are checked with $e(g^{q_L(\tau)}, g^{Z_{\Omega_{e-1}}(\tau)}) \stackrel{?}{=} e(\mathsf{cm}_{c_e}/\tilde{\mathsf{cm}}_{e-1}, g)$ and $e(g^{q_R(\tau)}, g^{Z_{\Delta_e}(\tau)}) \stackrel{?}{=} e(\mathsf{cm}_c, g)$. This step certify that the extended polynomial $c_e(x)$ agrees with $\tilde{b}_{e-1}(x)$ on Ω_{e-1} and vanishes on Δ_e.

Then $\mathcal{P}$ runs 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 $\tilde{b}_{e-1}(x)$. Afterwards, the verifier verifies the proofs. If all checks pass, they update the recorded values accordingly. $\mathcal{P}$ updates $\tilde{b}_e(x) := c_t(x) + \hat{b}_e(x)$, and $\mathcal{V}$ updates $\tilde{\mathsf{cm}}_e := \mathsf{cm}_{c_e} \cdot \mathsf{cm}_{b_e}$ 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 constant-size proofs) for $\mathcal{V}$. In comparison, if the domain is fixed at 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 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 $\frac{N}{N_e} \approx \frac{10^6}{10^3} \approx 1,000\times$. Over $T = 12$ epochs, suppose N_e doubles each epoch from 10^3 up to 10^6, then with dynamic user sets, the total prover time is $\sum_{t=1}^{T} N_e \approx 2N$ group operations and $\sum_{t=1}^{T} N_e \log N_e \approx 2N \log N$ field operations; without dynamic user sets, the total prover time is $T \cdot N$ group operations and $T \cdot N \log N$ field operations, so the total prover time is reduced by at least $6\times$.

6. Applications

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

6.1. Auditing Social Media Platforms

Content creators rely on social media platforms such as Facebook, TikTok, and YouTube for their livelihood [30]. These large platforms, however, offer little verifiability and transparency about how contents are prioritized and monetized, leading to widespread concerns [11], [12], [13], [14], [15], [16]. Recent regulations, such as the EU's Digital Services Act, call for greater scrutiny and auditing, but the specific technical means are unclear. VAR can potentially fill a gap by offering a means to audit content delivery on social media platforms.

One area where VAR can be directly applied is to audit the billing of paid boosts. Content creators pay to increase the exposure of their content, through services such as Boost Posts on Instagram, Promote on TikTok, Promoted Posts in X [31], [32], [33]. Self-reported viewship is problematic as the platforms stand to gain by inflating. Another application of VAR is to detect *stealth suppression* attacks, a strengthened form of so-called "shadow banning," in which the platform quietly limits the delivery of a creator's content while keeping the user interface, including reported views and engagement statistics, looking normal.

Mapping to VAR. VAR can be applied as shown in Fig. 6. The content creator acts as the Verifier $\mathcal{V}$, the social platform plays the role of the Prover $\mathcal{P}$, and the viewers form the user set U. The creator first issues (encrypted) content-specific receipts to eligible viewers (e.g., followers) by embedding them in the content they upload to $\mathcal{P}$. When the platform delivers the content to a viewer, the viewer retrieves and spends the content-specific receipt by decrypting and revealing it to the platform. The platform collects receipts to be used at subsequent audits. At any point in time after issuance, the creator can make an audit request to the platform. The platform and the creator execute the VAR audit procedure at the end of which, the creator should be convinced of the count claimed by the platform.

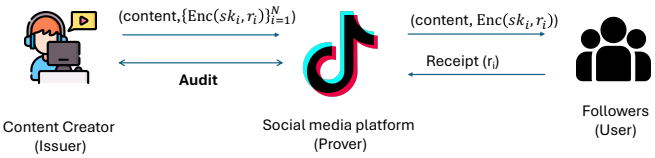


Figure 6: VAR for auditing social media platforms.

A few points to discuss. First, implementing VAR requires modifying the client software (e.g., in the form of a modified app or a browser extension). The incentives for development efforts may come from their potential to improve the trustworthiness of the platforms. Second, VAR requires the content creator and its followers to have well-known public keys. In emergent social media platforms with native support for DID [34], such as BlueSky [35] (and ATProtocol variants) and Web3 social protocols (Lens [36], Farcaster [37]), each user has a long-term public key stored in their DID profile. For legacy platforms using platform-specific identities, such as YouTube, binding public keys to a centralized identity is not directly supported. One possibility is for users to send their public key in a direct message to the content creator, or publish it at a public bulletin board accompanied by a proof of account ownership using an oracle protocol [38], [39], [40].

Finally, VAR does not prevent inflation due to bots created by users or the platform (which is permitted by our model, where we view such malicious users as being part of the malicious platform). However, a content creator can impose further eligibility conditions to weed out bots,

such as ignoring the views from new or inactive accounts. The point is that the verifier of VAR is also the issuer, so they can choose the universe of users.

6.2. 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) [41]. Workforce agencies under programs such as WIOA [42] 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 [43].

Mapping to VAR. In all of these scenarios, the issuer chooses the 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 and makes them available to each user in U. When users consume 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.

6.3. Read Receipt for Broadcast Messages

Motivation. 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.

7. Implementation and Evaluation

In this section, we report on implementation details and the performance evaluation of P-VAR and S-VAR.

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 [44], [45] 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 efficiently, we use the *barycentric Lagrange formula* [8]. Specifically, given n evaluation points $\{(x_i, f(x_i))\}_{i\in[n]}$, we reconstruct at $x = 0$:

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

where $w_i = 1/\prod_{j\neq i}(x_j - x_i)$ are the *barycentric weights*. With $M(x) = \prod_{i=1}^{n}(x - x_i)$, the weights can be written as $w_i = w(x_i)$, where $w(x) = \frac{1}{M'(x_i)}$. To compute all $M'(x_i)$ efficiently, we build a *subproduct tree* [9] 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 [9] 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 [46] 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] and a scheme based on generic SNARK proofs.

Baseline 1: Predicate aggregate signatures. As described in §3.3, the PAS scheme of Qiu et al. [5] is a close enough baseline as it achieves all properties but deniability. Since the original paper did not provide an implementation, we implement the core inner-product arguments in Rust 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 introduce this baseline solution in §3.3 and describe it 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 [18] `v1.0.0-beta.15`, an open-source domain-specific language (DSL) for writing SNARK circuits. We use Aztec's Barretenberg [47], Noir's default 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 benchmark provides a lower bound on the proving and verification time for this baseline.

Choices of baselines. We consider alternative baselines such as hinTS [29]. While the proof time of hinTS is acceptable, it requires each user to run a preprocessing procedure 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. All runtime measurements for a particular operation are averaged over 10 runs, after 3 warmup runs. We also plot standard deviations, but they may be too small to be legible.

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 configuration 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.

Mapping server configurations to applications. Recall that in our applications, a significant portion of the computation is done on the prover side. The content creator application (§6.1) might justify running the prover on the large server as the prover already maintains significant infrastructure to serve large user bases. In contrast, the broadcast receipts (§6.3) and the service reimbursement (§6.2) applications are expected to generally have fewer users and presumably less well-resourced provers, for whom we use the small servers. For issuance and verification (which occurs within the same party), corporations and governments wishing to audit reimbursements may use a large issuer. In contrast, the moderators who issue receipts to receive read receipts and social media content creators might use constrained hardware, represented by our small server.

7.3. Issuance performance

We first report on the performance of the issuance procedure. Note that our baselines do not require explicit issuance (as they only need each user to hold a private/public key, which is assumed to be known ahead of time by the issuer or verifier), so we exclude them from the evaluation.

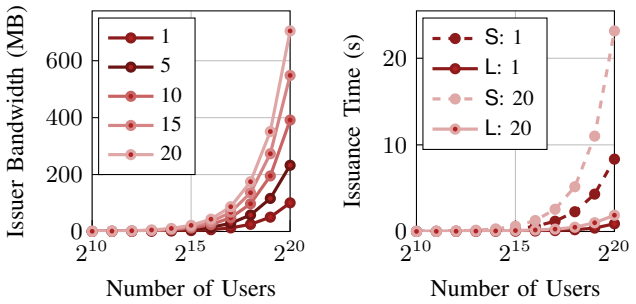
For our protocols, two metrics are of interest: the bandwidth consumption and the runtime of the issuer.

Bandwidth. The issuer bandwidth is the total amount of data sent by the issuer in supplying each user with receipts. Recall from §4, that for S-VAR, a receipt consists of a random value and a signature, regardless of the number of thresholds. On top of the n receipts, the S-VAR issuer also sends auxiliary data aux to the prover. In P-VAR instead, there are as many per-user receipts as there are supported epochs, and each receipt is also a random value and a signature.

Issuance time. The runtime of issuance is defined as the time taken to compute all receipts for all users. The issuance procedure for S-VAR primarily consists of FFT-based polynomial interpolation and polynomial evaluation. For P-VAR, it mainly consists of group operations. Our reported issuance times include the signing of receipts.

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 §4.4; that is, 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.

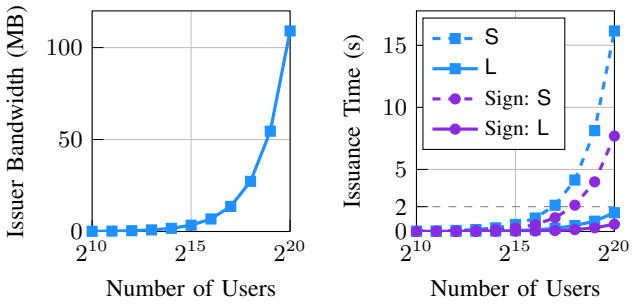
Evaluation results. The results are shown in Figs. 7 and 8.



(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 7: 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 8: 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. 8b also shows the signing time, which is part of issuance. L = `c7i.16xlarge`, S = `c7i.xlarge`.

S-VAR. Fig. 7a 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 7b 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 is concretely efficient; in the largest setting we benchmarked – one million users and 20 thresholds – it takes only 23.15 s (small VM) and 1.876 s (large VM). We found that EdDSA signing dominates the issuer runtime on the small VM – about 90% of total issuance time – whereas 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. 8a shows the issuer bandwidth of P-VAR for one epoch. Concretely, issuance in P-VAR consumes 0.106 MB in bandwidth for 2^{10} users and 109.05 MB for 2^{20} users.

Figure 8b shows the issuance time. For 2^{10} users, issuance takes 0.023 s on the small VM and 0.005 s on the large VM; for 2^{20} users, issuance takes 16.156 s on the small VM and 1.516 s on the large VM.

For reference, Fig. 8b shows the time it takes to produce n Ed25519 signatures, as a function of the number n of users. Signing contributes around 50% (small VM) and 33% (large VM) to the total issuance time. Concretely, producing 2^{10} signatures takes 0.008 s on the small VM and 0.001 s on the large VM, and producing 2^{20} signatures takes 7.68 s on the small VM and 0.58 s on the large VM. Recall that issuance in both S-VAR and P-VAR involves signing each user receipt individually. In S-VAR, the issuer signs once per user, regardless of the number of thresholds. Thus, the measured signing time in Fig. 8b applies to both P-VAR (for one epoch) and to S-VAR (for *any* number of thresholds).

7.3.1. Comparison between S-VAR and P-VAR assuming the same number of supported audits.

The above results evaluate S-VAR and P-VAR separately. The two schemes have different tradeoffs: S-VAR's overhead scales with the precision (the number of thresholds), while P-VAR's 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.

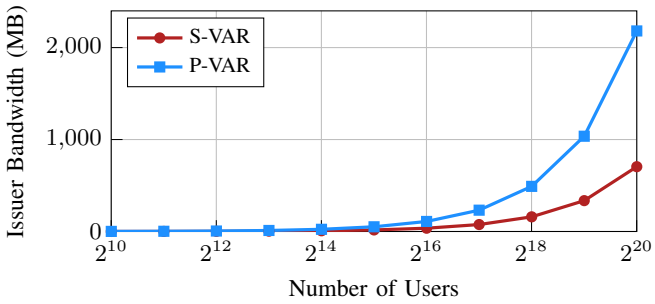
Bandwidth. Figure 9a shows the bandwidth of S-VAR and P-VAR as a function of the number n of users. Concretely,

- For 2^{10} users, issuance in S-VAR consumes 0.36 MB in bandwidth, and P-VAR consumes 1.06 MB.
- For 2^{20} users, issuance in S-VAR consumes 704.6 MB in bandwidth, and P-VAR consumes 2,181 MB.

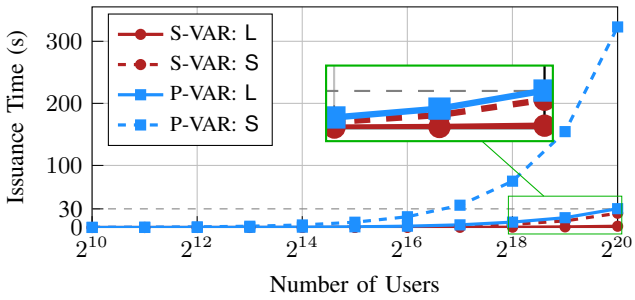
A trade-off emerges between the granularity of audits and the issuer's bandwidth. In particular, while S-VAR requires about $\frac{1}{3}$ of the issuer bandwidth consumed by P-VAR to support $\log n$ audits, P-VAR support exact audit counts, while S-VAR only supports auditing at powers-of-2 thresholds.

Issuance time. Figure 9b shows the issuance time of S-VAR and P-VAR on large and small servers. Concretely,

- For 2^{10} users, issuance in S-VAR takes 0.013 s, and P-VAR takes 0.235 s on the small VM.
- For 2^{20} users, issuance in S-VAR takes 2.02 s, and P-VAR takes 30.3 s on the large VM.



(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.

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, and there is roughly a $10\times$ speedup when running on L (which has 16x more cores than S). Finally, we observe the same tradeoff as in the bandwidth comparison: P-VAR requires approximately an order-of-magnitude higher issuance time but provides more fine-grained audits. Note that, despite being embarrassingly parallelizable, our algorithms cannot fully leverage a large number of cores; thus, the issuance time on L initially is only $4\times$ better than that on S; however, as the number of users increases, the ratio approaches $10\times$.

Storage footprint on issuer and users. At the end of the issuance procedure, in both schemes, 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 S-VAR and P-VAR. 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 kB; in S-VAR, the user bandwidth is independent of the number of audits and remains under 100 B.

7.4. Spend and Audit Performance

7.4.1. Spend. In the spend procedure of both S-VAR and P-VAR, each user simply sends his constant-sized receipt (96 B for S-VAR and 104 B for P-VAR); therefore, 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. In order to allow the best possible setting for our baselines, we only provide baseline benchmarks on our bigger VM (`c7i.16xlarge`). For all solutions we benchmark, the proving time at audit is independent of the number of spent receipts (n is the total number the system supports). We highlight some concrete numbers:

- For 2^{10} users, S-VAR takes 0.076 s, and P-VAR takes 0.15 s on the small VM.
- For 2^{20} users, S-VAR takes 34 s, and P-VAR takes 9.74 s on the large VM.

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 scales poorly and quickly becomes impractical. At the largest parameter size we tested, 2^{20}, the PAS solution's prover performs $30\times$ worse than the next-best construction S-VAR. Even on the smaller VM, S-VAR performs about $9\times$ better than the PAS solution 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 the small-parameter regime, P-VAR outperforms P-VAR across every machine and every 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 a fragmented and irregular subset, eliminating the ability to use FFTs and forcing interpolation to proceed in super-linear time.

Bandwidth. The bandwidth for both solutions is constant at audit time: 32 B in S-VAR and 1.1 kB in P-VAR.

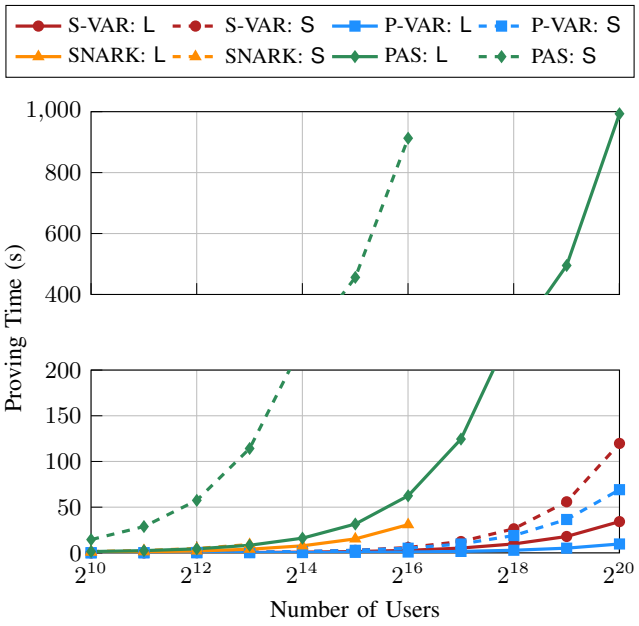


Figure 10: Prover time for S-VAR and P-VAR. L = `c7i.16xlarge`, S = `c7i.xlarge`. **Note the discontinuity in the y-axis, and the change of scale.**

Round complexity and latency. For each solution, the latency added by these proofs is just the proving time, together with the network latency of any interactions, and the proof generation time dominates. Per [48], the round trip latency for a packet over TCP is 150ms, and in the case of P-VAR the number of rounds for an audit is 2 and thus the latency is 300ms. In the case of S-VAR, we have a single round, so only 150ms latency. Both SNARK and PAS are non-interactive, i.e., one round.

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 B strings, taking negligible time (less than 100 µs on both VMs). The P-VAR verifier runs in about 19ms on the small VM and about 16ms 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 625ms. The SNARK baseline presented here causes verification time to increase linearly with the number of users, reaching 469ms on L, at 2^{16} users. Note that this time can be shortened to a constant, approximately 28ms by making the public keys a part of the witness but only at the expense of significantly higher proving time.

7.5. Summary

Our experiments show that across all metrics, both our VAR schemes outperform existing, more heavyweight solutions. At small scale (a thousand users), both schemes require less than 0.025 s for issuance, and less than 0.15 s for audit proving. Our schemes are also *concretely* fast at large scale (a million users); Issuance takes less than 2 s for either scheme, and for audit proving time, P-VAR requires less than 10 s and S-VAR requires less than 35 s.

Our results also give rise to interesting trade-offs between our solutions. As we have observed in §7.3, the issuer for S-VAR runs faster than the issuer for P-VAR when supporting the same number of audits. This trend is flipped in the proving stage, where P-VAR outperforms all other solutions on the respective machines. For a smaller number of users, the prover time benefit wrought by a bigger machine is not as readily apparent, but as the number of users grows, the ratio of the proving time on L to that on S approaches the difference in the number of cores for each solution. For smaller numbers of users, the difference between the two solutions is not as stark.

Thus, an application must select between P-VAR and S-VAR based on the expected number of users, the size of the issuer and the size of the prover. For instance, for an application like §6.2, we would recommend P-VAR, as it works well with a smaller prover but larger issuer.

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, enabling efficient verification. As we discuss in §3.3, plain aggregate signatures lack signer anonymity, deniability, and an efficient method to produce a proof of count, and thus do not help towards constructing 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 that individual signatures are valid and that a public predicate (e.g., a minimum number of distinct signers) is satisfied. PAS matches some efficiency and security goals of VAR, but its inner-product pairing arguments make it significantly expensive and thus intractable at our million-user scale; also, it does not achieve deniability, similar to all signature-based solutions discussed here. We therefore treat PAS as a conceptual strawman solution in §3.3.

More recent pairing-based threshold signature schemes [28], [29] offer succinct, weighted, and multi-threshold signing. In hinTS [29], each user publishes a "hint" derived from their secret key and the Lagrange basis polynomials, enabling the aggregator to efficiently prove the well-formedness of the aggregate public key, via polynomial commitments and generalized sumcheck-style argument. This signing and verification efficiency comes at the cost of $O(n)$ per-user group operations for hint generation and expensive preprocessing over all hints on the aggregator side. Among hinTS [29] and the close variant of Das et al. [28], hinTS is most relevant for us since it offers a blinded variant that hides the exact set of signers, and this

is why we use it as one of our baselines in §7. Finally, all three schemes [5], [28], [29] are signature-based and thus do not provide deniability, which is a core property in VAR.

Privacy-preserving aggregate statistics. One line of work (see, e.g., [49], [50], [51], [52]) 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 [53] 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.

Torres et al. [54] propose a voting-style protocol for privacy-preserving ad reporting at web scale; clients send encrypted one-hot ballots under a public key, the ad server homomorphically aggregates them, and a trusted tallying party decrypts only aggregate per-add counts. Acharya et al. [55] introduce non-interactive verifiable aggregation (NIVA), a new primitive for secure computation of aggregate statistics over a large population of clients. They construct an efficient NIVA protocol using inner-product functional encryption and fully linear PCPs to enforce NP validity predicates on each client input. 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. [56] build an anonymous reputation system where users can rate each product at most once; their system model, however, relies on a trusted group manager who controls group public keys and can always 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. [57] 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 [58] 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 [59] 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., [60], [61], [62]) 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., [63], [64], [65]) 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 [28], [29], [66] 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. [23], offer a more direct and lightweight approach. Kate et al. [7] formalized bottom-up secret sharing, first used by Baird et al. [67] 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) [68] and more efficient related primitives [69] 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. [69] relies on pairing-based cryptography similarly to our P-VAR construction, which in our case enables *exact* auditing.

9. Discussion

Streaming delivery of receipts. In our descriptions 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 in a way that receipts are relayed by the service provider in a streaming fashion, so that users and the issuers do not need to interact (beyond the initial key setup), and 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, the 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, it retrieves the necessary ciphertext from $\mathcal{P}$, decrypts it using sk_i, and spends it as usual.

The social media application (§6.1) 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 adverarial 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 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 inflating soundness, while limiting what an otherwise semi-honest $\mathcal{V}$ can infer about user spending.

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 issuance of receipts and their privacy-preserving aggregation into a compact proof of count. We formalized VAR's security goals via an ideal functionality, and presented two constructions: S-VAR, a protocol based on bottom-up secret sharing that enables tiered "fuzzy" audits with constant-size receipts independent of the number of thresholds, and P-VAR, a pairing-based protocol enabling exact audits with constant-time verification and an extension to dynamic user sets; we proved both schemes secure with respect to our ideal functionality. We implemented and benchmarked our protocols, showing that they scale to one million users, while existing solutions are either at least an order of magnitude slower in proving or do not scale to this regime. Finally, we showcased three diverse applications of VAR: auditing social media platforms, preventing fraudulent reimbursement claims, and detecting censorship in broadcast.

Acknowledgements

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. [Online]. Available: https://www.justice.gov/usao-sdny/pr/us-attorney-announces-129-million-settlement-door-submitting-fraudulent-cost-reports

[2] "Criminal Division | Case Summaries," Jun. 2024. [Online]. Available: https://www.justice.gov/criminal/criminal-fraud/health-care-fraud-unit/2024-national-hcf-case-summaries

[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. [Online]. Available: https://www.justice.gov/usao-az/pr/seven-charged-arizona-part-department-justices-2024-national-health-care-fraud

[4] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22*. 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] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979. [Online]. Available: https://dl.acm.org/doi/10.1145/359168.359176

[7] A. Kate, P. Mukherjee, H. Saleem, P. Sarkar, and B. Roberts, "ANARKey: A New Approach to (Socially) Recover Keys," 2025, publication info: Preprint. [Online]. Available: https://eprint.iacr.org/2025/551

[8] J.-P. Berrut and L. N. Trefethen, "Barycentric lagrange interpolation," *SIAM Review*, vol. 46, no. 3, pp. 501–517, 2004. [Online]. Available: https://doi.org/10.1137/S0036144502417715

[9] A. Borodin and R. Moenck, "Fast modular transforms," *Journal of Computer and System Sciences*, vol. 8, no. 3, pp. 366–386, 1974. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022000074800292

[10] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward, "Aurora: Transparent succinct arguments for r1cs," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2019, pp. 103–128.

[11] 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. [Online]. Available: https://www.nytimes.com/2018/04/03/us/youtube-shooting.html

[12] P. Mozur, "A former youtube insider says the company failed to curb extremism," *The New York Times*, June 2019. [Online]. Available: https://www.nytimes.com/2019/06/25/technology/youtube-extremism.html

[13] R. Molla, "Youtube's lgbtq community says its videos keep getting restricted or demonetized," *Vox*, October 2019. [Online]. Available: https://www.vox.com/culture/2019/10/10/20893258/youtube-lgbtq-censorship-demonetization-nerd-city-algorithm-report

[14] V. Staff, "Youtube settles three-year-long racial bias lawsuit from black and hispanic creators," *Verdict*, February 2025. [Online]. Available: https://www.verdict.co.uk/news/youtube-settles-three-year-long-racial-bias-lawsuit-from-black-and-hispanic-creators/

[15] 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. [Online]. Available: https://policyreview.info/articles/analysis/algorithmic-dance-youtubes-adpocalypse-and-gatekeeping-cultural-content-digital

[16] M. D. Staff, "Algorithms of favoritism: Power wins again," *Modern Diplomacy*, February 2025. [Online]. Available: https://moderndiplomacy.eu/2025/02/04/algorithms-of-favoritism-power-wins-again/

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

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

[19] 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. [Online]. Available: https://doi.org/10.1137/0217017

[20] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *International conference on the theory and application of cryptology and information security*. Springer, 2010, pp. 177–194.

[21] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," in *Proceedings of the 1st ACM Conference on Computer and Communications Security*, ser. CCS '93. New York, NY, USA: Association for Computing Machinery, 1993, p. 62–73. [Online]. Available: https://doi.org/10.1145/168588.168596

[22] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, ser. SP '14. USA: IEEE Computer Society, 2014, p. 459–474. [Online]. Available: https://doi.org/10.1109/SP.2014.36

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

[24] A. Kiayias, M. Osmanoglu, A. Russell, and Q. Tang, "Space Efficient Computational Multi-Secret Sharing and Its Applications," 2018, publication info: Preprint. MINOR revision. [Online]. Available: https://eprint.iacr.org/2018/1010

[25] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," Cryptology ePrint Archive, Paper 2000/067, 2000. [Online]. Available: https://eprint.iacr.org/2000/067

[26] 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.

[27] 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.

[28] 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 *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 356–370.

[29] 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.

[30] K. Cabello. The state of the creator economy – assessing the economic, cultural, and educational impact of youtube in the us in 2022 — oxford economics. [Online]. Available: https://www.oxfordeconomics.com/resource/youtube-us/

[31] "How to boost an instagram post," https://help.instagram.com/2090822074310288/, 2025, accessed: 2025-12-01.

[32] "Use promote to grow your tiktok audience," https://support.tiktok.com/en/using-tiktok/growing-your-audience/use-promote-to-grow-your-tiktok-audience, 2025, accessed: 2025-02-15.

[33] "Increase your reach on x," https://help.x.com/en/managing-your-account/increase-x-reach, 2025, accessed: 2025-02-15.

[34] Bluesky. (2023) The `did:plc` identifier method. [Online]. Available: https://atproto.com/specs/did

[35] ——, "AT protocol specification," 2023, https://atproto.com.

[36] L. Protocol. (2024) Lens protocol documentation. [Online]. Available: https://docs.lens.xyz/

[37] Farcaster. (2024) Farcaster protocol documentation. [Online]. Available: https://docs.farcaster.xyz/

[38] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town Crier: An Authenticated Data Feed for Smart Contracts," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 270–282. [Online]. Available: https://dl.acm.org/doi/10.1145/2976749.2978326

[39] F. Zhang, D. Maram, H. Malvai, S. Goldfeder, and A. Juels, "DECO: Liberating Web Data Using Decentralized Oracles for TLS," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 1919–1938. [Online]. Available: https://dl.acm.org/doi/10.1145/3372297.3417239

[40] D. Maram, H. Malvai, F. Zhang, N. Jean-Louis, A. Frolov, T. Kell, T. Lobban, C. Moy, A. Juels, and A. Miller, "CanDID: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability," in *42nd IEEE symposium on security and privacy, SP 2021, san francisco, CA, USA, 24-27 may 2021*. IEEE, 2021, pp. 1348–1366. [Online]. Available: https://doi.org/10.1109/SP40001.2021.00038

[41] Federal Transit Administration. (2016) Shared mobility faqs: Eligibility under FTA grant programs. Describes how FTA funds can reimburse transit agencies that contract with shared mobility or taxi operators, including voucher programs. Accessed: 2025-12-01. [Online]. Available: https://www.transit.dot.gov/regulations-and-guidance/shared-mobility-faqs-eligibility-under-fta-grant-programs

[42] Learn & Work Ecosystem Library. (2018) How does WIOA provide/pay for training? Explains that Individual Training Accounts (ITAs) under WIOA are used to pay Eligible Training Providers to train participants in approved programs of study. Accessed: 2025-12-01. [Online]. Available: https://learnworkecosystemlibrary.com/etp-1-pager/

[43] Connecticut Department of Social Services. (2013) Regulation concerning payment of behavioral health clinic services. Medicaid regulation specifying fee-schedule-based payment for individual, group, and family psychotherapy and other behavioral health clinic services. Accessed: 2025-12-01. [Online]. Available: https://portal.ct.gov/-/media/Departments-and-Agencies/DSS/Reimbursement-and-Certificate-of-Need/regulation_concerning_payment_of_behavioral_health_clinic_services.pdf

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

[45] 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. [Online]. Available: https://doi.org/10.1007/s13389-012-0027-1

[46] 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. [Online]. Available: https://doi.org/10.1145/322092.322099

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

[48] "Latency numbers everyone should know," https://static.googleusercontent.com/media/sre.google/en//static/pdf/rule-of-thumb-latency-numbers-letter.pdf, google Site Reliability Engineering, accessed 2025-12-04.

[49] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *14th USENIX symposium on networked systems design and implementation (NSDI 17)*, 2017, pp. 259–282.

[50] 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.

[51] 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 *Proceedings of the 26th symposium on operating systems principles*, 2017, pp. 441–459.

[52] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly) logarithmic overhead," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020, pp. 1253–1269.

[53] 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*. USENIX Association, 2023, pp. 4805–4822.

[54] M. Green, W. Ladd, and I. Miers, "A Protocol for Privately Reporting Ad Impressions at Scale," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. Vienna Austria: ACM, Oct. 2016, pp. 1591–1601. [Online]. Available: https://dl.acm.org/doi/10.1145/2976749.2978407

[55] 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. [Online]. Available: https://doi.org/10.56553/popets-2025-0171

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

[57] 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.

[58] A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda, "Privacy pass: Bypassing internet challenges anonymously," *Proceedings on Privacy Enhancing Technologies*, 2018.

[59] F. Benhamouda, M. Raykova, and K. Seth, "Anonymous counting tokens," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2023, pp. 245–278.

[60] B. Kreuter, T. Lepoint, M. Orrù, and M. Raykova, "Anonymous tokens with private metadata bit," in *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part I*, ser. Lecture Notes in Computer Science, D. Micciancio and T. Ristenpart, Eds., vol. 12170. Springer, 2020, pp. 308–336. [Online]. Available: https://doi.org/10.1007/978-3-030-56784-2_11

[61] 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.

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

[63] Y. Ji and K. Chalkias, "Generalized proof of liabilities," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds. ACM, 2021, pp. 3465–3486. [Online]. Available: https://doi.org/10.1145/3460120.3484802

[64] J. Xin, A. Haghighi, X. Tian, and D. Papadopoulos, "Notus: Dynamic proofs of liabilities from zero-knowledge RSA accumulators," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 1453–1470. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/xin

[65] F. Falzon, K. Elkhiyaoui, Y. Manevich, and A. D. Caro, "Short privacy-preserving proofs of liabilities," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, W. Meng, C. D. Jensen, C. Cremers, and E. Kirda, Eds. ACM, 2023, pp. 1805–1819. [Online]. Available: https://doi.org/10.1145/3576915.3616645

[66] 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.

[67] L. Baird, S. Garg, A. Jain, P. Mukherjee, R. Sinha, M. Wang, and Y. Zhang, " Threshold Signatures in the Multiverse ," in *2023 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2023, pp. 1454–1470. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.10179436

[68] 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.

[69] S. Garg, D. Kolonelos, G.-V. Policharla, and M. Wang, "Threshold Encryption with Silent Setup," in *Advances in Cryptology – CRYPTO 2024*, L. Reyzin and D. Stebila, Eds. Cham: Springer Nature Switzerland, 2024, pp. 352–386.

[70] 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.

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

Appendix A. Additional Preliminaries

In this section, we provide additional preliminaries.

Digital signatures. A digital signature scheme is a tuple of PPT algorithms $\mathsf{DS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$. On input a security parameter λ, Gen outputs a key pair (sk, vk), where sk is the secret signing key and vk is the public verification key. On input a message m and sk, the signing algorithm Sign outputs a signature σ; on input a verification key vk, a message m, and a signature σ, and the verification algorithm outputs a bit $b \in \{0,1\}$. Correctness requires that for all messages m and all $(sk, vk) \leftarrow \mathsf{Gen}(1^\lambda)$, we have $\mathsf{Verify}(vk, m, \mathsf{Sign}(sk, m)) = 1$. The standard security notion is *existential unforgeability under adaptive chosen-message attacks* (EUF-CMA) [19], which informally states that no efficient adversary, even with oracle access to $\mathsf{Sign}(sk, \cdot)$, can produce a valid signature on any new message it did not previously query.

zk-SNARK. An argument system for an NP relation R consists of a computationally bounded prover $\mathcal{P}$ and a verifier $\mathcal{V}$, where $\mathcal{P}$ convinces $\mathcal{V}$ that a witness w exists such that $(x, w) \in R$ for input x. A setup algorithm Gen produces public parameters. The triple $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ is a zero-knowledge argument (ZKA) of knowledge for R if it satisfies *completeness*, *knowledge soundness*, and *zero knowledge* (see [70], [71]). It is *succinct* if both the proof size and verifier's time are bounded by $\mathsf{poly}(\lambda, |x|, \log |R|)$, where $\log |R|$ is the size of the circuit computing R. In this paper, one of the baselines we compare our solutions against is based on zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs), i.e., ZKAs of knowledge satisfying all three properties above and are succinct.