Pethoa: Dynamic Proofs of Assets for Hidden Address Sets on Account-Based Blockchains

Yangrui Zhou, Dongliang Cai, Yuxuan Chen, Liang Zhang*, Haibin Kan**

Fudan University

Abstract. Proofs of solvency allow custodians to demonstrate that their assets cover their liabilities, but conventional Proofs of Assets (PoA) often require publicly disclosing reserve addresses, exposing financial activities and enabling long-term transaction tracing. While zero-knowledge proofs can hide reserve addresses and prove asset ownership and balances against a public blockchain state, generating proofs for evolving blockchain states remains computationally expensive, particularly for chains with proof-unfriendly state structures. We introduce Dynamic Private Proof of Assets (DPoA), a framework for efficiently updating proofs of assets over hidden sets of on-chain addresses. Our construction separates asset-proof generation into an expensive initialization phase and lightweight incremental updates. During initialization, the prover establishes ownership and state membership for the reserve addresses. Subsequently, for each epoch, the prover locally updates its proof state based on newly finalized blocks and generates a zero-knowledge proof to demonstrate the correctness of the update, avoiding the need to regenerate the entire asset proof from scratch. Experimental results demonstrate the efficiency of our approach: generating a conventional static proof for thousands of addresses takes several minutes on an NVIDIA RTX 5090 GPU, whereas our dynamic update requires only a few seconds per block and runs entirely on a 14-core CPU.

Keywords: Proof of Assets · Zero knowledge proof · Accumulator · Blockchain

1 Introduction

Zero-knowledge proof of assets (PoA) allows a prover to demonstrate ownership of a quantity of assets without revealing private information [8,9]. It can be used by individual users to satisfy asset thresholds for smart-contract interactions, access restricted liquidity pools, or participate in Sybil-resistant voting without locking funds or disclosing their financial footprint. Its most prominent institutional application is proof of solvency, where a custodian proves that its assets cover its liabilities. A complete proof-of-solvency system combines a proof of liabilities (PoL), which accounts for user balances owed by the custodian, with a PoA, which

* Corresponding author.
** Corresponding author.

accounts for assets controlled by the custodian [8,3,5,6]. These two components are then compared in zero knowledge to demonstrate that the assets sufficiently cover all liabilities.

The liability side is comparatively well suited to custom cryptographic constructions. A custodian maintains its liability database off-chain and can choose a proof-friendly authenticated data structure, such as a cryptographic accumulator or a dynamic authenticated ledger [1,3,5,6,7]. The asset side is different. On-chain balances are stored in a global state structure fixed by the blockchain protocol. An exchange cannot require the underlying blockchain to store its reserve balances in a zero-knowledge-friendly accumulator, making private PoA more challenging [14].

A common PoA approach adopted by custodians is public-address auditing, where custodians disclose their wallet addresses so that anyone can verify their reserves [1]. However, publicly available addresses allow anyone to trace a custodian's inflows, outflows, counterparties, liquidity management, and operational strategies, potentially revealing sensitive business information [30,29]. Exposing cold wallet addresses may also increase security risks by making high-value wallets easier to identify and target. It is therefore desirable to establish the validity of reserve assets without publicly disclosing the underlying addresses.

A natural approach is to construct a zero-knowledge proof that each hidden address is controlled by the custodian and that its claimed balance is consistent with the blockchain state. However, private PoA introduces additional challenges. First, two custodians may use the same wallet in their asset proofs, causing the same assets to be counted more than once. Second, a custodian may exploit temporary borrowing by acquiring assets shortly before generating a proof and returning them afterward, making its reserves appear larger than they are over time [4]. Existing work has addressed duplicate claims using privacy-preserving wallet identifiers. PLUME [2], for example, provides a mechanism for generating deterministic yet pseudorandom-looking identifiers for ECDSA key pairs, enabling verifiers to detect repeated claims without directly linking the identifiers to public keys. Such mechanisms can help mitigate duplicate claims, but the challenge of maintaining private asset proofs as blockchain states evolve remains.

One possible countermeasure to temporary borrowing is to require custodians to generate asset proofs frequently, potentially after every newly finalized block. Frequent verification makes it more difficult for a custodian to temporarily acquire assets and use them to support a single favorable snapshot. However, straightforward private PoA constructions are computationally expensive. In general, a private PoA must establish both ownership of hidden wallet addresses and consistency of their balances with the blockchain state. Ownership verification typically involves signature verification, while state verification requires authentication paths, such as Merkle proofs [13], against the blockchain state root. Performing these operations inside zero-knowledge circuits is costly, particularly when the custodian controls a large number of addresses. Recomputing these proofs for the entire reserve set at every epoch makes frequent asset verification impractical.

Dynamic proof of assets (DPoA) offers a way to alleviate this computational burden by allowing an existing asset proof to be updated according to changes in newly finalized blocks, rather than recomputing the proof from scratch. Provisions [8] introduced the idea of incremental asset updates. However, its address-privacy model relies on publicly disclosed addresses and a set of decoys, or an anonymity set, to obscure which addresses belong to the custodian. This does not provide full reserve-address privacy: the reserve and decoy addresses remain publicly available, and their on-chain activity may enable adversaries to distinguish the custodian's addresses through transaction analysis. Moreover, incorporating new reserve addresses may require disclosing additional decoy addresses to preserve the anonymity set. Thus, while Provisions supports dynamic asset updates, its privacy model does not meet the requirement of keeping the reserve address set entirely hidden.

These observations reveal two complementary challenges in dynamic PoA. A scheme that hides reserve addresses but recomputes expensive ownership and state proofs for every address at each epoch cannot efficiently support frequent verification. Conversely, a scheme that supports incremental updates but exposes reserve addresses or relies on publicly known anonymity sets does not provide the desired address privacy. Achieving practical dynamic PoA therefore requires both privacy-preserving representation of the reserve address set and an efficient mechanism for maintaining the asset proof as the blockchain evolves.

In this paper, we present **Pethoa**, a dynamic proof-of-assets system for account-based blockchains that bridges the gap between address privacy and efficient incremental verification. Pethoa represents the custodian's reserve address set using a cryptographic accumulator, avoiding the need to publicly disclose reserve addresses or construct a public anonymity set. To update the asset proof, Pethoa privately computes the intersection between the hidden reserve address set and the public list of addresses modified in a newly finalized block. The intersection results are represented using a vector commitment, enabling the prover to derive the corresponding aggregate balance changes without revealing which addresses belong to the reserve set. Furthermore, by exploiting the algebraic structure of the accumulator, Pethoa constructs a lightweight zero-knowledge circuit for the update procedure, avoiding the need to recompute expensive ownership and state authentication proofs over the entire reserve set at each epoch. Pethoa thus supports private reserve addresses while enabling efficient incremental asset verification.

We implement a prototype of Pethoa and evaluate its performance in account-based blockchain settings. Our evaluation considers the cost of initialization and dynamic updates, including proof generation and verification. The experimental results demonstrate the practicality of the proposed construction and show that Pethoa can substantially reduce the amortized cost of dynamic asset verification while preserving address privacy.

Contributions. Our contributions are summarized as follows:

– **Privacy-preserving dynamic PoA.** We propose Pethoa, a dynamic proof-of-assets system that keeps the reserve address set hidden without relying

on publicly disclosed anonymity sets, addressing the privacy limitations of existing dynamic PoA approaches.
- **Compact zero-knowledge circuit for dynamic updates.** We design a private-intersection-based update mechanism to identify reserve addresses affected by newly finalized blockchain updates without revealing their identities. By exploiting the algebraic properties of cryptographic accumulators, we transform expensive operations into computations outside the zero-knowledge circuit, leaving only lightweight relations to be verified inside the circuit. This enables a compact and efficient update circuit, reducing the cost of zero-knowledge proof generation while maintaining the correctness of incremental asset updates.
- **Implementation and evaluation.** We implement a prototype of Pethoa and experimentally evaluate its performance, demonstrating the feasibility of maintaining private asset proofs efficiently in account-based blockchain environments.

Scope and limitations. Pethoa hides the reserve addresses but may reveal the number of addresses included in the proof and does not eliminate leakage through on-chain transaction patterns or other side channels [29,30]. Its overall computational complexity may still scale linearly with the reserve set size. The verifier is assumed to maintain access to a blockchain full node, and verification costs may be higher than existing methods, making direct on-chain verification less suitable. Furthermore, Pethoa does not implement duplicate-claim prevention, which could be incorporated using mechanisms such as PLUME [2]. Finally, it does not eliminate third-party wallet cooperation inherent to reserve auditing; frequent verification can constrain temporary asset transfers but cannot fully prevent them.

2 Preliminaries

Let $\mathbb{F}_q$ be a prime field, and let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic groups of prime order q, equipped with a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We denote their generators by G_1, G_2 and use $[x]_1 = xG_1$ and $[x]_2 = xG_2$ for $x \in \mathbb{F}_q$.

2.1 Pedersen Commitment

Definition 1 (Pedersen Commitment). *Let $\mathbb{G}$ be a cyclic group of prime order q with independent generators G, H. The Pedersen commitment scheme [19] is defined as $\mathsf{Com}(m; r) = mG + rH$, where $m, r \in \mathbb{F}_q$. It is perfectly hiding and computationally binding under the discrete logarithm assumption, and satisfies the additive homomorphism property $\mathsf{Com}(m_1; r_1) + \mathsf{Com}(m_2; r_2) = \mathsf{Com}(m_1 + m_2; r_1 + r_2)$.*

2.2 KZG Polynomial Commitment

Definition 2 (KZG Polynomial Commitment). *A KZG polynomial commitment scheme [10] for polynomials of degree at most d consists of algorithms* $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verify})$*:*

- $\mathsf{Setup}(1^\lambda, d) \to \mathsf{pp}$*, generating public parameters from a secret* $\tau \in \mathbb{F}_q$*;*
- $\mathsf{Com}(\mathsf{pp}, f) \to C_f$*, where* $C_f = [f(\tau)]_1$*;*
- $\mathsf{Open}(\mathsf{pp}, f, z) \to (y, \pi_z)$*, where* $y = f(z)$ *and* $\pi_z = [Q_z(\tau)]_1$ *for* $Q_z(T) = (f(T) - y)/(T - z)$*;*
- $\mathsf{Verify}(\mathsf{pp}, C_f, z, y, \pi_z) \to \{0, 1\}$*, accepting iff*

$$e(C_f - [y]_1, [1]_2) = e(\pi_z, [\tau]_2 - z[1]_2).$$

The scheme satisfies correctness and computational binding under the corresponding KZG security assumption.

2.3 Rank-1 Constraint System (R1CS)

Definition 3 (R1CS). *An R1CS [12] over* $\mathbb{F}_q$ *is specified by matrices* $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}_q^{M \times N}$*. Given public input* $x \in \mathbb{F}_q^\ell$ *and witness* $w \in \mathbb{F}_q^{N-\ell-1}$*, let* $\mathbf{z} = (1, x^\mathsf{T}, w^\mathsf{T})^\mathsf{T}$*. The constraint system is satisfied iff*

$$(\mathbf{Az}) \circ (\mathbf{Bz}) = \mathbf{Cz},$$

where $\circ$ *denotes component-wise multiplication.*

2.4 zkSNARKs

Definition 4 (zkSNARK). *A zero-knowledge succinct non-interactive argument of knowledge (zkSNARK) [23] for an NP relation* $\mathcal{R}$ *is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *such that:* $\mathsf{Setup}(1^\lambda, \mathcal{R}) \to (\mathsf{pk}, \mathsf{vk})$*,* $\mathsf{Prove}(\mathsf{pk}, x, w) \to \pi$*, and* $\mathsf{Verify}(\mathsf{vk}, x, \pi) \to \{0, 1\}$*. It satisfies completeness, knowledge soundness, and zero knowledge, with proof size and verification complexity substantially smaller than the size of the computation represented by* $\mathcal{R}$*.*

2.5 Bulletproofs

Definition 5 (Bulletproofs). *Bulletproofs [12] are transparent zero-knowledge arguments of knowledge for arithmetic relations, with no trusted setup and logarithmic proof size in the number of constraints. For a arithmetic relation* $\mathcal{R}$*, a Bulletproof consists of algorithms* $(\mathsf{Prove}, \mathsf{Verify})$ *that allow a prover holding* (v, r) *to prove that a commitment* $C = \mathsf{Com}(v; r)$ *satisfies* $\mathcal{R}$*, without revealing* v*, while ensuring completeness, soundness, and zero knowledge.*

2.6 Range Proofs

Definition 6 (Range Proof). *Given a commitment $C = \mathsf{Com}(v; r)$ and an integer ℓ, the range relation is defined as*

$$\mathcal{R}_{\mathsf{range}}((C, \ell); (v, r)) = 1 \iff C = \mathsf{Com}(v; r) \ \wedge \ 0 \le v < 2^{\ell},$$

where v is interpreted as its canonical integer representative in $\{0, \ldots, q-1\}$. A range proof [34,12] is a zero-knowledge argument of knowledge for $\mathcal{R}_{\mathsf{range}}$.

3 Model and Security Goals

3.1 State and Notation

Let $\mathcal{A}$ be the blockchain address space. For a finalized state root sr, $\mathsf{bal}_{\mathsf{sr}}(a)$ denotes the canonical integer balance of address $a \in \mathcal{A}$.

At epoch i, the prover controls a hidden, duplicate-free reserve set $S_i \subseteq \mathcal{A}$ of size $n_i = |S_i|$. Its aggregate balance under sr_i is $sum_i = \sum_{a \in S_i} \mathsf{bal}_{\mathsf{sr}_i}(a)$. The public state is $X_i = (\mathsf{sr}_i, d_i, C_i, n_i)$, where d_i is a binding digest of S_i and $C_i = \mathsf{Com}(sum_i; r_i)$ is a Pedersen commitment with opening randomness r_i. The symbol W_i denotes the prover's private state needed to continue from X_i: it contains S_i, the ownership credentials for its addresses, and the opening (sum_i, r_i) of C_i. Any additional secret representation needed to realize d_i is construction-specific and is introduced only in Section 4.

For an ordinary transition from sr_{i-1} to sr_i, the verifier obtains the complete account-state changes produced while executing the corresponding finalized blocks. From it, the verifier deterministically derives a canonical balance-delta list $\Delta_i = \big((a_{i,j}, \delta_{i,j})\big)_{j=1}^{m_i}$ where the addresses $a_{i,j}$ are sorted and distinct, and $\delta_{i,j} = \mathsf{bal}_{\mathsf{sr}_i}(a_{i,j}) - \mathsf{bal}_{\mathsf{sr}_{i-1}}(a_{i,j}) \neq 0$. And we use $\Gamma_i = (\mathsf{sr}_{i-1}, \mathsf{sr}_i, \Delta_i)$ to denote the resulting public transition record.

We write $\mathbf{a}_i = (a_{i,1}, \ldots, a_{i,m_i})$, $\boldsymbol{\delta}_i = (\delta_{i,1}, \ldots, \delta_{i,m_i})$, and $\mathbf{u}_i = (u_{i,1}, \ldots, u_{i,m_i})$ for the hidden membership vector with $u_{i,j} = 1[a_{i,j} \in S_{i-1}]$. The reserve-set balance change is $\Delta sum_i = \langle \mathbf{u}_i, \boldsymbol{\delta}_i \rangle$. An ordinary update keeps the reserve set fixed, hence $d_i = d_{i-1}$ and $n_i = n_{i-1}$, while changing the root and aggregate commitment. An insertion at the already accepted root sr_i adds a new address $a^\star$ with authenticated balance $b^\star = \mathsf{bal}_{\mathsf{sr}_i}(a^\star)$ and produces a state $X_i' = (\mathsf{sr}_i, d_i', C_i', n_i{+}1)$. We use $\omega^\star$ for the ownership credential and native-chain authentication data associated with $a^\star$, and $t \in \mathbb{Z}$ for a public asset threshold. The public bounds D and $m_{\max}$ specify, respectively, the maximum supported reserve-set digest degree and the maximum number of entries in an ordinary-update record.

All balances and balance changes have their usual integer semantics, while the zero-knowledge proofs perform arithmetic over $\mathbb{F}_q$. We therefore fix public bounds that ensure that the relevant integers have unique representations modulo q.

Let B_{bal} bound the balance of one account and let B_{sum} bound the total balance of the reserve set. Thus, for every authenticated account, $0 \le \mathsf{bal}_{\mathsf{sr}}(a) \le B_{\mathsf{bal}}$

and $0 \le sum_i \le B_{\mathsf{sum}}$. Since the reserve set contains at most D addresses, it is sufficient to choose $B_{\mathsf{sum}} \ge DB_{\mathsf{bal}}$. For an ordinary update, each public balance difference δ_j is a signed integer. We let B_{upd} bound the absolute value of any aggregate balance change that can be selected from one canonical update:

$$\left| \sum_{j=1}^{m} v_j \delta_j \right| \le B_{\mathsf{upd}} \qquad \text{for all } (v_1, \ldots, v_m) \in \{0,1\}^m.$$

In particular, the actual reserve-set update $\Delta sum_i = \sum_j u_j \delta_j$ lies in $[-B_{\mathsf{upd}}, B_{\mathsf{upd}}]$.

Finally, let B_{cmp} bound every public threshold or committed liability used in a comparison proof. We choose the field order q large enough that $2B_{\mathsf{upd}} < q, 2B_{\mathsf{sum}} + B_{\mathsf{upd}} < q, B_{\mathsf{sum}} + B_{\mathsf{cmp}} + 1 < q$. The meanings of these inequalities are given in appendix.

3.2 Security Goals

We define the following security requirements for a DPoA scheme, using the standard simulation-based perspective of zero-knowledge cryptography [22].

State soundness Let $X_i = (\mathsf{sr}_i, d_i, C_i, n_i)$ be a public DPoA state. We say that X_i is *valid* if there exists a witness W_i containing a duplicate-free reserve set S_i, ownership credentials for its addresses, a nonzero masking scalar α_i, and an opening (sum_i, r_i) of C_i such that the following properties hold:

1. $|S_i| = n_i$;
2. every address in S_i is controlled by the prover;
3. d_i is a binding digest of S_i
4. the aggregate commitment satisfies $sum_i = \sum_{a \in S_i} \mathsf{bal}_{\mathsf{sr}_i}(a), C_i = \mathsf{Com}(sum_i; r_i)$.

We write $\mathsf{ValidState}(X_i; W_i) = 1$ when these conditions hold.

State soundness requires that a polynomial-time malicious prover can cause the verifier to accept an invalid state only with negligible probability. Moreover, starting from a valid state, the prover cannot forge an accepted transition whose successor state is invalid.

Correctness An honest prover with a valid reserve state, valid ownership credentials, and correct blockchain-authentication data can generate proofs that are accepted by the verifier.

Range Proof Soundness For a valid public state, an accepted range proof must return the correct comparison result for the value bound by its asset commitment. In particular, a polynomial-time malicious prover cannot convince the verifier that the committed asset value is above a threshold when it is below it, or vice versa. The same requirement applies when the threshold is replaced by a committed liability value.

State Privacy The public states and proof transcripts should reveal no information about the hidden reserve state beyond explicitly defined public leakage. We formalize this requirement by a stateful simulator that receives only the public leakage, including blockchain state roots, canonical balance-delta lists, reserve-set sizes and operation types. Without knowing the reserve addresses, balances, membership vectors, or commitment openings, the simulator must generate a sequence $\widetilde{X}_0, \widetilde{X}_1, \ldots$ together with the corresponding proof transcripts whose joint distribution is computationally indistinguishable from a real execution. Therefore, beyond the explicitly released leakage, neither the persistent public states nor the proofs reveal information about the hidden reserve set or its intersection with public blockchain updates.

Range Proof Privacy Range proofs should reveal no information about the committed asset value beyond the public comparison statement and its Boolean result. For any fixed valid public state X_i containing an aggregate commitment C_i, and for any public threshold query t with result

$$b = \mathbf{1}[sum_i \geq t],$$

there exists a PPT simulator that, given only the public statement (X_i, t, b), generates a range-proof transcript computationally indistinguishable from a real one. In particular, the proof reveals neither the aggregate asset value sum_i nor an opening of C_i beyond what is implied by the disclosed Boolean result. The same requirement applies to comparisons against a committed liability value.

4 Construction

In this section, we present the protocols interactively to make their message order and dependencies explicit. They can be made non-interactive using the standard Fiat–Shamir transformation [20] in the random-oracle model: for round j of a component identified by tag, the verifier's challenge is replaced by

$$c_j = H_{\mathsf{FS}}\big(\mathsf{encode}(\mathsf{tag}, \mathsf{ctx}, x, j, \mathsf{tr}_{<j})\big),$$

where x is the complete public statement and $\mathsf{tr}_{<j}$ contains all preceding messages, including the prover's message immediately before this challenge. The context ctx identifies the public parameters, blockchain, operation, and session, and binds the relevant predecessor and successor states. Here, $H_{\mathsf{FS}} : \{0,1\}^* \to \mathbb{F}_q$ is a hash-to-field random oracle.

In our system, both prover and the verifier should have access to new finalized blocks independently. For example, they could both run full execution nodes of the underlying blockchain. All public paramters are generated in the setup phase before the protocol starts. The overall system could be understood as follows: the prover maintains a private state including its reserve-addresses, their total balances under the current state root, and publish the commitment to each element in this private state. Each time a new block is finalized and the state root

is updated, the prover also update its private state and state commitment, as well as generating a proof which takes the new block as public input and proves the correctness of state commitment update. Then it will conduct a range proof to show that the updated total balance still exceeds the given threshold (liability or its commitment).

4.1 KZG variants

Our construction uses KZG-opening arguments that link polynomial evaluations to external commitments while hiding the evaluated values and, when required, the evaluation point. These interfaces abstract and extend techniques underlying position-hiding commitment linkability in Caulk [16] and the commit-and-prove methodology of LegoSNARK [15]. We instantiate these interfaces for Pethoa rather than invoking the complete lookup or proof-composition systems of those works. Below are the description of these interfaces. Here, C is a KZG commitment to a degree-bounded polynomial f.

- **Public point and committed value (KZG-CVOpen).** The public input is $x = (C, z, C_y)$, and the prover's witness is $w = (f, y, \rho_y)$. The interface proves that
$$C = [f(\tau)]_1, \qquad C_y = \mathsf{Com}(y; \rho_y), \qquad y = f(z),$$
without revealing f and y.
- **Public points and committed values (Batch-KZG-CVOpen).** The public input is $x = (C, \mathbf{z}, D_Y)$, where $\mathbf{z} = (z_1, \ldots, z_m)$ consists of distinct public points. The prover's witness is $w = (f, \mathbf{y}, \rho_Y)$. The interface proves that
$$C = [f(\tau)]_1, \qquad D_Y = \mathsf{Com}(\mathbf{y}; \rho_Y), \qquad y_j = f(z_j) \quad \text{for every } j,$$
without revealing f and the evaluation vector $\mathbf{y}$. The order of its coordinates agrees with that of $\mathbf{z}$.
- **Committed point and committed value (KZG-CPVOpen).** The public input is $x = (C, C_u, C_y)$, and the prover's witness is $w = (f, u, \rho_u, y, \rho_y)$. The interface proves that
$$\begin{aligned} C &= [f(\tau)]_1, \\ C_u &= \mathsf{Com}(u; \rho_u), \qquad C_y = \mathsf{Com}(y; \rho_y), \\ y &= f(u), \end{aligned}$$
without revealing the polynomial f, the point u, and the evaluation y.

The relationship to prior work is further discussed in the related-work section; concrete protocols and their component security arguments appear in Appendix B.

4.2 Initialization

Initialization establishes the initial hidden reserve state at a finalized blockchain root sr_0. Let the prover's reserve set be $S_0 = \{a_1, \ldots, a_n\}$, where $n \le D$.

Authenticating the reserve set and its aggregate. For every $a_k \in S_0$, the prover obtains its ownership credential ω_k^{own}, authenticated balance b_k, and native-state proof π_k^{chain} under sr_0. Let $x_k = \mathsf{Enc}_{\mathbb{F}}(a_k)$. The prover verifies $\mathsf{OwnVerify}(a_k, \omega_k^{\text{own}}) = 1$ and $\mathsf{ChainVerify}(sr_0, a_k, b_k, \pi_k^{\text{chain}}) = 1$ for all k, together with the balance range constraints and duplicate-freeness of the encoded addresses. It computes the aggregate reserve balance $\mathsf{sum}_0 = \sum_{k=1}^{n} b_k$ and commits to it as $C_0 = \mathsf{Com}(\mathsf{sum}_0; r_0)$. This commitment serves as the public representation of the reserve assets at the initial state root.

Committing to the hidden reserve set. The prover constructs the set polynomial $F_{S_0}(T) = \prod_{k=1}^{n}(T - x_k)$, whose roots correspond to the encoded reserve addresses [11,28]. It samples a nonzero scalar $\alpha \in \mathbb{F}_q^*$ and forms the masked polynomial $P_0(T) = \alpha F_{S_0}(T)$. It publishes the KZG digest $d_0 = [P_0(\tau)]_1$.

Binding the digest to the authenticated addresses. To bind the digest to the addresses used in the ownership and balance checks, the prover commits to the polynomial-defining values as $h_{\text{shape}} = \mathsf{HCom}((\alpha, x_1, \ldots, x_n); s_{\text{shape}})$. The verifier samples $\zeta_{\text{init}} \xleftarrow{\$} \mathbb{F}_q$ and sends it to the prover. The prover evaluates $y = P_0(\zeta_{\text{init}}) = \alpha \prod_{k=1}^{n}(\zeta_{\text{init}} - x_k)$, commits to it as $C_y = \mathsf{Com}(y; r_y)$, and provides a KZG-CVOpen proof π_y that the polynomial represented by d_0 evaluates at ζ_{init} to the value committed in C_y.

The initialization relation verifies that the committed evaluation is consistent with the authenticated address encodings and α, while also checking the ownership, balance, duplicate-freeness, and aggregate commitment relations. This binds the public digest d_0 to the hidden reserve set authenticated at sr_0. The resulting public state is $X_0 = (sr_0, d_0, C_0, n)$. The prover retains the reserve set, ownership credentials, masking scalar, polynomial representation, and commitment openings in its private state W_0.

4.3 Dynamic Update

An ordinary update changes account balances but leaves the hidden reserve set unchanged, i.e., $S_i = S_{i-1}$. The prover therefore reuses the masked set polynomial P, digest d, and set size n. Let $\Delta_i = ((a_1, \delta_1), \ldots, (a_m, \delta_m))$ be the canonical balance-delta list derived from the finalized transition from sr_{i-1} to sr_i, where each address appears once and δ_j denotes its balance change. The goal is to update the committed reserve balance using the public changes without revealing which changed addresses belong to the hidden reserve set.

Private intersection with the public update list. For each changed address, the prover computes $x_j = \mathsf{Enc}_{\mathbb{F}}(a_j)$ and $y_j = P(x_j)$. Since the roots of P correspond to the reserve addresses, it derives the membership bit $u_j = \mathbf{1}[a_j \in S_{i-1}] = \mathbf{1}[y_j = 0]$. The vector $\mathbf{y} = (y_1, \ldots, y_m)$ thus encodes the intersection between the public changed-address list and the hidden reserve set. The prover commits to the evaluation vector as $D_Y = \sum_{j=1}^{m} y_j G_{Y,j} + r_Y H_Y$ and provides a batch KZG-CVOpen proof π_i^{batch} showing that these committed evaluations correspond to

the polynomial represented by the predecessor digest d_{i-1} at the public points $(x_j)_{j=1}^m$. This binds the hidden evaluations to the existing reserve-set digest without revealing the evaluation vector.

Establishing the membership vector. The prover commits to the membership vector $\mathbf{u} = (u_1, \ldots, u_m)$ as $C_U = \sum_{j=1}^m u_j U_j + r_U H_U$. To prove that each committed bit is consistent with its polynomial evaluation, the prover supplies auxiliary values z_j and proves the relations $u_j y_j = 0$ and $y_j z_j = 1 - u_j$ for every j. These relations imply that $u_j = 1$ if and only if $y_j = 0$, and enforce $u_j \in \{0, 1\}$. The resulting committed-input R1CS proof is denoted by π_i^{logic}. By using D_Y and C_U in the same relation, the membership vector is linked to the polynomial evaluations established above.

Updating the aggregate balance. The reserve balance changes only at addresses in the reserve set, so its update is $\Delta_i^{\text{sum}} = \sum_{j=1}^m u_j \delta_j = \langle \mathbf{u}, \boldsymbol{\delta}_i \rangle$. The prover commits to this change as $C_D = \mathsf{Com}(\Delta_i^{\text{sum}}; r_D)$ and computes $\mathsf{sum}_i = \mathsf{sum}_{i-1} + \Delta_i^{\text{sum}}$ and $r_i = r_{i-1} + r_D$. By the homomorphic property of the Pedersen commitment, the updated aggregate commitment is $C_i = C_{i-1} + C_D$. The prover supplies an inner-product proof [12] π_i^{ip} establishing that the committed membership vector and aggregate change satisfy the above inner-product relation, including the prescribed bound on Δ_i^{sum}.

Verification and state transition. The verifier independently derives the canonical update list Δ_i from the finalized blockchain transition. It checks that the predecessor state is the last accepted state, the new root is finalized, the update list is canonical and within the supported size bound, and the aggregate commitment is updated consistently. It then verifies π_i^{batch} against d_{i-1} and D_Y, π_i^{logic} using the same commitments D_Y and C_U, and π_i^{ip} using C_U, C_D, and the public balance-delta vector. The shared commitments link the polynomial evaluations, membership vector, and aggregate balance change to the same update. Upon successful verification, the public state becomes $X_i = (sr_i, d_{i-1}, C_i, n_{i-1})$. Thus, the blockchain root and aggregate balance commitment advance, while the reserve-set digest and size remain unchanged.

4.4 Address Insertion

Insertion adds a new address to the hidden reserve set without reinitializing the system. Let the current state be $X_i = (sr_i, d_i, C_i, n_i)$, and let $a^\star$ be the new address with authenticated balance $b^\star$ under sr_i. The successor state is $X_i' = (sr_i, d_i', C_i', n_i + 1)$. The insertion proof establishes that the prover controls the new address, its balance is authenticated, the address was not previously in the reserve set, and the public state is updated consistently.

Prior non-membership. Let $u = \mathsf{Enc}_{\mathbb{F}}(a^\star)$ and $y = P_i(u)$. Since the roots of P_i correspond exactly to the current reserve addresses, $y \neq 0$ certifies that $a^\star$ is not already in the reserve set, except with negligible encoding-collision

probability. The prover commits to the point and evaluation as $C_u = \mathsf{Com}(u; \rho_u)$ and $C_y = \mathsf{Com}(y; \rho_y)$, samples $\beta \in \mathbb{F}_q^*$, and publishes the randomized point handle $R_u = [\beta(\tau - u)]_2$. An interactive $\mathsf{HandleAuth}$ protocol proves that R_u is formed using the same hidden point committed in C_u, without revealing u.

Proving the hidden evaluation. Given the authenticated point handle, the prover provides a KZG-CPVOpen proof π_{cpv} that the polynomial represented by d_i evaluates at the point committed in C_u to the value committed in C_y. It also proves in zero knowledge that C_y commits to a nonzero value, by demonstrating knowledge of an inverse ν satisfying $y\nu = 1$, with y bound to the opening of C_y. Together, these proofs establish that the inserted address is absent from the existing reserve set, while hiding both its encoding and polynomial evaluation.

Authenticating the new address and balance. Let $\omega^\star = (\omega^\star_{\mathrm{own}}, \pi^\star_{\mathrm{chain}})$ denote the ownership credential and native-state authentication data for $a^\star$. The prover commits to its balance as $C_b = \mathsf{Com}(b^\star; r_b)$ and provides a committed-input proof establishing that the same hidden address committed in C_u satisfies the ownership verification relation and that its committed balance in C_b is authenticated under sr_i. The proof also verifies the commitment openings and balance range. Thus, the non-membership and authentication proofs are bound to the same committed address.

Updating the reserve state. The new set polynomial is $P_i'(T) = \beta P_i(T)(T-u)$, and the prover publishes the successor digest $d_i' = [P_i'(\tau)]_1$. Using the authenticated point handle, the verifier checks the digest transition $e(d_i', [1]_2) = e(d_i, R_u)$. This establishes that the new digest represents the previous set with the inserted point, up to the prescribed polynomial masking. The aggregate balance commitment and reserve-set size are updated as $C_i' = C_i + C_b$ and $n_i' = n_i + 1$. The prover updates its private reserve set, aggregate balance, commitment opening, polynomial representation, and ownership data accordingly. The resulting public state is $X_i' = (sr_i, d_i', C_i', n_i + 1)$. The insertion proof consists of the commitments and proof components for the point handle, polynomial evaluation, nonzero evaluation, and authenticated ownership and balance relation, all bound to the same predecessor state, successor state, and relevant commitments.

4.5 Range Proofs

A range proof allows the prover to demonstrate whether the committed reserve balance satisfies a given threshold without revealing the balance. Given a valid state X_i and a public threshold $t \in [0, B_{\mathrm{cmp}}]$, the protocol returns the Boolean outcome $b = \mathbf{1}[\mathsf{sum}_i \geq t]$.

Positive outcome. When $b = 1$, the prover demonstrates that the committed balance is at least t by proving knowledge of $(\mathsf{sum}_i, r_i, s_+)$ satisfying $C_i = \mathsf{sum}_i V + r_i H_B$, $\mathsf{sum}_i - t = s_+$, and $0 \leq s_+ \leq B_{\mathrm{sum}}$. The verifier checks the commitment opening relation and range constraint, establishing $\mathsf{sum}_i \geq t$.

Negative outcome. When $b = 0$, the prover demonstrates that the committed balance is strictly below t by proving knowledge of $(\mathsf{sum}_i, r_i, s_-)$ satisfying $C_i = \mathsf{sum}_i V + r_i H_B$, $t - 1 - \mathsf{sum}_i = s_-$, and $0 \le s_- \le B_{\mathrm{cmp}}$. Since balances are integers, this relation is equivalent to $\mathsf{sum}_i < t$. The verifier checks the corresponding commitment opening and range constraints. The outcome b is included in the public statement, and only the corresponding branch is verified.

Comparison against a committed liability. The same construction supports comparison against a committed liability $C_L = LV + r_L H_B$. The prover establishes whether $\mathsf{sum}_i \ge L$ without revealing either value. The positive branch proves a nonnegative difference $\mathsf{sum}_i - L$, while the negative branch proves a nonnegative difference $L - 1 - \mathsf{sum}_i$. The liability is additionally constrained to the admissible interval $[0, B_{\mathrm{cmp}}]$. These comparison relations can be implemented using Bulletproof range proofs [12] or the same SNARK/zkVM backend used for the other proof components.

5 System Performance

5.1 Complexity Analysis

Let n denote the size of the hidden reserve set, m the number of addresses involved in one update interval, and ℓ the bit length of the value used in a range comparison. We omit one-time setup, finality verification, and the construction of the public update list.

Initialization. The prover constructs the set polynomial and its KZG commitment, and proves ownership and authenticated balances for all n reserve addresses. The polynomial operations and proof generation require $\widetilde{O}(n)$ prover work. With a succinct proof system, the verification time and proof size are $O(1)$ with respect to n.

Ordinary updates. The prover evaluates the degree-n set polynomial at the m public update points and generates a batch opening. Using fast multipoint evaluation, the polynomial computation costs $\widetilde{O}(n + m)$. The update circuit contains $O(m)$ constraints, and the associated inner-product proof requires $O(m)$ prover work. Thus, the total prover complexity is $\widetilde{O}(n + m)$. Verification requires $O(m)$ work, while the proof size is $O(m)$, dominated by the batch-opening responses. Neither verification nor proof size depends on n.

Insertion. Inserting a new address requires evaluating the set polynomial, computing its quotient polynomial, and updating its commitment, which costs $O(n)$ prover work. Since the insertion proof involves only the new address and its balance, a succinct proof system yields $O(1)$ verification time and proof size with respect to n.

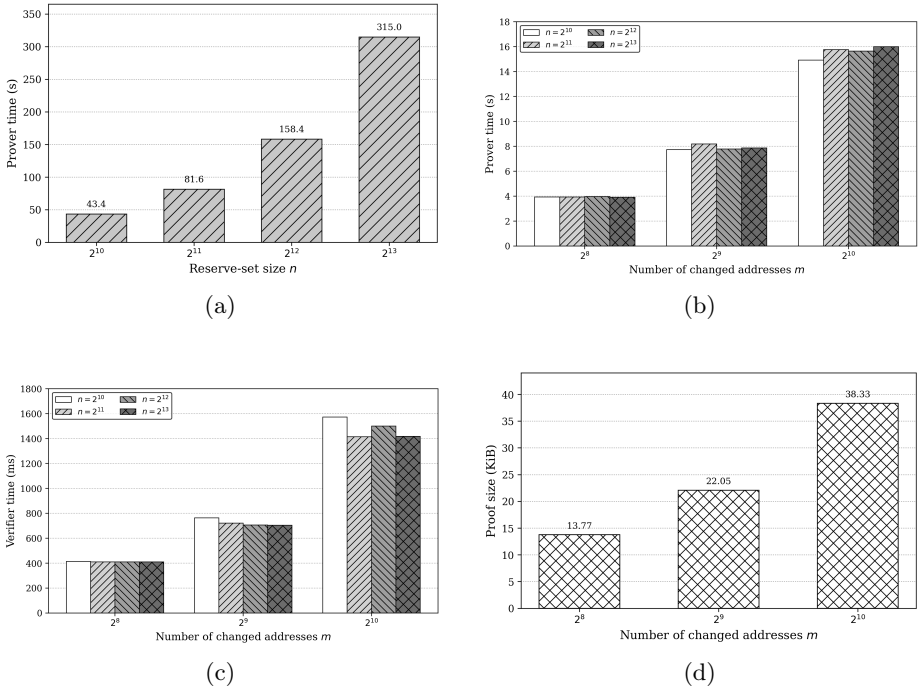


Fig. 1. Performance of Pethoa. Subfigure (a) shows the initialization prover time for different reserve-set sizes n. Subfigure (b) shows the update prover time for different reserve-set sizes n and numbers of changed addresses m. Subfigures (c) and (d) report the corresponding verifier time and proof size, respectively.

Range proofs. A range proof for an ℓ-bit value using Bulletproofs requires $O(\ell)$ prover and verifier work and $O(\ell)$ group elements in the proof. Its complexity is independent of n and m.

5.2 Experimental Environment

All experiments were conducted on a Linux server running Ubuntu 22.04, equipped with dual Intel Xeon Gold 6530 CPUs, 64 GB of RAM, and an NVIDIA RTX 5090 GPU. The software environment consists of SP1 zkVM [36] v6.2.4 and CUDA 13.2.

The GPU is utilized solely to accelerate SP1 proof generation during the initialization and address insertion phases. All other computations, including the entire update stage, are executed on the CPU. Furthermore, our benchmarks strictly measure the core proof time, excluding the overhead associated with data loading and zkVM preparation.

We use synthetic blockchain states represented by a depth-32 binary Merkle tree with Keccak hashing. Native state proofs are verified only during initialization

Table 1. Asymptotic costs of the main protocol operations.

Operation	Prover	Verifier	Proof size
Initialization	$\widetilde{O}(n)$	$O(1)$	$O(1)$
Ordinary update	$\widetilde{O}(n+m)$	$O(m)$	$O(m)$
Insertion	$O(n)$	$O(1)$	$O(1)$
Range proof	$O(\ell)$	$O(\ell)$	$O(\log \ell)$

Table 2. Comparison of PoA schemes.

Scheme	Full privacy	Incremental proof	Prover time	Proof size	Verifier time
Provisions [8]*	No	Yes	≈ 14.5 min	≈ 310 MB	≈ 3.2 min
gOTzilla [9]	No	No	14.89 s†	6.18 MB†	N/R
Static baseline‡	Yes	No	2.59 min	1.21 MiB	67.06 ms
Pethoa initialization§	Yes	N/A	2.64 min	1.21 MiB	69.01 ms
Pethoa update§	Yes	Yes	7.78 s	22.05 KiB	0.71 s
Pethoa insertion§	Yes	N/A	4.66 s	1.21 MiB	81.04 ms

* Provisions is evaluated with an anonymity set of $N = 500{,}000$ public keys. The reported costs are for full PoA generation and verification, excluding the non-collusion extension. Incremental updates are supported but not separately benchmarked.
† gOTzilla reports a 14.89 s end-to-end runtime and 6.18 MB of total interactive communication for one hidden selection from $N = 2^{20}$ entries; separate prover and verifier costs are not reported.
‡ The static baseline verifies ownership and Merkle proofs for all hidden addresses within a single zero-knowledge circuit.
§ Pethoa is evaluated with $n = 2^{12}$ reserve addresses. The ordinary update uses $m = 2^9$ changed addresses, while insertion adds one address.

and address insertion. Ordinary updates instead operate on the public balance-delta list, so their proof-generation cost does not depend on the native state-tree structure for a fixed update list. To assess the cost of native Ethereum state authentication, we additionally benchmark MPT proof verification at the same account counts in Appendix C. Its proving cost is comparable to that of our initialization benchmark, suggesting that native state authentication adds a modest constant-factor overhead rather than an order-of-magnitude increase. For ownership proofs, we employ standard ECDSA signatures. To optimize the performance of computationally expensive operations in SP1, such as Keccak hashing and Secp256k1 signature verification, we leverage precompiled syscalls [37].

We tested the system across various private set sizes (n), specifically 2^{10}, 2^{11}, 2^{12}, and 2^{13}. For each configuration, the size of the delta list was set to 2^8, 2^9, and 2^{10}, which approximates the number of state transitions typically contained within a single block.

5.3 Experimental Evaluation

We first compare the performance of our dynamic system against a static proof baseline. Similar to our system, the static proof includes both ownership and Merkle proof verification. The primary difference is that the static baseline does not require authenticating the KZG digest, a mandatory step in the initial-

ization phase of our dynamic system. Experimental results indicate that this authentication introduces only a marginal overhead.

During initialization, the total proof generation time grows linearly with the private set size n, reaching several minutes even with GPU acceleration. In contrast, during the update stage, the prover time is reduced to mere seconds, despite being executed exclusively on the CPU. Furthermore, this update time is largely independent of n in our experiments, but grows approximately linearly with the update size m. This is because the $\widetilde{O}(n)$ polynomial operations are performed efficiently outside the zero-knowledge circuit, while most of the proving time is spent on the $O(m)$ in-circuit computations. We use Bulletproofs for these computations to reduce the prover's overhead, at the cost of increased verifier computation. Nevertheless, this overhead remains acceptable for off-chain verifiers, such as independent auditing institutions.

For $n = 2^{12}$, the update prover takes 7.78 seconds when $m = 2^9$ and 15.64 seconds when $m = 2^{10}$, with corresponding verifier times of 0.71 and 1.50 seconds. To assess the practical relevance of these results, we refer to the Ethereum block analysis accompanying EIP-7928 [18]. The analysis covers 1,000 Ethereum mainnet blocks and reports an average of 795 balance-change records per block. Although this number does not exactly correspond to the number of distinct addresses with nonzero net balance changes used in Pethoa, it provides a useful reference for the update workload.

Using $m = 800$ as an illustrative workload, linear interpolation between our measurements gives an estimated prover time of 12.20 seconds and a verifier time of 1.15 seconds. Ethereum operates with a 12-second slot interval [38], so the estimated prover time is close to the interval between blocks. These results suggest that Pethoa has the potential to support frequent, near-block-by-block asset verification. Moreover, the update phase currently runs entirely on the CPU because a suitable GPU implementation of Bulletproofs is not available in our experimental setup. GPU acceleration may further improve performance. However, these estimates do not include blockchain data processing and communication overhead, and sustained performance under real Ethereum workloads remains to be evaluated.

Regarding address insertion, the primary computational bottlenecks remain the ECDSA and Merkle proof verifications. The only operation dependent on n is the KZG witness generation, which accounts for a negligible fraction of the total execution time. Consequently, the insertion time appears effectively constant at our experimental scale.

Although we do not apply a Groth16 [23] wrapper to our SP1 proofs, we utilize the framework's compressed mode. As a result, the proof sizes for both initialization and insertion remain nearly constant, albeit relatively large.

Finally, we compare our system with Provisions and the PoA instantiation of gOTzilla, both of which were developed and evaluated in the context of Bitcoin.

Unlike our approach, both of these systems rely on an anonymity set: they publish a large group of addresses and simply hide the custodian's actual holdings

within it. Consequently, their notion of privacy is defined merely as the verifier's inability to distinguish which specific addresses belong to the prover.

Regarding proof generation, gOTzilla is a purely static scheme based on an MPC-in-the-Head (MPCitH) protocol, a paradigm originating in zero-knowledge constructions from secure multiparty computation [35]. Provisions allows incremental updates for new blocks, similar to our system, but it imposes a heavy burden on the verifier, who must maintain the full state information for every address in the massive anonymity set.

Because the underlying architectural models are different, we can only provide a rough performance comparison. For this baseline comparison, our system is evaluated with a precise private set of 2^{12} addresses. In contrast, we reference a Provisions configuration with an anonymity set of 5×10^5 addresses (which intentionally obscures the actual number of owned addresses), and a gOTzilla configuration that proves ownership of a single address hidden within an anonymity set of 2^{20}.

6 Related Work

6.1 Proofs of Assets

Proof of Assets (PoA) enables a custodian to demonstrate ownership of on-chain assets, and is commonly combined with Proof of Liabilities (PoL) to establish solvency. Privacy-preserving PoA aims to verify asset ownership and aggregate balances without revealing the underlying reserve addresses.

Provisions [8] introduces a privacy-preserving proof of solvency for Bitcoin exchanges based on a public anonymity set. The custodian uses hidden selectors to identify its owned addresses within the candidate set and cryptographically binds address ownership to the corresponding balances. Provisions also supports incremental auditing, allowing unchanged per-address proof components to be reused across updates. However, its privacy depends on a publicly disclosed candidate set, exposing the addresses in the anonymity set and potentially allowing reserve addresses to be identified through transaction analysis.

gOTzilla [9] improves the efficiency of proving large disjunctions by separating the evaluation of an expensive computation from private selection among candidate statements. Its PoA application allows a prover to privately select an entry from a large UTXO set and prove control over it without revealing the selected entry.

Pethoa addresses a different combination of requirements: keeping the reserve address set private while efficiently maintaining asset proofs as blockchain states evolve. Unlike Provisions, it does not require a public candidate address set or decoys. Unlike gOTzilla, it maintains an aggregate proof over a hidden reserve set rather than proving the selection of a single asset entry. Pethoa supports dynamic updates by identifying the intersection between the hidden reserve set and the public blockchain update list, and uses a compact update circuit to reduce the cost of recurring proof generation.

6.2 Accumulators and Committed-Evaluation Proofs

Pethoa's KZG-based constructions are closely related to existing commitment-based proof techniques, particularly Caulk [16] and LegoSNARK [15], which also leverage cryptographic commitments to establish relations over hidden values. The similarity is not merely at the level of using commitments: these works share the underlying idea of binding witness values to cryptographic commitments and proving consistency of related values without revealing the witnesses. Caulk uses KZG-based vector commitments to support position-hiding membership and lookup proofs, while LegoSNARK provides a commit-and-prove framework for composing proofs over committed witnesses.

Building on this paradigm, Pethoa abstracts reusable KZG-based proof interfaces independent of downstream applications. We instantiate these interfaces in PoA to maintain the hidden reserve set and its aggregate asset value under dynamic blockchain updates.

6.3 Duplicate-Claim Prevention

Privacy-preserving asset proofs may allow the same wallet to be claimed by multiple participants, potentially leading to double counting of assets. Preventing such duplicate claims requires a mechanism to recognize repeated claims without publicly linking them to the underlying wallet identity.

PLUME [2] introduces a deterministic nullifier scheme for ECDSA key pairs that enables a key holder to produce a unique, pseudonymous identifier under a specified context. The nullifier can be verified through a zero-knowledge proof linking it to the corresponding secret key, without revealing the public key. Applications can record previously used nullifiers and reject repeated claims, thereby preventing the same key from being used more than once within a nullifier domain. PLUME is complementary to Pethoa: Pethoa maintains privacy-preserving asset proofs and dynamic reserve updates, while PLUME can provide duplicate-claim detection across independent proofs or custodians.

7 Conclusion

We presented Pethoa, a dynamic privacy-preserving Proof of Assets system for hidden address sets on account-based blockchains. Pethoa enables custodians to update their committed asset totals as the blockchain evolves, without revealing their reserve addresses or which addresses are affected by public updates. By combining private intersection with commitment-based update verification, Pethoa avoids repeatedly performing expensive ownership and state authentication checks, which are confined to initialization. Our security analysis establishes soundness and simulation-based privacy under the stated assumptions.

Experimental results demonstrate the efficiency of this approach. Initialization takes 43.417 s for $n = 2^{10}$ addresses and 5.25 min for $n = 2^{13}$, whereas update proving depends primarily on the public update-list size. For $m = 2^8$, updates take

about 3.94 s with a proof size of 13.77 KiB, largely independent of the reserve-set size. For $m = 2^{10}$, proving takes approximately 15–16 s with a proof size of 38.33 KiB. These results demonstrate that Pethoa enables efficient recurring asset updates without sacrificing reserve-set privacy.

References

1. G. Maxwell: Proving your Bitcoin reserves. Online discussion note, 2014.
2. Aayush Gupta, Kobi Gurkan: PLUME: An ECDSA Nullifier Scheme for Unique Pseudonymity within Zero Knowledge Proofs. *Cryptology ePrint Archive*, 2022
3. Y. Ji and K. Chalkias: Generalized proof of liabilities. In *ACM CCS*, 2021.
4. K. Chalkias, P. Chatzigiannis, and Y. Ji: Broken proofs of solvency in blockchain custodial wallets and exchanges. In *Financial Cryptography and Data Security Workshops (FC)*, pages 106–117, 2022.
5. F. Falzon, K. Elkhiyaoui, Y. Manevich, and A. Di Caro: Short privacy-preserving proofs of liabilities. In *ACM CCS*, 2023.
6. 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)*, pages 1453–1470, 2024.
7. J. Kuszmaul: Verkle trees. Online manuscript, 2018.
8. G. G. Dagher, B. Bünz, J. Bonneau, J. Clark, and D. Boneh: Provisions: Privacy-preserving proofs of solvency for Bitcoin exchanges. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 720–731, 2015.
9. F. Baldimtsi, P. Chatzigiannis, S. D. Gordon, P. H. Le, and D. McVicker: gOTzilla: Efficient disjunctive zero-knowledge proofs from MPC in the head, with application to proofs of assets in cryptocurrencies. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2022(4):229–249, 2022.
10. A. Kate, G. M. Zaverucha, and I. Goldberg: Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, 2010.
11. L. Nguyen: Accumulators from bilinear pairings and applications. In *CT-RSA*, pages 275–292, 2005.
12. B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell: Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symposium on Security and Privacy (SP)*, 2018.
13. R. C. Merkle: A digital signature based on a conventional encryption function. In *CRYPTO*, 1987.
14. G. Wood: Ethereum: A secure decentralised generalised transaction ledger. Ethereum Yellow Paper, 2014.
15. M. Campanelli, D. Fiore, and A. Querol: LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In *ACM CCS*, pages 2075–2092, 2019.
16. A. Zapico, V. Buterin, D. Khovratovich, M. Maller, A. Nitulescu, and M. Simkin: Caulk: Lookup arguments in sublinear time. In *ACM CCS*, pages 3121–3134, 2022.
17. C.-P. Schnorr: Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.
18. Ethereum Improvement Proposals: EIP-7928 component size and compression analysis: 1000 blocks (60M gas limit). Technical report, 2026. `https://eips.ethereum.org/assets/eip-7928/bal_size_analysis_60m_2026_10`.

19. T. P. Pedersen: Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology—CRYPTO '91*, pages 129–140, 1991.
20. A. Fiat and A. Shamir: How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—CRYPTO '86*, pages 186–194. Springer, 1987.
21. D. Pointcheval and J. Stern: Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13:361–396, 2000.
22. S. Goldwasser, S. Micali, and C. Rackoff: The knowledge complexity of interactive proof systems. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing (STOC)*, pages 291–304, 1985.
23. J. Groth: On the size of pairing-based non-interactive arguments. In *Advances in Cryptology—EUROCRYPT 2016*, pages 305–326. Springer, 2016.
24. J. T. Schwartz: Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.
25. J. Benaloh and M. de Mare: One-way accumulators: A decentralized alternative to digital signatures. In *Advances in Cryptology—EUROCRYPT '93*, pages 274–285. Springer, 1994.
26. J. Camenisch and A. Lysyanskaya: Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Advances in Cryptology—CRYPTO 2002*, pages 61–76. Springer, 2002.
27. I. Damgård and N. Triandopoulos: Supporting non-membership proofs with bilinear-map accumulators. Cryptology ePrint Archive, Report 2008/538, 2008.
28. M. J. Freedman, K. Nissim, and B. Pinkas: Efficient private matching and set intersection. In *Advances in Cryptology—EUROCRYPT 2004*, pages 1–19. Springer, 2004.
29. S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage: A fistful of bitcoins: Characterizing payments among men with no names. In *Proceedings of the 2013 ACM Internet Measurement Conference (IMC)*, pages 127–140, 2013.
30. D. Ron and A. Shamir: Quantitative analysis of the full bitcoin transaction graph. In *Financial Cryptography and Data Security*, pages 6–24. Springer, 2013.
31. M. Campanelli, D. Fiore, S. Han, J. Kim, D. Kolonelos, and H. Oh: Succinct zero-knowledge batch proofs for set accumulators. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 455–469, 2022.
32. S. Srinivasan, I. Karantaidou, F. Baldimtsi, and C. Papamanthou: Batching, aggregation, and zero-knowledge proofs in bilinear accumulators. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2719–2733, 2022.
33. J. Li, N. Li, and R. Xue: Universal accumulators with efficient nonmembership proofs. In *Applied Cryptography and Network Security (ACNS 2007)*, pages 253–269. Springer, 2007.
34. F. Boudot: Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology—EUROCRYPT 2000*, pages 431–444. Springer, 2000.
35. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai: Zero-knowledge from secure multiparty computation. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 21–30, 2007.
36. Succinct Labs: SP1: A zero-knowledge virtual machine for RISC-V programs. Software repository. `https://github.com/succinctlabs/sp1`. Accessed 22 November 2026.

37. Succinct Labs: Precompiles. SP1 documentation. https://docs.succinct.xyz/docs/sp1/optimizing-programs/precompiles. Accessed 22 November 2026.
38. Ethereum.org: Proof-of-stake (PoS). Ethereum developer documentation. https://ethereum.org/developers/docs/consensus-mechanisms/pos/. Accessed 22 November 2026.

A Formal Security Analysis

A.1 Assumptions

We distinguish the underlying cryptographic assumptions from the security guarantees required of the proof components and the conditions imposed by the application.

(A1) We assume that the discrete-logarithm problem is hard in the commitment group and that the Pedersen generators are generated without known discrete-logarithm relations. Under this assumption, Pedersen commitments are computationally binding. With uniformly sampled blinding randomness, they are perfectly hiding.

(A2) We assume the hardness assumptions underlying the binding and evaluation-soundness guarantees of the adopted KZG construction, for the degree bound and SRS used in our protocol. Under these assumptions, we rely on polynomial binding and sound single-point and multipoint openings.

(A3) The underlying proof systems provide completeness, knowledge soundness, and zero knowledge for the relations in which they are used. The security of the committed-evaluation protocols constructed from these components is established in Appendix B.

(A4) $\mathsf{OwnVerify}$ is sound as a proof of address control, and $\mathsf{ChainVerify}$ is sound for the balance authenticated by the stated blockchain state root.

(A5) $\mathsf{Enc}_{\mathbb{F}}$ is injective over valid addresses. All public bounds are chosen so that the required integer relations do not wrap around modulo q.

For privacy, we additionally assume a simulatable global setup and simulation algorithms for all zero-knowledge proof components. The simulators may be invoked adaptively under the same simulated setup and domain-separated programmable random oracles.

A.2 Soundness

Lemma 1 (Initialization Soundness).
If $\mathsf{VerInit}(vk, X_0, \pi_{\mathsf{init}}) = 1$*, then, except with negligible probability, there exists a witness* W_0 *such that* $\mathsf{ValidState}(X_0; W_0) = 1$.

Proof. By knowledge soundness of the zero-knowledge proof system used in the initialization, an accepting proof yields the hidden addresses, their ownership witnesses, their claimed balances, the masking scalar α_0, and an opening of the aggregate commitment C_0.

Soundness of $\mathsf{OwnVerify}$ guarantees that every extracted address is controlled by the prover. Soundness of $\mathsf{ChainVerify}$ guarantees that each extracted balance is the balance authenticated under the public state root sr_0. The initialization relation additionally enforces duplicate-freeness and

$$C_0 = \mathsf{Com}\left(\sum_{a \in S_0} \mathsf{bal}_{\mathsf{sr}_0}(a); r_0\right).$$

It remains to show that d_0 binds the same address set. Before the Fiat–Shamir challenge ζ_{init} is derived, the shape commitment fixes the addresses and the scalar α_0. The KZG-CVOpen proof shows that the polynomial committed by d_0 evaluates at ζ_{init} to the value used by the initialization proof, while the latter checks

$$P_0(\zeta_{\mathsf{init}}) = \alpha_0 \prod_{a \in S_0} \big(\zeta_{\mathsf{init}} - \mathsf{Enc}_{\mathbb{F}}(a)\big).$$

If the polynomial bound by d_0 were different from

$$\alpha_0 \prod_{a \in S_0} (T - \mathsf{Enc}_{\mathbb{F}}(a)),$$

the two degree-bounded polynomials could agree at the random challenge point only with negligible probability [24]. Therefore d_0 binds exactly the authenticated reserve set S_0, and all conditions of $\mathsf{ValidState}$ hold.

Lemma 2 (Update Soundness). *Let X_i be a valid public state. If an ordinary update from X_i to X_{i+1} is accepted, then, except with negligible probability, X_{i+1} is also a valid public state.*

Proof. Let S_i and P_i be the reserve set and masked set polynomial bound by the valid predecessor state. Let $\Delta_i = ((a_j, \delta_j))_{j=1}^{m}$ be the canonical balance-delta list derived by the verifier, and set $x_j = \mathsf{Enc}_{\mathbb{F}}(a_j)$.

Knowledge soundness of Batch-KZG-CVOpen fixes an ordered vector

$$\mathbf{y} = (y_1, \ldots, y_m)$$

such that

$$y_j = P_i(x_j)$$

for every j. The R1CS proof uses the same vector commitment of $\mathbf{y}$. Binding property of Pedersen vector commitment therefore forces the R1CS witness to use the same values y_j.

For every coordinate, the R1CS relation enforces

$$u_j y_j = 0, \qquad y_j z_j = 1 - u_j.$$

Hence

$$u_j = 1 \iff y_j = 0 \iff a_j \in S_i.$$

The same membership vector is bound to the inner-product proof. Soundness of this proof and binding of the corresponding commitments give

$$\Delta sum_i = \sum_{j=1}^{m} u_j \delta_j = \sum_{j=1}^{m} \mathbf{1}[a_j \in S_i]\delta_j.$$

The public bounds ensure that this is also the intended equality over the integers.

The verifier checks

$$C_{i+1} = C_i + C_D,$$

where C_D commits to Δsum_i. Therefore the value bound by C_{i+1} is

$$sum_{i+1} = sum_i + \Delta sum_i.$$

Because the public delta list is complete, this value is exactly

$$\sum_{a \in S_i} \mathsf{bal}_{\mathsf{sr}_{i+1}}(a).$$

Finally, an ordinary update leaves both the digest and the set size unchanged. Hence the successor state binds the same reserve set, now with its correct aggregate balance under sr_{i+1}. Thus $\mathsf{ValidState}(X_{i+1}; W_{i+1}) = 1$ for an appropriate successor witness W_{i+1}.

Lemma 3 (Insertion Soundness). *Let X_i be a valid public state. If an insertion producing X_i' is accepted, then, except with negligible probability, X_i' is also a valid public state.*

Proof. Let S_i and P_i be the reserve set and masked set polynomial bound by the predecessor state. Let $a^\star$ be a new address, $b^\star$ be its balance under state root sr, and $u = \mathsf{Enc}_{\mathbb{F}}(a^\star)$.

Soundness of HandleAuth and KZG-CPVOpen yields a value y and a nonzero scalar β satisfying

$$R_u = [\beta(\tau - u)]_2, \qquad y = P_i(u).$$

Soundness of the committed-nonzero proof gives

$$y \neq 0.$$

Therefore u is not a root of the old set polynomial.

Soundness of $\mathsf{OwnVerify}$ shows that the prover controls $a^\star$, and soundness of $\mathsf{ChainVerify}$ gives

$$b^\star = \mathsf{bal}_{\mathsf{sr}_i}(a^\star).$$

The verifier also checks

$$e(d_i', [1]_2) = e(d_i, R_u).$$

Together with the authenticated handle relation, this gives

$$d_i' = [\beta P_i(\tau)(\tau - u)]_1.$$

Hence d'_i binds the set polynomial whose roots are exactly

$$S'_i = S_i \cup \{a^\star\}.$$

Finally,

$$C'_i = C_i + C_b$$

and binding of C_b imply that the successor aggregate is

$$sum'_i = sum_i + b^\star = \sum_{a \in S'_i} \mathsf{bal}_{\mathsf{sr}_i}(a).$$

The verifier also checks $n'_i = n_i + 1$. Therefore the successor state satisfies all conditions of ValidState.

Theorem 1 (State Soundness). *Under assumptions (A1)–(A5), a polynomial-time malicious prover can cause the verifier to accept an invalid public state only with negligible probability. Moreover, starting from a valid state, it cannot produce an accepted transition whose successor state is invalid, except with negligible probability.*

Proof. Lemma 1 shows that every accepted initialization produces a valid state. Lemmas 2 and 3 show that each supported state transition preserves validity.

Therefore, starting from an accepted initialization, induction over any polynomial-length sequence of accepted updates and insertions shows that every resulting public state is valid, except with the sum of the negligible failure probabilities of the underlying components. A union bound over polynomially many transitions keeps this probability negligible.

Theorem 2 (Range Proof Soundness). *For every valid public state, a polynomial-time malicious prover can produce an accepted range proof with an incorrect comparison result only with negligible probability.*

Proof. Let C_i bind the aggregate value sum_i of a valid state. Binding of C_i prevents the prover from using a different aggregate value, and range-proof soundness together with the public integer bounds rules out modular wraparound. Therefore an accepted proof gives the correct comparison result except with negligible probability. The committed-liability case follows identically.

A.3 Privacy

We first recall the distribution of a freshly masked set digest. Let

$$P_S(T) = \alpha F_S(T), \qquad \alpha \xleftarrow{\$} \mathbb{F}_q^*.$$

Conditioned on $F_S(\tau) \neq 0$,

$$[P_S(\tau)]_1 = [\alpha F_S(\tau)]_1$$

is uniform over $G_1 \setminus \{O\}$, since multiplication by the fixed nonzero value $F_S(\tau)$ is a permutation of $\mathbb{F}_q^*$. If the address set is independent of τ, the exceptional event satisfies

$$\Pr[F_S(\tau) = 0] \leq \frac{|S|}{q}.$$

Theorem 3 (State Privacy). *Under the hiding of Pedersen commitments, the hiding property of the masked set digest, and the zero-knowledge simulation guarantees of the underlying proof components, the persistent public states and the initialization, update, and insertion proof transcripts of Pethoa are computationally simulatable from the defined public leakage.*

Proof. We construct a stateful simulator $\mathsf{Sim}_{\mathsf{state}}$ that maintains its own simulated public state together with auxiliary dummy values and openings used only for simulation. The simulator receives no reserve addresses, individual balances, membership vectors, or real aggregate values.

For the commit-and-prove components, we use their simulation algorithms under the simulated global setup. These simulators take the fixed public commitments and other public inputs as their statements and generate the corresponding proof transcripts without their openings or witnesses. For the transparent Bulletproof components, the simulator instead maintains dummy committed values satisfying the corresponding relations, so that the simulated Bulletproof statements are valid.

Initialization Given only the initialization leakage

$$(\mathsf{init}, \mathsf{sr}_0, n_0),$$

the simulator samples

$$\widetilde{d}_0 \xleftarrow{\$} G_1 \setminus \{O\}.$$

It also chooses an arbitrary dummy aggregate value $\widetilde{sum}_0$ in the admissible balance domain and fresh randomness

$$\widetilde{r}_0 \xleftarrow{\$} \mathbb{F}_q,$$

and defines

$$\widetilde{C}_0 = \mathsf{Com}(\widetilde{sum}_0; \widetilde{r}_0).$$

The simulator stores $(\widetilde{sum}_0, \widetilde{r}_0)$ as part of its private simulation state.

It then invokes the simulators of KZG-CVOpen and of the initialization commit-and-prove system on the fixed simulated public objects to obtain the corresponding proof transcripts. The resulting simulated public state is

$$\widetilde{X}_0 = (\mathsf{sr}_0, \widetilde{d}_0, \widetilde{C}_0, n_0).$$

Conditioned on

$$F_{S_0}(\tau) \neq 0,$$

the real digest

$$d_0 = [\alpha_0 F_{S_0}(\tau)]_1$$

is uniform over $G_1 \backslash \{O\}$, and hence $\widetilde{d}_0$ has exactly the same distribution. Moreover, by perfect hiding of Pedersen commitments,

$$\mathsf{Com}(sum_0; r_0) \equiv \mathsf{Com}(\widetilde{sum}_0; \widetilde{r}_0),$$

so replacing the real aggregate by the dummy aggregate does not change the distribution of the public commitment. Finally, the simulated proof components are computationally indistinguishable from the real ones by the zero-knowledge guarantees of the corresponding commit-and-prove protocols.

Ordinary update Suppose the simulator has already generated

$$\widetilde{X}_i = (\mathsf{sr}_i, \widetilde{d}_i, \widetilde{C}_i, n_i)$$

and privately maintains an opening

$$\widetilde{C}_i = \mathsf{Com}(\widetilde{sum}_i; \widetilde{r}_i).$$

Given the public update leakage

$$(\mathsf{update}, \mathit{\Gamma}_i),$$

where

$$\mathit{\Gamma}_i = ((a_j, \delta_j))_{j=1}^m,$$

the simulator preserves the public set state by setting

$$\widetilde{d}_{i+1} = \widetilde{d}_i, \qquad n_{i+1} = n_i.$$

The simulator next chooses dummy values for the committed quantities. In particular, it chooses a dummy membership vector $\widetilde{\mathbf{u}}$ and defines

$$\widetilde{\mathit{\Delta sum}}_i = \langle \widetilde{\mathbf{u}}, \boldsymbol{\delta}_i \rangle .$$

For example, it may take

$$\widetilde{\mathbf{u}} = \mathbf{0}, \qquad \widetilde{\mathit{\Delta sum}}_i = 0.$$

It samples fresh openings and forms

$$\widetilde{C}_U = \mathsf{Com}(\widetilde{\mathbf{u}}; \widetilde{r}_U),$$

and

$$\widetilde{C}_D = \mathsf{Com}(\widetilde{\mathit{\Delta sum}}_i; \widetilde{r}_D).$$

Thus the public statement of the Bulletproof inner-product component is a valid statement by construction.

The simulator updates its dummy aggregate state according to

$$\widetilde{sum}_{i+1} = \widetilde{sum}_i + \widetilde{\Delta sum}_i,$$

and

$$\widetilde{r}_{i+1} = \widetilde{r}_i + \widetilde{r}_D.$$

It therefore sets

$$\widetilde{C}_{i+1} = \widetilde{C}_i + \widetilde{C}_D = \mathsf{Com}(\widetilde{sum}_{i+1}; \widetilde{r}_{i+1}).$$

Hence the public commitment-update equation is preserved exactly.

For the hidden polynomial-evaluation vector, the simulator chooses dummy values satisfying the committed-R1CS relation. For example, when $\widetilde{\mathbf{u}} = \mathbf{0}$, it may choose

$$\widetilde{y}_j = 1, \qquad \widetilde{z}_j = 1$$

for every j, which gives

$$\widetilde{u}_j \widetilde{y}_j = 0, \qquad \widetilde{y}_j \widetilde{z}_j = 1 - \widetilde{u}_j.$$

It then samples fresh randomness and forms

$$\widetilde{D}_Y = \mathsf{Com}(\widetilde{\mathbf{y}}; \widetilde{r}_Y).$$

The simulator invokes the committed-R1CS simulator on $\widetilde{D}_Y$ and $\widetilde{C}_U$, and invokes the Bulletproof inner-product simulator on $\widetilde{C}_U$, $\widetilde{C}_D$, and the public vector $\boldsymbol{\delta}_i$. Since the latter statement is valid by construction, its standard zero-knowledge guarantee applies.

Finally, the simulator invokes the simulator of Batch-KZG-CVOpen on the fixed public objects

$$(\widetilde{d}_i, (x_1, \ldots, x_m), \widetilde{D}_Y), \qquad x_j = \mathsf{Enc}_{\mathbb{F}}(a_j).$$

This component is simulated using its commit-and-prove simulation trapdoor and therefore does not require the simulator to know the real polynomial evaluations or the opening of $\widetilde{D}_Y$ corresponding to evaluations of the polynomial bound by $\widetilde{d}_i$.

Consequently, the simulator produces the successor state

$$\widetilde{X}_{i+1} = (\mathsf{sr}_{i+1}, \widetilde{d}_{i+1}, \widetilde{C}_{i+1}, n_i)$$

together with an update transcript having all of the same publicly checked cross-component and cross-state relations as a real execution, without learning the actual reserve-set intersection or aggregate balance change.

Insertion Suppose the simulator currently maintains

$$\widetilde{X}_i = (\mathsf{sr}_i, \widetilde{d}_i, \widetilde{C}_i, n_i)$$

and an opening

$$\widetilde{C}_i = \mathsf{Com}(\widetilde{sum}_i; \widetilde{r}_i).$$

Given the public leakage that an insertion occurs, the simulator samples

$$t \xleftarrow{\$} \mathbb{F}_q^*$$

and defines

$$\widetilde{R}_u = [t]_2, \qquad \widetilde{d}'_i = t\widetilde{d}_i.$$

Thus the public digest-transition pairing relation is satisfied exactly.

It chooses arbitrary dummy committed values for the hidden insertion data. In particular, it chooses a dummy point $\widetilde{u}$, a nonzero dummy evaluation $\widetilde{y}$, and a dummy inserted balance $\widetilde{b}$ in the admissible balance domain, together with fresh Pedersen randomness, and computes

$$\widetilde{C}_u = \mathsf{Com}(\widetilde{u}; \widetilde{r}_u),$$

$$\widetilde{C}_y = \mathsf{Com}(\widetilde{y}; \widetilde{r}_y),$$

and

$$\widetilde{C}_b = \mathsf{Com}(\widetilde{b}; \widetilde{r}_b).$$

The dummy aggregate state is updated as

$$\widetilde{sum}'_i = \widetilde{sum}_i + \widetilde{b}, \qquad \widetilde{r}'_i = \widetilde{r}_i + \widetilde{r}_b,$$

and the simulator sets

$$\widetilde{C}'_i = \widetilde{C}_i + \widetilde{C}_b = \mathsf{Com}(\widetilde{sum}'_i; \widetilde{r}'_i).$$

It also sets

$$n'_i = n_i + 1.$$

The simulator then invokes the simulators of HandleAuth, KZG-CPVOpen, the committed-nonzero proof, and the insertion authentication proof on these fixed public commitments and public state values. These components are simulated in programmable random oracle model.

In a real insertion,

$$R_u = [\beta(\tau - u)]_2$$

and

$$d'_i = \beta(\tau - u)d_i.$$

Writing

$$t = \beta(\tau - u),$$

and conditioning on $u \neq \tau$, multiplication by the fixed nonzero value $\tau - u$ maps a uniform $\beta \in \mathbb{F}_q^*$ to a uniform $t \in \mathbb{F}_q^*$. Therefore

$$(R_u, d_i') = ([t]_2, t\, d_i)$$

has exactly the same distribution as

$$(\widetilde{R}_u, \widetilde{d}_i').$$

The remaining hidden insertion values occur only through hiding commitments and zero-knowledge proof components.

Hybrid argument Consider a real execution consisting of an initialization followed by a polynomial number of ordinary updates and insertions.

We first replace, in chronological order, the commit-and-prove transcripts by outputs of their corresponding simulators under the simulated global setup. By the multi-theorem zero-knowledge guarantees of these components, this changes the adversary's view by at most a negligible amount.

We next replace the hidden values committed in Pedersen commitments by the dummy values maintained by the state simulator. Each commitment is generated with fresh randomness, and Pedersen commitments are perfectly hiding. The replacements therefore preserve the distributions of the individual commitments. The simulator maintains the same shared commitments across proof components and explicitly preserves all public algebraic relations between consecutive states, such as

$$\widetilde{C}_{i+1} = \widetilde{C}_i + \widetilde{C}_D.$$

For the transparent Bulletproof components, the dummy values are chosen so that their public statements remain valid, and their proof transcripts can therefore be replaced by outputs of the corresponding zero-knowledge simulators.

Finally, every freshly generated masked set digest is replaced by an independent uniform element of $G_1 \setminus \{O\}$. Conditioned on

$$F_{S_i}(\tau) \neq 0,$$

this replacement is exact. Over all polynomially many freshly generated digests, the probability of the exceptional events is at most

$$\sum_{i \in I_{\mathsf{fresh}}} \frac{|S_i|}{q},$$

which is negligible.

After these replacements, the resulting joint distribution is exactly the one generated by $\mathsf{Sim}_{\mathsf{state}}$ from the allowed public leakage. Therefore, for every PPT adversary $\mathcal{A}$,

$$\mathsf{View}_{\mathcal{A}}^{\mathsf{real}} \approx_c \mathsf{View}_{\mathcal{A},\mathsf{Sim}_{\mathsf{state}}}^{\mathsf{ideal}},$$

which proves state privacy.

Theorem 4 (Range Proof Privacy). *For any fixed valid public state X_i, a range proof reveals no information about the committed aggregate asset value beyond the public threshold statement and its Boolean comparison result.*

Proof. Let

$$X_i = (\mathsf{sr}_i, d_i, C_i, n_i)$$

be a fixed valid public state, and let t be a public threshold. The public Boolean result is

$$b = \mathbf{1}[sum_i \geq t].$$

Hence the public statement of the range proof contains

$$(X_i, t, b),$$

while the aggregate value sum_i and the opening of C_i remain private witnesses.

Since the range proof system is zero-knowledge, there exists a PPT simulator

$$\mathsf{Sim}_{\mathsf{range}}$$

that, given only the public statement

$$(X_i, t, b),$$

produces a transcript

$$\widetilde{\pi}_{\mathsf{range}}$$

computationally indistinguishable from a real range proof.

Therefore, conditioned on the already public state X_i, the threshold proof reveals no information about sum_i or the opening of C_i beyond the disclosed comparison result b. The same argument applies to a comparison against a committed liability value.

B Variants of KZG Opening Protocol

This appendix gives the protocols for the committed-evaluation interfaces used in Pethoa. Each protocol links a KZG opening to Pedersen-committed data. We first fix the notation, then give the algorithms and their security arguments.

Common notation All scalars lie in $\mathbb{F}_q$. The pairing is $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, with additive notation in $\mathbb{G}_1, \mathbb{G}_2$ and multiplicative notation in $\mathbb{G}_T$. We write $[a]_j = aG_j$, $\mathcal{O}$ for the source-group identity, and $g_T = e([1]_1, [1]_2)$. The KZG SRS encodes powers of a secret τ up to degree D_{srs}. The polynomial commitment is $C = [f(\tau)]_1$.

Scalar and ordered vector commitments use the fixed bases

$$\mathsf{Com}(y; \rho) = yV_E + \rho H_E, \qquad \mathsf{Com}(\mathbf{y}; \rho) = \sum_{j=1}^{m} y_j G_j^Y + \rho H_Y.$$

We use $C_y = \mathsf{Com}(y; \rho_y)$ for a hidden value, $C_u = \mathsf{Com}(u; \rho_u)$ for a hidden point, and $D_Y = \mathsf{Com}(\mathbf{y}; \rho_Y)$ for a hidden evaluation vector. The commitment bases, vector length, and coordinate order are public.

For a public point z, define $D_z = [\tau - z]_1$ and $\widehat{D}_z = [\tau - z]_2$. For distinct public points $\mathbf{z} = (z_1, \ldots, z_m)$, let $Z(T) = \prod_j (T - z_j)$ and let $I_{\mathbf{v}}$ interpolate an ordered vector $\mathbf{v}$ at these points: $I_{\mathbf{v}}(z_j) = v_j$ and $\deg I_{\mathbf{v}} < m$. The symbol W denotes the ordinary KZG quotient commitment:

$$W = \begin{cases} [(f(T) - y)/(T - z)]_1 & \text{for one point,} \\ [(f(T) - I_{\mathbf{y}}(T))/Z(T)]_1 & \text{for a batch.} \end{cases}$$

Here brackets around a polynomial mean evaluation at τ followed by group encoding. All polynomial degrees must fit the SRS. For $m = 0$, we use $Z = 1$ and $I_\emptyset = 0$.

For a hidden point u, choose $\beta \in \mathbb{F}_q^*$ and define

$$R_u = [\beta(\tau - u)]_2, \qquad \widetilde{W} = [\beta^{-1} Q_u(\tau)]_1,$$

where $Q_u(T) = (f(T) - y)/(T - u)$. The handle R_u is public; u and β are hidden. Handle authentication uses $\mu = \beta u$ and auxiliary commitments $C_\beta = \mathsf{Com}(\beta; \rho_\beta)$ and $C_\mu = \mathsf{Com}(\mu; \rho_\mu)$. A committed multiplication proof $\pi_\times$ proves $\mu = \beta u$ using these commitments and C_u.

Challenges and security conventions The algorithms use an interactive challenge $c \leftarrow \mathbb{F}_q$. Their Fiat–Shamir versions compute $c = H_{\mathsf{FS}}(\mathsf{tag} \parallel x \parallel A \parallel \mathsf{ctx})$, where x is the complete public statement, A contains the first messages, and ctx binds the surrounding session. Each protocol uses a different tag. Parameters and encodings are fixed by the protocol identifier. Verifiers check canonical subgroup encodings and recompute the challenge.

The soundness arguments below extract witnesses from two accepting transcripts with the same first messages and different challenges [17]. Write $\Delta c = c_1 - c_2$ and $\Delta v = v_1 - v_2$ for their differences. Division of a group element by Δc means multiplication by $(\Delta c)^{-1}$. The privacy arguments simulate transcripts from fixed public inputs, without commitment openings. They establish honest-verifier zero knowledge; the non-interactive versions use programmable random-oracle simulation [21] and the extraction assumptions stated in Appendix A. The multiplication proof is assumed to provide knowledge soundness and zero knowledge for fixed commitments under the same setup model. These component arguments do not by themselves establish privacy of the surrounding sequence of Pethoa states, which is proved in Appendix A.

B.1 Public Point and Committed Value: KZG-CVOpen

Algorithm 1 KZG-CVOpen

Input: Public (C, z, C_y); witness (y, ρ_y, W).

1: Prover samples $a, b, t \leftarrow \mathbb{F}_q$.
2: Send $A_P = aV_E + bH_E$ and $A_C = a[1]_1 + tD_z$.
3: Receive challenge $c \leftarrow \mathbb{F}_q$.
4: Return $s_y = a + cy$, $s_\rho = b + c\rho_y$, and $\Pi = t[1]_1 + cW$.
5: Verifier accepts iff both equations below hold:
$A_P + cC_y = s_y V_E + s_\rho H_E$,
$e(A_C + cC - s_y[1]_1, [1]_2) = e(\Pi, \widehat{D}_z)$.

Completeness and soundness The protocol proves that C_y commits to the evaluation of C at the public point z. For an honest prover, $C - y[1]_1 = [(T-z)Q_z(T)]_1$. Thus the second verification equation is the ordinary opening relation multiplied by the challenge and combined with the random mask t. The first equation follows from Pedersen homomorphism.

To prove soundness, take two accepting transcripts with the same (A_P, A_C) and different challenges. Their responses yield

$$y = \frac{\Delta s_y}{\Delta c}, \qquad \rho_y = \frac{\Delta s_\rho}{\Delta c}, \qquad W = \frac{\Delta \Pi}{\Delta c}.$$

Subtracting the Pedersen checks gives $C_y = \mathsf{Com}(y; \rho_y)$. Taking the ratio of the pairing checks and removing the common factor Δc gives

$$e(C - y[1]_1, [1]_2) = e(W, [\tau - z]_2).$$

The extracted value therefore both opens C_y and has a valid KZG opening. KZG evaluation soundness implies $y = f(z)$.

Privacy Given only (C, z, C_y), the simulator samples $c, s_y, s_\rho, t \leftarrow \mathbb{F}_q$ and sets

$$\begin{aligned} \Pi &= t[1]_1, \\ A_P &= s_y V_E + s_\rho H_E - cC_y, \\ A_C &= s_y[1]_1 + tD_z - cC. \end{aligned}$$

Both verifier checks now hold by construction. For a fixed challenge in a real execution, s_y, s_ρ, and Π are independently uniform because their masks are independently uniform. The verification equations determine the corresponding first messages, exactly as above. Hence the simulated transcript has the same distribution as an honest transcript for a valid statement. For Fiat–Shamir, the simulator programs the query containing $(C, z, C_y, A_P, A_C, \mathsf{ctx})$ to return c.

B.2 Public Points and a Committed Vector: Batch-KZG-CVOpen

Algorithm 2 Batch-KZG-CVOpen

Input: Public $(C, \mathbf{z}, D_Y)$; witness $(\mathbf{y}, \rho_Y, W)$.

1: Prover samples $\mathbf{a} \leftarrow \mathbb{F}_q^m$, $r, t \leftarrow \mathbb{F}_q$.
2: Compute $I_{\mathbf{a}}$ interpolating $\mathbf{a}$ at $\mathbf{z}$.
3: Send $A_Y = \mathsf{Com}(\mathbf{a}; r)$ and $A_C = [I_{\mathbf{a}}(\tau)]_1 + t[Z(\tau)]_1$.
4: Receive challenge $c \leftarrow \mathbb{F}_q$.
5: Return $\mathbf{s} = \mathbf{a} + c\mathbf{y}$, $s_\rho = r + c\rho_Y$, and $\mathit{\Pi} = t[1]_1 + cW$.
6: Verifier interpolates S with $S(z_j) = s_j$ for each j.
7: Accept iff the points are distinct and both checks hold:
$A_Y + cD_Y = \mathsf{Com}(\mathbf{s}; s_\rho)$,
$e(A_C + cC - [S(\tau)]_1, [1]_2) = e(\mathit{\Pi}, [Z(\tau)]_2)$.

Completeness and soundness The protocol proves $y_j = f(z_j)$ for every coordinate in D_Y. For an honest execution, interpolation is linear, so the verifier's polynomial satisfies $S = I_{\mathbf{a}} + cI_{\mathbf{y}}$. Consequently,

$$A_C + cC - [S(\tau)]_1 = t[Z(\tau)]_1 + c[f - I_{\mathbf{y}}]_1,$$

which gives the pairing check because $f - I_{\mathbf{y}} = QZ$. The Pedersen check follows from the same linear combination.

From two accepting transcripts, extract $\mathbf{y} = \mathit{\Delta}\mathbf{s}/\mathit{\Delta}c$, $\rho_Y = \mathit{\Delta}s_\rho/\mathit{\Delta}c$, and $W = \mathit{\Delta\Pi}/\mathit{\Delta}c$. Linearity of interpolation also applies to the difference between the two verifier polynomials. The verification equations therefore give

$$\begin{aligned} D_Y &= \mathsf{Com}(\mathbf{y}; \rho_Y), \\ e(C - [I_{\mathbf{y}}(\tau)]_1, [1]_2) &= e(W, [Z(\tau)]_2). \end{aligned}$$

Batch-opening soundness implies $y_j = f(z_j)$ for every j. The same response s_j is used at point z_j and with generator G_j^Y, so this argument also binds the order of the evaluations. For an empty list there are no evaluation claims; the surrounding aggregation proof is responsible for enforcing zero balance change.

Privacy The simulator samples $c, s_\rho, t \leftarrow \mathbb{F}_q$ and $\mathbf{s} \leftarrow \mathbb{F}_q^m$, computes $S = I_{\mathbf{s}}$, and sets

$$\begin{aligned} \mathit{\Pi} &= t[1]_1, \\ A_Y &= \mathsf{Com}(\mathbf{s}; s_\rho) - cD_Y, \\ A_C &= [S(\tau)]_1 + t[Z(\tau)]_1 - cC. \end{aligned}$$

Everything on the right is computable from public inputs and the sampled values. Both checks hold. As in the single-point protocol, the honest responses are independently uniform for each fixed challenge, and the first messages are reconstructed from them. Thus simulation preserves the transcript distribution without knowing any evaluation or the opening of D_Y. The Fiat–Shamir simulator programs the batch-tagged query on $(C, \mathbf{z}, D_Y, A_Y, A_C, \mathsf{ctx})$ to return c.

B.3 Hidden Point and Committed Value

The hidden-point interface has two components. HandleAuth binds R_u to the point in C_u; KZG-CPVOpen proves an evaluation relative to that same handle. Both components are necessary.

Algorithm 3 HandleAuth

Input: Public (C_u, R_u); witness (u, ρ_u, β) with $\beta \neq 0$.

1: Prover sets $\mu = \beta u$ and forms C_β, C_μ with fresh openings.
2: Generate $\pi_\times$ for $\mu = \beta u$, bound to $(C_u, R_u, C_\beta, C_\mu, \mathsf{ctx})$.
3: Sample $b, h, r_b, r_h \leftarrow \mathbb{F}_q$.
4: Send $C_\beta, C_\mu, \pi_\times$ and first messages $A_\beta = bV_E + r_bH_E$, $A_\mu = hV_E + r_hH_E$, $R' = b[\tau]_2 - h[1]_2$.
5: Receive challenge $c \leftarrow \mathbb{F}_q$.
6: Return $s_\beta = b + c\beta$, $s_\mu = h + c\mu$, $s_{\rho_\beta} = r_b + c\rho_\beta$, $s_{\rho_\mu} = r_h + c\rho_\mu$.
7: Verifier checks $R_u \neq \mathcal{O}$ and $\pi_\times$ on the same commitments, then checks:
$A_\beta + cC_\beta = s_\beta V_E + s_{\rho_\beta} H_E$,
$A_\mu + cC_\mu = s_\mu V_E + s_{\rho_\mu} H_E$,
$R' + cR_u = s_\beta[\tau]_2 - s_\mu[1]_2$.

HandleAuth The multiplication proof has its own domain-separated challenge. The linear proof treats $(C_\beta, C_\mu, \pi_\times)$ as additional public inputs when deriving its challenge. It does not require the multiplication protocol to share that challenge.

Completeness and soundness For honest witnesses, the first two checks follow from commitment homomorphism. The last follows from $R_u = \beta[\tau]_2 - \mu[1]_2$ and $\mu = \beta u$.

Two accepting linear transcripts extract $\beta = \Delta s_\beta / \Delta c$, $\mu = \Delta s_\mu / \Delta c$, and their commitment openings. The multiplication argument extracts openings satisfying $\mu = \beta u$. Binding of the shared commitments makes these values agree with those extracted from the linear argument. Subtracting the handle checks then gives

$$R_u = \beta[\tau]_2 - \mu[1]_2 = [\beta(\tau - u)]_2.$$

If $\beta = 0$, then $\mu = 0$ and $R_u = \mathcal{O}$, which the verifier rejects. Thus an accepting proof authenticates a nonzero-randomized handle for the point committed in C_u.

Privacy For fixed public commitments C_u, C_β, C_μ and handle R_u, first simulate $\pi_\times$ using the underlying committed multiplication simulator. Next sample c and the four scalar responses uniformly, and set

$$\begin{aligned} A_\beta &= s_\beta V_E + s_{\rho_\beta} H_E - cC_\beta, \\ A_\mu &= s_\mu V_E + s_{\rho_\mu} H_E - cC_\mu, \\ R' &= s_\beta[\tau]_2 - s_\mu[1]_2 - cR_u. \end{aligned}$$

These first messages satisfy all three linear checks. In an honest execution the four responses are independently uniform for a fixed challenge; the equations above reconstruct their first messages. The linear transcript therefore has the honest distribution, while privacy of $\pi_\times$ follows from its assumed simulator. Program the handle-tagged query, including the auxiliary commitments and $\pi_\times$, to return c.

This proves privacy of the proof transcripts for fixed auxiliary commitments. When these commitments are generated as part of HandleAuth, their fresh independent Pedersen randomness hides β and μ. The handle itself is uniform over nonidentity elements of $\mathbb{G}_2$ for a fresh uniform $\beta \neq 0$ and fixed $u \neq \tau$. This single-handle observation does not replace the joint-state privacy argument for insertion.

Algorithm 4 KZG-CPVOpen

Input: Public (C, C_y, R_u); witness $(y, \rho_y, \widetilde{W})$.

1: Prover samples $a, r \leftarrow \mathbb{F}_q$, $A_W \leftarrow \mathbb{G}_1$.
2: Send $A_y = aV_E + rH_E$ and $A_E = g_T^a e(A_W, R_u)$.
3: Receive challenge $c \leftarrow \mathbb{F}_q$.
4: Return $s_y = a + cy$, $s_\rho = r + c\rho_y$, $Z_W = A_W + c\widetilde{W}$.
5: Verifier rejects if $R_u = \mathcal{O}$; otherwise checks:
$s_y V_E + s_\rho H_E = A_y + cC_y$,
$g_T^{s_y} e(Z_W, R_u) = A_E e(C, [1]_2)^c$.

KZG-CPVOpen relative to an authenticated handle

Completeness and soundness The protocol proves knowledge of an opening of C_y and a handle-relative KZG witness. For an honest prover, the two randomizers cancel:

$$e(C - y[1]_1, [1]_2) = e(\widetilde{W}, R_u).$$

Substituting the responses into the verifier equations proves completeness.

Two accepting transcripts with identical first messages extract $y = \Delta s_y / \Delta c$, $\rho_y = \Delta s_\rho / \Delta c$, and $\widetilde{W} = \Delta Z_W / \Delta c$. Subtracting the first checks and taking the ratio of the second checks gives

$$C_y = \mathsf{Com}(y; \rho_y),$$
$$e(C - y[1]_1, [1]_2) = e(\widetilde{W}, R_u).$$

This establishes the handle-relative relation. It does not, on its own, identify the point encoded in R_u; that is the role of HandleAuth.

Privacy Given (C, C_y, R_u), sample $c, s_y, s_\rho \leftarrow \mathbb{F}_q$ and $Z_W \leftarrow \mathbb{G}_1$, then define

$$A_y = s_y V_E + s_\rho H_E - cC_y,$$
$$A_E = g_T^{s_y} e(Z_W, R_u) e(C, [1]_2)^{-c}.$$

The verifier checks hold directly. For a fixed challenge, the honest scalar responses are uniform, and Z_W is independently uniform because A_W is uniform. Their reconstructed first messages therefore have the honest distribution. Program the CPVOpen-tagged query on $(C, C_y, R_u, A_y, A_E, \mathsf{ctx})$ to return c. No hidden point, evaluation, or quotient witness is needed.

Combined guarantee When both protocols accept on the same handle, extraction gives $R_u = [\beta(\tau - u)]_2$, where u opens C_u and $\beta \neq 0$. The extracted evaluation relation becomes

$$e(C - y[1]_1, [1]_2) = e(\beta\widetilde{W}, [\tau - u]_2).$$

Thus $\beta\widetilde{W}$ is an ordinary KZG witness at u. Evaluation soundness implies $y = f(u)$, and the Pedersen checks bind u and y to their respective commitments. Under the stated composition assumptions, the two simulators hide the witnesses while preserving these shared public inputs.

Pethoa additionally needs $y \neq 0$ for insertion. This is a separate committed arithmetic proof of $y\nu = 1$ for a hidden inverse ν, linked to the same C_y. Its soundness and privacy follow from the underlying committed arithmetic proof. For a set polynomial, this nonzero check establishes prior nonmembership.

C Ethereum MPT Verification

The main experiments use a simplified Merkle tree to represent the blockchain state. To evaluate the cost of using Ethereum's native state tree, we additionally benchmark the verification of Ethereum Merkle–Patricia trie (MPT) state proofs [14] in SP1.

The experiment uses the same hardware and software environment as the initialization experiment. We construct one synthetic Ethereum MPT and generate account proofs for $n \in \{2^{10}, 2^{11}, 2^{12}, 2^{13}\}$ accounts. All account proofs are verified against the same state root. For each value of n, SP1 verifies the corresponding MPT proofs and generates one compressed proof. The reported prover time only measures this MPT-proof verification workload and does not include the other steps of Pethoa initialization.

Table 3 shows that the prover time increases with the number of account proofs, from 18.847 s for 2^{10} accounts to 2.07 min for 2^{13} accounts. In contrast, the verifier time remains around 50 ms and the SP1 proof size remains 1.21 MiB for all tested values of n.

D Integer Bounds

The proof systems used in Pethoa perform arithmetic over the prime field $\mathbb{F}_q$, whereas account balances, balance changes, and thresholds have their usual

Table 3. Cost of verifying Ethereum MPT state proofs in SP1.

n	Prover time	Verifier time	Proof size
2^{10}	18.847 s	51.098 ms	1.21 MiB
2^{11}	33.973 s	51.374 ms	1.21 MiB
2^{12}	1.06 min	50.807 ms	1.21 MiB
2^{13}	2.07 min	50.212 ms	1.21 MiB

integer meanings. We therefore restrict all application-level values to fixed public ranges that are well inside the field. This ensures that an accepted field equation cannot be satisfied only because of modular wraparound.

A simple sufficient choice is

$$0 \le B_{\mathsf{sum}} < \frac{q}{4}, \qquad 0 \le B_{\mathsf{upd}} < \frac{q}{4}, \qquad 0 \le B_{\mathsf{cmp}} < \frac{q}{2}.$$

Since the reserve set contains at most D addresses, the per-account balance bound is chosen such that

$$0 \le B_{\mathsf{bal}} \le \frac{B_{\mathsf{sum}}}{D} < \frac{q}{4D}.$$

For one ordinary update, the selected balance change is required to satisfy

$$-B_{\mathsf{upd}} \le \Delta sum_i \le B_{\mathsf{upd}}.$$

We use the canonical signed representation in $\mathbb{F}_q$: a non-negative integer x is represented by x, while a negative integer $-x$ is represented by $q - x$.

Since

$$B_{\mathsf{upd}} < \frac{q}{4},$$

the positive interval

$$[0, B_{\mathsf{upd}}]$$

and the negative interval

$$[q - B_{\mathsf{upd}}, q - 1]$$

are disjoint and separated by more than half of the field. Hence every allowed field element has a unique signed-integer interpretation.

The homomorphic commitment check implies

$$sum_i - sum_{i-1} - \Delta sum_i \equiv 0 \pmod{q}.$$

The largest magnitude relevant to this relation is bounded by

$$2B_{\mathsf{sum}} + B_{\mathsf{upd}}.$$

Under the ranges above,

$$2B_{\mathsf{sum}} + B_{\mathsf{upd}} < 2 \cdot \frac{q}{4} + \frac{q}{4} = \frac{3q}{4} < q.$$

Therefore two distinct admissible integer expressions cannot become equal merely after reduction modulo q.

Every public threshold, and every committed liability used in a comparison, is required to lie in

$$0 \le t, L \le B_{\mathsf{cmp}} < \frac{q}{2}.$$

For the positive branch, the prover proves

$$sum_i - t = s_+, \qquad 0 \le s_+ < 2^l \le \frac{q}{2}.$$

If instead $sum_i < t$, the left-hand side is a negative integer. Its field representation lies between $(\frac{q}{2}, q]$ and therefore cannot fall into the allowed range of s_+.

Similarly, for the negative branch the prover proves

$$t - 1 - sum_i = s_-, \qquad 0 \le s_- < 2^l \le \frac{q}{2}.$$

If $sum_i \ge t$, the left-hand side is negative and its field representation again lies outside the allowed range.

More explicitly, our parameter ranges imply

$$B_{\mathsf{sum}} + B_{\mathsf{cmp}} + 1 < \frac{q}{4} + \frac{q}{2} + 1 < q,$$

so neither comparison branch can be satisfied by modular wraparound. The same argument applies when t is replaced by a committed liability L.