Private Accounts Without $O(N)$ Work Through a Hybrid State Model

Philipp Gackstatter[1], François Garillot[1], Marti Górny[1], Adrian Hamelink[1], Al Kindi[1], Philippe Laferrière[2], Robin Salen[1], Brian Seong[1], and Bobbin Threadbare[1]

[1] Miden
{philipp.gackstatter, francois, marti, adrian.h, al.kindi, robin.s, brian.s, bobbinth}@miden.team
[2] Anza
laferriere.phil@gmail.com
Work done while at Miden.

Abstract. Privacy-preserving smart contracts face a scalability problem. In account-balance-vector designs, hiding which accounts changed requires that every transaction perform an indistinguishable update over the global state, so validators pay work linear in the number of accounts for every private transfer. UTXO systems avoid this by validating local spends, but do not provide the same account-style programming model. We argue that the usual "account vs. UTXO" framing conflates state representation and execution model, two choices that can be made separately. Miden, an Ethereum Layer 2, separates them. It uses persistent accounts for programmability, asynchronous notes for value transfer, and client-side proving for execution. Validators verify proofs and update commitments without re-executing transactions or accessing uninvolved state.

The result is an account-based platform with private transfers and private computation, while keeping validator work per transaction logarithmic in the number of accounts.

1 Introduction

1.1 The Privacy-Scalability Barrier

Consider a blockchain with $N =$ one million accounts. Alice wants to send Bob 5 ETH privately—observers should not learn that Alice and Bob transacted. How can this work?

In account-based systems (Ethereum-style), state is a list of account balances, and a private transfer must update two hidden balances. If validators execute this transaction, they either process only Alice and Bob's accounts, which reveals who transacted, or touch every one of the million accounts so that the two genuine updates hide among the rest. Any of the $\binom{N}{2} \approx 5 \times 10^{11}$ account pairs then looks equally likely to be the transacting pair. The first option gives up privacy. The

second costs validators work linear in N, because each private transfer rewrites all N account records.

UTXO-based systems avoid this cost because they check local spends in parallel, but plain UTXOs do not provide persistent state objects with stable programmable identities. Stateful contracts can be encoded by threading state through successive outputs, but the contract is not exposed through a stable account-style interface.

The barrier arises not from accounts per se but from requiring anonymity-preserving *global* updates to a shared account-balance vector. Shifting execution from validators to clients changes where this cost is paid, but not whether the update spans all accounts.

1.2 The False Dichotomy

Privacy-preserving systems that rely on client-generated proofs have often converged on UTXO and record representations, for a concrete reason. A proof of a transition over a persistent account must be regenerated each time that account is used, and repeated proofs on one account are linkable unless each one is hidden among indistinguishable updates to all N accounts. Moving to ephemeral objects that are spent once and replaced removes the persistent identifier, and with it the linkability problem.

Model	**Persistence**	**Sequencing Pressure**	**Privacy Impact**
UTXO	Outputs are consumed & replaced.	No shared inputs; independent spends parallelize naturally.	Spent objects carry no persistent identifier.
Accounts	Identities & storage persist & mutate.	Many valid "next" transitions per account → ordering.	Persistent identifiers, linkable unless transfers are mediated.

Table 1. Persistence as the key representational difference between UTXO and account models.

That standard response is sound (the trade-offs of Table 1 are real), but it settles two design questions at once. The first is the *execution model*, the choice of who executes a transaction and who verifies it. The second is the *state representation*, the choice of what persists, mutable accounts or one-time outputs. Nothing about client-side proving forces the second choice. Miden takes a combination the field has largely passed over: persistent and programmable accounts with client-side proving. It routes cross-account effects through notes, so validators verify proofs of single-account transitions and never access state outside the proven account.

1.3 Why Accounts Matter for Smart Contracts

The account model gives developers a first-class abstraction for persistent, stateful, policy-bearing objects. A contract is a long-lived object with address, state, code, interface, and policy, rather than a spend predicate on an output. Work on structured EUTxO contracts [22] notes that account-based platforms such as Ethereum have *"inherently stateful"* smart contracts, whereas EUTxO contracts are predicates on transaction data that authorize a specified ledger update. That is not a defect of EUTxO, because predictability and static reasoning are what the design aims for. It does mean that stateful applications must be encoded through outputs, datums, and conventions, rather than exposed as a first-class contract object.

Broader comparisons point the same way: analyses of the two programming paradigms find the account model better suited to general-purpose applications [19], and account-based contract languages express many familiar application patterns more directly and at shorter length than their UTXO counterparts [10].

Accounts provide encapsulation and stable interfaces. A contract account can own state and assets, expose procedures, enforce invariants, and be referred to by a stable identifier. Rich mutable state, e.g., dynamic maps, counters, order books, credential registries, and policy objects, is precisely what smart-contract developers want to mutate over time. UTXO-based systems can distribute such state, but the developer then programs a state-distribution protocol in addition to the application [22].

Miden shows that this abstraction need not imply validator-executed global state or $O(N)$ privacy updates. Accounts can instead be persistent, private, and programmable, with notes as the UTXO-like one-time objects for private transfer.

1.4 Open Problem #3

A recent SoK on Privacy-Preserving Transactions in Blockchains [9] formalizes the account-based privacy challenge as Open Problem #3: "Can we achieve full anonymity and confidentiality efficiently without transaction size being $o(N)$ and validator work being $o(N)$ in the account-based model?"

Global updates also create a concurrency penalty: if every transaction refreshes the entire balance vector, then every proof is tied to a specific global root, and semantically independent actions are serialized by the shared update—a head-of-line blocking that the SoK discusses as "front-running" in Zether-style protocols [9]. The penalty is most severe in proof-driven blockchains, where regeneration makes the lost race expensive.

Miden addresses the cost of private account transitions through per-account proofs and note-mediated transfers. The protocol hides private state and the correspondence between note creation and consumption, while account identifiers and note metadata remain public (Section 3.5).

1.5 Architecture for Programmable Private Accounts

Miden is a ZK-rollup on Ethereum [30]: an operator executes Miden's chain off-chain and posts block state roots and aggregated block proofs to an Ethereum contract, so Ethereum's consensus gives Miden blocks their finality. It implements the hybrid model of this paper through four components, described in Sections 2 to 8: accounts as independent state machines with client-side proving, notes as asynchronous messages for value transfer, a three-database accumulator state model, and hierarchical proof aggregation. The project is open-source and runs on a public testnet.

The privacy analysis assumes zero-knowledge transaction and aggregation proofs, obtained using established STARK hiding techniques [32].

2 The Actor Model and State Architecture

2.1 Actors as Accounts

Miden adopts the actor model [33,1,5]: each account is an isolated state machine with exclusive custody of its own storage, vault, and code. The actor model, proven in systems such as Erlang, Elixir, and Akka [36], avoids shared memory and implicit dependencies, so disjoint state can proceed without coordination.

Applied to blockchain state, this yields three benefits: (*i*) privacy through independent, proof-carried transitions; (*ii*) parallelism across disjoint account activity; (*iii*) elimination of re-entrancy vulnerabilities by construction. Validators do not execute private account logic—they verify proofs of individual account transitions.

An account comprises an ID (120-bit identifier encoding type, storage mode, and version), code (Miden Assembly procedures), storage (up to 255 named slots with 32-byte values or StorageMaps), a vault (sparse Merkle tree of assets), and a nonce for replay protection. Accounts are Turing-complete smart contracts supporting arbitrary logic.

Miden supports private and public accounts, which differ in state visibility and in how much ordering they need (Table 2). A private account updates single-owner state through client-side proving. Each proof certifies one owner's transition, so two private transactions on different accounts never conflict, and no global order between them is needed. A public account holds state that many parties can contend for, e.g., an auction or an exchange pool, and its transactions must be placed in one canonical order. Today the sequencer supplies that order. In a decentralized deployment, producing it requires consensus (Section 2.6).

A private account can alternatively serve as a *burner* account—created, used once, abandoned—so that its notes carry a fresh creator ID (Table 2; Section 3.6).

[3] Commitment–nullifier unlinkability holds under the assumptions of Section 3.3; public metadata and discovery may still identify counterparties. Burner accounts require migrating assets and code into each fresh account.

Property	Private Accounts	Private, burner mode	Public Accounts
State Visibility	Commitment, owner-only	One-time, then discarded	Publicly readable on-chain
Throughput	Scales with user proving	Scales with user proving	Bounded by shared-state ordering
Activity linkage[3]	Activity linked by account ID	Fresh account ID for each use	Public state and account activity
Use Cases	Private payments, credentials	One-time payments	DEXs, auctions, shared counters
Total order	Not required	Not required	Required (*slow lane*)

Table 2. Private vs. public accounts in Miden; all three modes use client-side proving for execution. The fast and slow lanes are defined in Section 2.6.

2.2 Asynchronous Message Passing via Notes

Accounts communicate via notes: messages containing assets and scripts defining consumption conditions. Creation and consumption are separate transactions (Figure 1): Alice creates a note and Bob consumes it in independent operations on independent accounts, so the protocol never needs atomic multi-account updates.

Notes use randomized commitments, nullifiers, and zero-knowledge proofs for private transfers [9,14], with optional batching (CoinJoin [41]). Unlike UTXO systems where notes represent unspent outputs, Miden's notes are messages between accounts. Accounts hold vaults; notes carry vault update instructions. This removes synchronous call stacks (eliminating re-entrancy), keeps dependencies explicit for parallel execution, and supports privacy without global anonymity-set updates.

The trade-off is asynchronous cross-account interaction: atomic swaps require escrow note patterns, and multi-step operations span multiple blocks.[4] Developers think in state machines and message passing rather than synchronous function calls.[5]

For trading applications, this message-passing model avoids the hub-and-spoke contention of classical CFMM order books [4] and aligns with batched intent-based trading approaches that decouple sequencing from execution [20,26].

2.3 Single-Account Transactions

A transaction in Miden updates exactly one account: given input (account state and notes to consume), the account executes its code and produces output (new

[4] If Miden adopted a FastPay-style design [11], Alice could take a sequencer certificate to her counterpart immediately.

[5] Network transactions for public shared state are also supported, enabling validator-executed operations on several public accounts simultaneously.

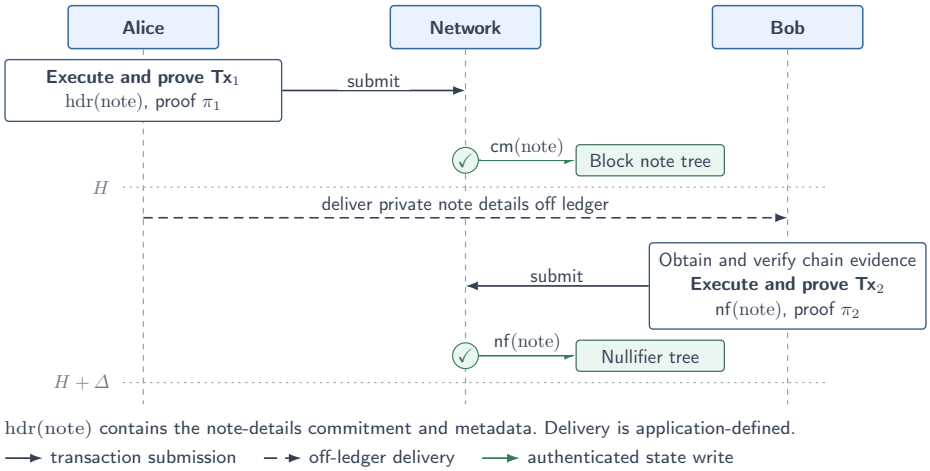


Fig. 1. Asynchronous value transfer via notes. Table 4 lists the ledger-visible information at creation and consumption; the two databases are defined in Section 2.4.

account state and created notes). By construction, each private proof updates exactly one owner-controlled account state, scoping proof-carrying privacy to single-owner state while genuinely shared state is coordinated through public sequencing.

This granularity is critical for proof-driven systems. In shared mutable state, two valid proofs can be built from the same pre-state but only one can be sequenced. In the normal course of using a Miden account, conflicting private proofs do not arise, since producing two requires the same owner to act twice on one account state. Failure is sender-local rather than system-wide, though recovery is still costly: regenerating a proof is substantially more expensive than re-creating a plain transaction message. Some proof-driven blockchains use application-level commutative updates [44], which are difficult to validate and compose [49,48].

The same asynchrony applies to transfers: an Alice→Bob payment is two transactions rather than one atomic multi-account update.

2.4 Three-Database State Model

Given single-account execution and asynchronous notes, global state decomposes into three accumulators (Table 3):

2.5 Consequences of Single-Account Execution

Single-account transactions imply that validators see only commitments on-chain. This creates two privacy questions: preventing double-spending without

[6] Each leaf of the Chain MMR is a block header holding the root of that block's note tree.

	Account DB	**Note DB**	**Nullifier DB**
Structure	Sparse Merkle tree	Chain MMR[6]	Sparse Merkle tree
Stores	Account commitments	Note commitments	Consumed note markers
Query	Random access by ID	Append, membership proof	Existence check, insert
Cost	$O(\log N)$	$O(\log M)$	$O(\log K)$

Table 3. Miden's three-database state model.

revealing which note was consumed (nullifiers, Section 3), and enabling recipients to discover relevant notes while managing read-path privacy and bandwidth (Section 3.4).

2.6 Sequencing, Consensus, and Current Deployment

Three roles must be kept apart. *Execution* happens on the client: the author of a transaction runs its account code locally and attaches a proof, and no operator ever executes private account logic. *Block inclusion* belongs to an operator, who orders transactions into batches and blocks, verifies their proofs, and applies the new commitments to the three state databases. *Finality* comes from Ethereum, as in any ZK-rollup [30]: block state roots and aggregated proofs posted to an Ethereum contract become canonical under Ethereum's consensus.

On the current testnet, a single sequencer performs block inclusion. Users trust it for availability and censorship resistance, but not for correct execution, which the proofs enforce.

Every Miden transaction declares its complete footprint, one account plus the notes it consumes. Two transactions on disjoint footprints cannot conflict, so any order between them gives the same result. Ordering every transaction with respect to every other transaction is the analogue of pessimistic concurrency control in database systems, and this state model makes it unnecessary for disjoint pairs.

With a single sequencer, avoiding that order saves little, because one operator can dictate a global order cheaply. The saving matters in a decentralized deployment. Transactions on singly owned state can be certified by Byzantine consistent broadcast without fixing a total order; only transactions on shared state need consensus. We call the first path the *fast lane* and the second the *slow lane*. That divide is established in theory [31] and in deployment, from FastPay to Sui's owned and shared objects and Linera's per-owner microchains [11,15,37].

Certification without consensus assumes owners do not equivocate by publishing two conflicting transitions of one account. Certification without consensus still requires verification: Byzantine consistent broadcast detects the two conflict-

ing transitions before either is executed. The equivocation is provable from the pair of proofs, and its damage stays inside the owner's account.[7]

3 Notes and Nullifiers

Accounts retain state across transfers, while notes are one-time objects. Miden inherits Zerocash's commitment–nullifier privacy mechanism [14]: a consumption transaction publishes a nullifier instead of the consumed commitment. The Miden VM adds programmable account and note logic. The public trace still records account activity and note metadata, and discovery may disclose further information about counterparties (Sections 3.5–3.6).

3.1 Notes and Commitments

A note is an immutable message carrying assets, a script and its inputs, a serial number, and public metadata. The script and inputs define the conditions under which an account may consume the note and how consumption changes the account's state. A random serial number, fixed at creation, distinguishes otherwise identical notes and provides the randomness needed to hide predictable contents. The public metadata includes the creating account's ID and a discovery tag that wallets use to filter candidate notes.

An account creates a note as a transaction output. Once the transaction has been validated and included in a block, the Note DB records a commitment to each output note not consumed within that block, using the hash function H_{cm}:

$$\mathsf{commitment} = H_{\mathrm{cm}}(\mathsf{note}), \tag{1}$$

where note denotes the complete note contents, including its public metadata.

Public notes publish their full contents. For a private note, the commitment and metadata are public, while the consuming account must obtain the remaining contents through a delivery mechanism described in Section 3.4.

3.2 Note Consumption and Nullifiers

When an account consumes a note by executing its script within a transaction, publishing the note's commitment would identify its creation record. The transaction instead publishes a deterministic nullifier computed using a hash function H_{nf} domain-separated from H_{cm}:

$$\mathsf{nullifier} = H_{\mathrm{nf}}(\mathsf{note}). \tag{2}$$

If the nullifier depended only on the serial number, an attacker could copy a public note's serial into a different note and consume that note first, blocking

[7] The state-model and validator-work arguments do not depend on a particular consensus protocol. The two lanes describe a decentralized operator set, not a performance claim about today (Section 8).

consumption of the original. Binding the nullifier to the note's contents prevents this attack.

For notes authenticated within the consuming transaction, the proof establishes knowledge of a note such that:

- its commitment $H_{\mathrm{cm}}(\mathsf{note})$ belongs to the authenticated Note DB;
- its script executes successfully in the context of the consuming account; and
- $H_{\mathrm{nf}}(\mathsf{note})$ equals the published nullifier.

All three conditions refer to the same note. Block producers verify the proof, reject nullifiers already in the Nullifier DB, and insert accepted nullifiers. Because the commitment fixes the note's contents and those contents determine its nullifier, any second consumption produces a previously accepted nullifier and is rejected.

3.3 Commitment–Nullifier Unlinkability

An observer could try to match a nullifier to a commitment by enumerating otherwise predictable note contents. Under the random-oracle idealization of the domain-separated hash functions, the secret random serial prevents that test. The analysis also assumes zero-knowledge transaction proofs that establish valid consumption without revealing private note data or the membership witness. Subsequent aggregation must disclose no additional private note data.

This protection applies to observers who lack the private note contents. Anyone who knows the complete note can compute its nullifier and recognize consumption, including a creator who possesses those contents. Public notes therefore do not provide consumption unlinkability.

3.4 Note Discovery

Bob needs a private note's contents before consuming it. Alice can deliver them over an encrypted side channel or distribute encrypted note data through a note-transport service or on-chain attachments [42,43]. For consumption authenticated within his transaction, Bob also needs a witness that the commitment belongs to the Note DB. This witness becomes available after the creating transaction is included in a block.

Recipients can scan an available encrypted note feed locally, as in Zcash's light-client protocol, where wallets download compact output data and trial-decrypt it [28]. This avoids recipient-specific discovery queries, but each wallet's bandwidth and processing grow with system-wide note traffic.

A wallet can instead query a node or note-transport service by tag and check matching entries locally. Chain nodes return membership witnesses for all matching commitments, so no separate witness request is needed. Section 6 describes how these witnesses are updated. Every Miden note carries a 32-bit public tag chosen by its creator under a convention agreed with the recipient. A common convention uses a short prefix of the recipient's account ID. Shorter prefixes reveal fewer account-ID bits but return more unrelated notes. Other conventions

derive tags from shared secrets, as in Aztec [7]. A tag helps a wallet find a note but does not authorize its consumption.

An account-prefix tag reveals part of the intended recipient's ID to every observer. Even when many notes match, a shared tag obscures the recipient only while several accounts remain plausible. When few recipients are active, even a short prefix may leave only one plausible recipient.

Tag queries expose additional information to the service that answers them: the requested tag, query time, and connection information such as the client's network address. Repeated queries can link activity over time. Direct delivery and local scanning avoid tag queries, but cannot remove information already published in the tag.

Efficient discovery with strong receiver privacy remains a challenge for private blockchains. Possible extensions to Miden include fuzzy message detection, which outsources detection with false positives [13], and oblivious message retrieval, which hides from the serving system which messages are relevant [38]. Recent work reduces the cost of oblivious retrieval [39,24], but these techniques still shift substantial computation or communication to the serving system. They are not yet a scalable, drop-in replacement for local scanning. Scanning could also be delegated to a trusted execution environment, as in Aleo's Shield Wallet [3], with additional hardware trust assumptions. These choices affect client synchronization and receiver privacy while leaving the state model and transaction validation unchanged.

3.5 Public Information and Transfer Privacy

Table 4 follows a private transfer from Alice's note creation through Bob's tag synchronization and consumption with a proof of membership in the Note DB. Synchronization adds no ledger entry, but reveals query information to the serving node (Section 3.4).

Step	**New ledger data**	**What observers learn**
Creation, block H	Creator account ID A, Alice's updated account commitment, note commitment and public metadata (including tag t)	Account A created a note carrying tag t
Witness retrieval, after block H	None	Serving node sees requested tags, query timing, and connection information
Consumption, block $H+\Delta$	Consuming account ID B, Bob's updated account commitment, and nullifier	Account B consumed a note

Table 4. Ledger and query observations during a private transfer.

The ledger identifies activity by both accounts. Under the assumptions of Section 3.3, an observer without the private note contents cannot use the commitment, nullifier, or proof to identify which note Bob consumed. For that observer, the note contents remain confidential and consumption remains cryptographically unlinkable [9,45].

Even with the private note contents hidden, an observer may infer a relationship between Alice and Bob if tag t leaves Bob as the only plausible recipient of a note Alice creates. If Bob then consumes a note shortly afterward, the timing provides further evidence. Query disclosures and other auxiliary information can also help identify counterparties, so sender–receiver unlinkability depends on the surrounding information as well as the commitment–nullifier construction. Section 3.6 discusses mitigations for linkage introduced by public creator IDs.

3.6 Creator Linkability and Mitigation

Public sender account IDs let observers group notes created by the same account. They can see that account A created a note at time T, though the ID alone does not identify the recipient. Optional discovery conventions may disclose recipient information (Section 3.4). The transferred assets and amounts, the note's private script and inputs, and the accounts' private state remain hidden. Linkage through public sender metadata can be mitigated within the existing note model without global rerandomization or protocol changes.[8]

Relay/re-emission accounts. Alice creates a note to a router, which waits, batches, and re-emits a fresh note to Bob. The router becomes the final sender.[9] Because notes are already asynchronous messages, this pattern is cheaper than Ethereum-style mixers [9].

Account hygiene. A sender can use separate note-origin accounts for separate contexts, so public metadata names a context account rather than the user's main account.

Private sender authentication. Authorization can reside in the note's private data or script instead of public metadata.

4 Client-Side Proving and Avoiding O(N) Work

Miden avoids the linear balance-vector updates of Anonymous Zether [21,23] and PriFHEte [40]. Validator work per transaction is dominated by verifying a client-generated proof and updating a small number of accumulator entries.

Client-side execution and the zkVM. The Miden VM [35] is a stack-based zkVM that executes programs and produces STARK proofs. Unlike circuit-based approaches, a zkVM uses one verifier for any program given as bytecode. The VM uses Merklized Abstract Syntax Trees (MAST) for program representation:

[8] Timing, amount, and tag structure can re-link flows without batching and padding [9].

[9] A single relay sees Alice unless multiple relayers or threshold routing is used [9].

provers disclose only executed control-flow paths, keeping private account logic hidden.

Replacing validator execution with proof verification. Miden distributes proving to users: each user executes their transaction locally, generates a STARK proof, and submits it with the new account commitment, created note commitments, and nullifiers. The validator verifies the proof ($O(1)$), updates the Account DB (one leaf), appends to the Note DB, and inserts nullifiers into the Nullifier DB, with no re-execution and no access to account internals.

Validator work per transaction is logarithmic (with N accounts, M notes, K nullifiers):

- Account DB: $O(\log N)$.
- Note DB: $O(\log M)$.
- Nullifier DB: $O(\log K)$.

Throughput scales with users proving. On a MacBook Pro M4 Max, client-side proving takes ∼460 ms per P2ID transaction [35]. Miden uses STARKs (transparent, post-quantum) rather than SNARKs [29], with recursive aggregation (Section 5) mitigating proof size.

5 Proof Aggregation and State Management

Batch and block production. Batch production verifies client-generated transaction proofs recursively inside the Miden VM,

maintaining ordering and checking note uniqueness. It produces batch proofs, which block production aggregates into block proofs. Block production also verifies the integrity of both databases and double-spend prevention.

Managing nullifier state growth. Nullifiers accumulate monotonically: every consumed note adds one nullifier, forever. At 1000 TPS, the Nullifier DB grows by ∼1 TB/year. The SoK identifies this growth as a deployment challenge because storage costs grow linearly with each posted transaction [9].

Miden addresses this growth with evolving nullifiers [18], partitioning nullifiers into epoch-based sparse Merkle trees $[s_1, \ldots, s_n]$, where s_i stores nullifiers from epoch i. A new epoch begins every L blocks. At the transition to epoch $n + 1$, the current tree s_n is frozen to prevent further updates, and a new tree s_{n+1} is created for the next epoch.

Each note records its creation block number r at commitment time. Consuming a note with creation epoch $c = \lfloor r/L \rfloor$ requires proving the nullifier is absent from all trees s_c through s_n, via membership proofs for the frozen epoch roots, non-membership in each frozen tree, and non-membership in s_n. After verification, the block producer adds the nullifier to s_n.

After an epoch ends and enough time passes for double-spend detection, validators prune its nullifier tree. The cost is that proof complexity grows with note age: consumption within the creation epoch needs one non-membership proof, and older notes need one per frozen epoch since creation. Longer epochs reduce proof complexity but delay pruning; shorter epochs do the reverse.

6 The Merkle Mountain Range and Stale-Proof Resilience

Merkle Mountain Ranges let users extend membership witnesses as the Note DB grows without re-querying, addressing the SoK's call for sublinear state size (Open Problem #1): witnesses extend locally, and the cost scales with blocks since creation rather than total transaction count.

The stale witness problem. In Merkle trees, membership proofs become invalid when leaves change, so users must re-query for fresh proofs, which leaks interest and adds synchronization overhead.

Why an MMR solves this. The Note DB uses a Merkle Mountain Range, an append-only accumulator [47,16]. An MMR with n leaves has at most $\lfloor \log_2 n \rfloor + 1$ peaks (exactly popcount(n)); on append, peaks merge like binary addition. Existing proofs never go stale: a proof valid at root R_1 can be extended to root R_2.

Extension protocol. The client holds a witness W at height H, consisting of an authentication path P and the peaks. At height $H' > H$, the client requests the peaks at both heights and computes $W' = (P, \text{peaks}_{H'})$. The extended witness verifies against root$(\text{peaks}_{H'})$. This requires transferring and computing $O(\log(H' - H))$ hashes.

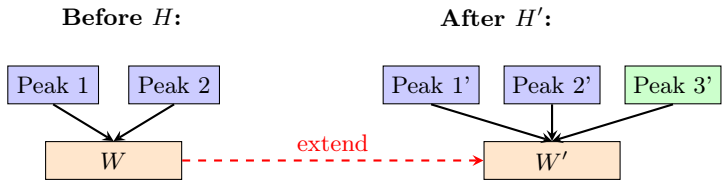


Fig. 2. MMR witness extension.

Enabling local transactions with privacy. Users obtain a witness once, extend it locally to the current root without further note-specific queries, and submit a proof that hides which note was consumed.

7 Related Work

The standard taxonomy [9] branches on "Account vs UTXO." Using the lens from Section 1.2, we instead position every system by its representation and execution model (Table 5).

System	Representation	Execution model
Ethereum	Accounts	Validator-executed shared-state
Anonymous Zether [21,23]	Accounts	Validator-executed updates across a chosen anonymity set
Bitcoin	UTXO	Signature validation over local spends
Zcash [14]	UTXO	ZK proofs with nullifiers/commitments
EUTXO [22]	Extended UTXO	Validator-executed per-output scripts
Aleo/Aztec [17]	Private records	Client-generated ZK proofs
Starknet pool [27]	Shared contract	Client STARKs over shared pool
Miden	Accounts + notes	Per-account proof-carrying execution

Table 5. Positioning Miden among related systems.

The ZEXE lineage and record-based private computation. The lineage starts with Zcash, where private payments run on Zerocash's commitments and nullifiers [14,34]. On that axis the systems coincide by construction. Miden combines the commitment–nullifier architecture with persistent programmable accounts, while retaining one-time notes (Section 3). Zcash shielded outputs carry no persistent state or programmable identity. Zcash's light-client protocol downloads compact output data for trial decryption [28]. Miden can likewise use local scanning of encrypted note data, with optional tag-based filtering to reduce discovery work (Section 3.4). Validator work per transaction is comparable in both—proof verification plus logarithmic accumulator updates. Zcash thus achieves efficient private payments simply by not having accounts. The SoK's question of private *accounts* without $O(N)$ work stays open.

ZEXE [17] models state as UTXO-style records consumed and replaced in zero-knowledge proofs, with the chain maintaining a commitment to the record set and the spent serial numbers. Because proofs are tied to specific input records, pooled state creates head-of-line bottlenecks. Aleo and Aztec adopt this record model for private state; persistent updatable state is public [6,2].

Miden instead keeps long-lived private state in persistent accounts and uses notes only for cross-account transfer, shifting private mutable state out of shared pools into per-account accumulators.

Beyond payments: stateful private applications. Privacy-preserving ledgers also target stateful credentials, e.g., zk-Promises' anonymous moderation [46] and zk-Cookies' continuous authentication [25]. Miden supports such applications with private account state, callbacks via notes, and nullifiers preventing rollback.

Relative to Anonymous Zether, Miden avoids updates across an anonymity set; relative to EUTXO, it places long-lived mutable state in accounts.

8 Conclusion and Future Work

Summary of contributions. Miden addresses three open problems from [9]. Per-account proofs and note-mediated transfer support private accounts without

$O(N)$ work (Open Problem 3). For the witness-refresh problem (Open Problem 1), MMR witness extension requires $O(\log(H' - H))$ local work (Section 6). Epoch-based nullifiers address account-based state growth (Open Problem 6; Section 5). The resulting architecture supports programmable private accounts with logarithmic validator work per transaction.

Future work. Block production is centralized today (Section 2.6); decentralizing it along the fast and slow lanes is the main open task [11,12,15,8,37]. Further work includes epoch-based nullifier trees (Section 5), creator-linkability mitigations (Section 3.6), and continued investigation of efficient note discovery with stronger receiver privacy.

Future systematizations should classify systems by state representation and execution model, two independent choices, rather than use the dichotomy between accounts and one-time outputs as the primary split.

References

1. Agha, G.: Actors: A Model of Concurrent Computation in Distributed Systems. The MIT Press (Dec 1986). https://doi.org/10.7551/mitpress/1086.001.0001
2. Aleo: Managing state: Using records. https://developer.aleo.org/sdk/guides/managing_state/#using-records (Accessed: 2026-02-19)
3. Aleo: Shield wallet changelog: Encrypted record scanning in v1.14.0. https://aleo.org/shield/changelog (Accessed: 2026-09-23)
4. Angeris, G., Chitra, T.: Improved price oracles: Constant function market makers. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. p. 80–91. AFT '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3419614.3423251
5. Armstrong, J.: Making Reliable Distributed Systems in the Presence of Software Errors. Ph.D. thesis, Royal Institute of Technology, Stockholm (2003), https://erlang.org/download/armstrong_thesis_2003.pdf
6. Aztec: State management. https://docs.aztec.network/developers/docs/foundational-topics/state_management (Accessed: 2026-02-19)
7. Aztec: Note discovery. https://docs.aztec.network/developers/docs/foundational-topics/advanced/storage/note_discovery (Accessed: 2026-03-13)
8. Babel, K., Chursin, A., Danezis, G., Kichidis, A., Kokoris-Kogias, L., Koshy, A., Sonnino, A., Tian, M.: Mysticeti: Reaching the Limits of Latency with Uncertified DAGs (2023). https://doi.org/10.48550/arXiv.2310.14821, https://doi.org/10.48550/arXiv.2310.14821
9. Baldimtsi, F., Chalkias, K.K., Madathil, V., Roy, A.: SoK: Privacy-Preserving Transactions in Blockchains (2024), https://ia.cr/2024/1959
10. Bartoletti, M., Benetollo, L., Bugliesi, M., Crafa, S., Dal Sasso, G., Pettinau, R., Pinna, A., Piras, M., Rossi, S., Salis, S., Span
'o, A., Tkachenko, V., Tonelli, R., Zunino, R.: Smart contract languages: A comparative analysis. Future Generation Computer Systems p. 107563 (2025). https://doi.org/10.1016/j.future.2024.107563
11. Baudet, M., Danezis, G., Sonnino, A.: FastPay: High-Performance Byzantine Fault Tolerant Settlement. arXiv:2003.11506 [cs] (Nov 2020). https://doi.org/10.48550/arXiv.2003.11506

12. Baudet, M., Sonnino, A., Kelkar, M., Danezis, G.: Zef: Low-latency, Scalable, Private Payments. arXiv:2201.05671 [cs] (Jan 2022). https://doi.org/10.48550/arXiv.2201.05671
13. Beck, G., Len, J., Miers, I., Green, M.: Fuzzy Message Detection. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 1507–1528. CCS '21, Association for Computing Machinery, New York, NY, USA (Nov 2021). https://doi.org/10.1145/3460120.3484545
14. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized Anonymous Payments from Bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474 (2014). https://doi.org/10.1109/SP.2014.36
15. Blackshear, S., Chursin, A., Danezis, G., Kichidis, A., Kokoris-Kogias, L., Li, X., Logan, M., Menon, A., Nowacki, T., Sonnino, A., Williams, B., Zhang, L.: Sui Lutris: A Blockchain Combining Broadcast and Consensus (2023). https://doi.org/10.48550/arXiv.2310.18042, https://doi.org/10.48550/arXiv.2310.18042
16. Bonneau, J., Chen, J., Christ, M., Karantaidou, I.: Merkle Mountain Ranges are Optimal: On Witness Update Frequency for Cryptographic Accumulators. In: Advances in Cryptology – CRYPTO 2025 (2025). https://doi.org/10.1007/978-3-032-01878-6_6
17. Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., Wu, H.: Zexe: Enabling decentralized private computation. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 947–964 (2020). https://doi.org/10.1109/SP40000.2020.00050
18. Bowe, S., Miers, I.: A Note on Notes: Towards Scalable Anonymous Payments via Evolving Nullifiers and Oblivious Synchronization (2025), https://ia.cr/2025/2031
19. Brünjes, L., Gabbay, M.J.: UTxO- vs Account-Based Smart Contract Blockchain Programming Paradigms. In: Leveraging Applications of Formal Methods, Verification and Validation: Applications (ISoLA 2020). pp. 73–88 (2020). https://doi.org/10.1007/978-3-030-61467-6_6
20. Budish, E., Cramton, P., Shim, J.: The High-Frequency Trading Arms Race: Frequent Batch Auctions as a Market Design Response *. Q J Econ **130**(4), 1547–1621 (Nov 2015). https://doi.org/10.1093/qje/qjv027
21. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards Privacy in a Smart Contract World. In: Financial Cryptography and Data Security – FC 2020. pp. 423–443 (2020). https://doi.org/10.1007/978-3-030-51280-4_23
22. Chakravarty, M.M.T., Chapman, J., MacKenzie, K., Melkonian, O., Peyton Jones, M., Wadler, P.: The Extended UTXO Model. In: Financial Cryptography and Data Security: FC 2020 International Workshops, AsiaUSEC, CoDeFi, VOTING, and WTSC, Kota Kinabalu, Malaysia, February 14, 2020, Revised Selected Papers. pp. 525–539. Springer-Verlag, Berlin, Heidelberg (Feb 2020). https://doi.org/10.1007/978-3-030-54455-3_37
23. Diamond, B.E.: Many-out-of-many proofs and applications to anonymous zether. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 1800–1817 (2021). https://doi.org/10.1109/SP40001.2021.00026
24. Fisch, B., Liu, Z., Tromer, E., Wang, Y.: UnifOMR: Oblivious message retrieval with near-optimal concrete efficiency. Cryptology ePrint Archive, Paper 2026/910 (2026), https://eprint.iacr.org/2026/910
25. Frolov, A., Triedman, H., Miers, I.: Zk-Cookies: Continuous Anonymous Authentication for the Web (2025), https://ia.cr/2025/1938

26. Garreta, A., Gągol, A., Stouka, A.P., Straszak, D., Zajac, M.: COMMON: Order book with privacy. Cryptology ePrint Archive, Paper 2023/1868 (2023), https://ia.cr/2023/1868
27. Goldberg, L., Dotan, M., Dror, I., Kaempfer, G., Levi, N., Oved, N., Reder, A., Veredgorn, A., Wolfgor, N.: Scalable compliant privacy on starknet. Cryptology ePrint Archive, Paper 2026/474 (2026), https://ia.cr/2026/474
28. Grigg, J., Hopwood, D.E.: ZIP 307: Light client protocol for payment detection. https://zips.z.cash/zip-0307 (Accessed: 2026-09-23)
29. Groth, J.: On the Size of Pairing-Based Non-interactive Arguments. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016. pp. 305–326. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_11
30. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: SoK: Layer-Two Blockchain Protocols. In: Financial Cryptography and Data Security (FC 2020). pp. 201–226 (2020). https://doi.org/10.1007/978-3-030-51280-4_12
31. Guerraoui, R., Kuznetsov, P., Monti, M., Pavlovič, M., Seredinschi, D.A.: The consensus number of a cryptocurrency. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC). pp. 307–316 (2019). https://doi.org/10.1145/3293611.3331589
32. Haböck, U., Kindi, A.: A note on adding zero-knowledge to STARKs. Cryptology ePrint Archive, Paper 2024/1037 (2024), https://ia.cr/2024/1037
33. Hewitt, C., Bishop, P., Steiger, R.: A universal modular ACTOR formalism for artificial intelligence. In: Proceedings of the 3rd International Joint Conference on Artificial Intelligence. pp. 235–245. IJCAI'73, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (Aug 1973)
34. Hopwood, D., Bowe, S., Hornby, T., Wilcox, N.: Zcash Protocol Specification, Version 2022.3.8 [NU5] p. 221 (2022), https://zips.z.cash/protocol/protocol.pdf, see Section 4.1.8 for Orchard nullifier design
35. Labs, P., contributors, M.: Miden architecture overview. https://docs.miden.xyz/miden-vm/imported/miden-base/src/index (Accessed: 2026-02-19)
36. Lightbend Inc.: Akka documentation: Release 2.5.23 (2019), https://doc.akka.io/docs/akka/2.5.23/
37. Linera team: Linera: A Blockchain Infrastructure for Highly Scalable Web3 Applications. Tech. rep., Linera (2023), https://linera.io/whitepaper
38. Liu, Z., Tromer, E.: Oblivious Message Retrieval (2021)
39. Liu, Z., Tromer, E., Wang, Y.: PerfOMR: Oblivious message retrieval with reduced communication and computation. Cryptology ePrint Archive, Paper 2024/204 (2024), https://eprint.iacr.org/2024/204
40. Madathil, V., Scafuro, A.: PriFHEte: Achieving Full-Privacy in Account-based Cryptocurrencies is Possible (2023), https://ia.cr/2023/710
41. Maxwell, G.: Coinjoin: Bitcoin privacy for the real world. Bitcoin Forum (August 2013), https://bitcointalk.org/index.php?topic=279249.0
42. Miden contributors: Miden addresses. https://github.com/0xMiden/protocol/blob/701b6e10acdd17d95a53e2b63b22056c39e4c253/docs/src/account/address.md (Accessed: 2026-09-23)
43. Miden contributors: Miden notes. https://github.com/0xMiden/protocol/blob/701b6e10acdd17d95a53e2b63b22056c39e4c253/docs/src/note.md (Accessed: 2026-09-23)
44. o1Labs: Exploring mina's unique architecture. https://www.o1labs.org/blog/exploring-minas-unique-architecture (Accessed: 2026-03-13)

45. Pfitzmann, A., Hansen, M.: A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. http://dud.inf.tu-dresden.de/literature/Anon_Terminology_v0.34.pdf (2010)
46. Shih, M., Rosenberg, M., Kailad, H., Miers, I.: zk-promises: Anonymous moderation, reputation, and blocking from anonymous credentials with callbacks. In: 34th USENIX Security Symposium (USENIX Security 25). pp. 4995–5014. USENIX Association, Seattle, WA (Aug 2025), https://www.usenix.org/conference/usenixsecurity25/presentation/shih
47. Todd, P.: Merkle mountain ranges. OpenTimestamps (Mar 2013), https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md, accessed: 2026-02-19
48. Weidner, M., Miller, H., Meiklejohn, C.: Composing and Decomposing Op-Based CRDTs with Semidirect Products (Apr 2020). https://doi.org/10.48550/arXiv.2004.04303, https://doi.org/10.48550/arXiv.2004.04303
49. Zhang, Y., Huang, Y., Wei, H., Ma, X.: MET: Model Checking-Driven Explorative Testing of CRDT Designs and Implementations (Apr 2022). https://doi.org/10.48550/arXiv.2204.14129, https://doi.org/10.48550/arXiv.2204.14129