SPAN: Privacy-Preserving Proofs of Sustained Aggregate Holdings in UTXO Privacy Pools

Zonglun Li[1] and Xue Liu[1,2]

[1] McGill University
zonglun.li@mail.mcgill.ca, xueliu@cs.mcgill.ca
[2] Mohamed Bin Zayed University of Artificial Intelligence

Abstract. Privacy pools are expanding from private transfers to tokenized real-world assets and holding-dependent rewards. These applications motivate evidence of sustained holdings across note replacements. We define a privacy-preserving interval proof of holding (PP-IPoH) scheme and present SPAN for arbitrary-value unspent transaction output (UTXO) pools. Two incrementally verifiable computations (IVCs) authenticate selected records and establish spend completeness, certifying a balance lower bound after every finalized transaction. An independent zero-knowledge succinct argument proves that this bound meets a threshold throughout an interval and supports subsequent queries over the same certified history. We prove completeness, knowledge soundness, and privacy under the stated component and ledger conditions. Experiments show linear proving time, adding about 2.9 seconds per additional transaction, and essentially constant maximum memory usage near 3.8 GB.

Keywords: privacy pools · privacy-preserving interval proof of holding · incrementally verifiable computation · authenticated data structures

1 Introduction

Privacy pools allow users to deposit assets and later withdraw without revealing the link to their deposits [2]. Their note-based designs build on Zerocash, which also underlies Zcash [3,2]. Tornado Cash Nova supports arbitrary-value deposits and private transfers [24]. In this unspent transaction output (UTXO) model, payments consume private input notes to create recipient and change outputs.

Privacy pools are also being developed for tokenized real-world assets (RWAs) and holding-dependent rewards [17,16,23]. For tokenized RWAs, DART combines confidential transfers with per-asset balance proofs [22]. Confidentiality and controlled data sharing also feature in the Bank for International Settlements' unified-ledger proposal [1]. Eligibility and rewards tied to retained assets introduce a need for verifiable evidence of holdings over time.

Eligibility to exercise shareholder rights can depend on sustained ownership. The United States' 2020 shareholder-proposal rule and Germany's meeting-agenda provisions have linked eligibility to maintaining a qualifying stake over a prescribed period [25,8]. Private tokenized shares call for the same historical evidence. Mixer rewards give another use for holding histories. AMR rewards

retained fixed-amount deposits, and Tornado Cash Classic awards points for deposit duration [16,23]. Both associate duration with individual notes.

These applications motivate a privacy-preserving interval proof of holding (PP-IPoH): evidence that a claimant maintained a specified aggregate quantity throughout a qualifying period while protecting the supporting notes and transfers. We consider homogeneous units of one asset. The proof supplies the historical holding component of an application-defined eligibility policy; identity binding, legal ownership, and any market valuation are application-level checks.

Existing holding mechanisms track individual notes. A fixed-note proof authenticates value, ownership, and creation before an interval, then establishes nullifier absence at its end [17]. In arbitrary-value UTXO pools, payments replace notes. Aggregate holdings may stay above a threshold after all initial notes are spent, while sufficient endpoint balances can hide an intermediate shortfall. The history must record every selected note's spend at its actual transaction position.

UltraMixer formalizes aggregate interval holding and gives a concrete fixed-note proof [17]. It outlines multi-nullifier batching and time-window segmentation for changing holdings; the reusable balance-history construction remains unspecified. SPAN supplies this construction. Two incrementally verifiable computations (IVCs) authenticate selected creations and spends, establish spend completeness, and certify a balance lower bound at every transaction state. An independent proof answers subsequent interval and threshold queries, reusing the history proofs and their verification results.

Technical Overview. A PP-IPoH scheme for a UTXO-based privacy pool must address three challenges:

1. **Authenticating holdings across note replacements.** Payments change the notes supporting a balance. A valid history must authenticate those changes and deduct every spend of a counted note at its actual transaction position.
2. **Protecting the supporting history.** The evidence identifies notes, amounts, and transaction positions. Certification must protect these private data while binding all component proofs to the same history.
3. **Controlling history and query costs.** A circuit that expands every historical check grows with the history. Repeating certification for each interval or threshold duplicates work over the same records.

SPAN combines the two IVCs with an external zero-knowledge succinct non-interactive argument of knowledge (SNARK). The transaction IVC authenticates selected records and balance changes. The lifecycle IVC matches creation and spend records and checks every note claimed unspent against the finalized nullifier set, establishing a balance lower bound. Hiding commitments link the proofs; the stipulated zero-knowledge interfaces protect their witnesses under the declared disclosure model. Bounded transitions certify the history, and a Merkle tree stores interval summaries for later minimum-balance queries.

Contributions. We present SPAN with three contributions:

1. **A formal holding construction.** We define PP-IPoH and construct it using two incrementally verifiable computations (IVCs) and an external query proof.

We prove completeness, knowledge soundness, and privacy under explicit cryptographic and ledger assumptions.
2. **Reusable histories with spend completeness.** Lifecycle consistency makes the recorded balance a lower bound at every transaction state. Bounded recursive steps certify the history; an independent query proof checks later intervals and thresholds using logarithmically many authenticated summaries.
3. **Implementation and evaluation.** We implement the full protocol with SuperNova and Groth16. Ten configurations demonstrate linear proving time, essentially constant maximum memory usage, low-cost query reuse, and a time–memory tradeoff with UltraMixer.

2 System Model and Preliminaries

To state the holding claim precisely, we fix the pool, its evidence, and the proof interfaces. We consider one privacy pool for a single fungible asset. Its public parameters, encodings, and proof verification keys are fixed and known to all parties. A user proves past holdings to a verifier over an authenticated session. Both parties derive their evidence from the same finalized pool log. Protocol operations take the fixed parameters implicitly.

2.1 Notation and Cryptographic Tools

Let λ be the security parameter and $\mathsf{negl}(\lambda)$ a negligible function. Algorithms are probabilistic polynomial time. Write $[n] = \{1, \ldots, n\}$, $[0] = \varnothing$, and $[a, b] = \{a, \ldots, b\}$. Write $x \xleftarrow{\$} X$ for uniform sampling and $\mathbf{1}[P]$ for the indicator of P. Typed encoding Encode is unambiguous. MSet discards only list order. Multiset-equality checks use $\mathbb{F}_p$, where p is prime. Hashing H uses domain-separation tags. We model the commitment and record domains and H_{FS} as classical random oracles [9]. We also use a pseudorandom function PRF and public-key encryption [3]. Hash and PRF calls abbreviate their fixed pool-specific encodings, detailed in Appendix A.9. Verification returns one on acceptance.

Commitments. A commitment fixes a private message, verified through its opening. We use $C \leftarrow \mathsf{Commit}(m; r)$ and $\mathsf{VerifyOpen}(C, m, r)$, for message m and randomness r. We require correctness, binding, and hiding. The user-key instance is Pedersen's $\mathsf{Com}(m; r) = g^m h^r$ in a prime-order group $\mathbb{G}$ of order r_G, with $m, r \in \mathbb{Z}_{r_G}$ and an unknown discrete-logarithm relation between g, h [21]. It is perfectly hiding and computationally binding. List and history hash commitments use independent private randomness for computational hiding in the random-oracle model. Appendix A.8 gives integer-preserving encodings between the commitment group's scalar field and $\mathbb{F}_p$.

Merkle authentication. A Merkle root authenticates records and their positions through Merkle authentication paths [19]; sparse trees also authenticate absent keys through empty leaves [7]. The interfaces $\mathsf{MT.Verify}(R, i, a; \pi_M)$ and $\mathsf{MT.VerifyMulti}(R, I, \mathbf{a}; \pi_M)$ check one or several complete records. The check $\mathsf{MT.VerifyUpdate}(R, R', i, a, a'; \pi_M)$ verifies a replacement using identical siblings; $\mathsf{SMT.VerifyNonMem}(R, k; \pi_M)$ checks an empty leaf. Fixed encodings and collision resistance bind these checks. Holding circuits take the paths privately (Appendix A.7).

Hash-chain commitments to lists. Hash chaining authenticates record order [13]. A randomized hash commitment hides and binds the final chain value and list length. For type t, $\mathsf{ListInit}_t$ initializes the count and hash-chain value, $\mathsf{Append}_t(J, i, a)$ appends record a at i, and $\mathsf{ListCom}_t(n, J; \xi)$ commits to the final hash-chain value and length. The check $\mathsf{VerifyList}_t(C, \mathbf{a}, \xi)$ scans every entry from that fixed initial state and verifies the commitment, including empty lists. IVC spreads this scan across transitions (Appendix A.9).

Randomized multiset equality. Random products compare contents and multiplicities across list orders, as in related permutation arguments [11]. Define $\mathsf{Prod}_\gamma(\mathbf{x}) = \prod_{i=1}^{n}(\gamma - x_i)$ for $\mathbf{x} \in \mathbb{F}_p^n$, with empty product one. Matching lengths and products test equality. For fixed unequal multisets and independent uniform γ, false acceptance is bounded by n/p, including zero factors. Challenges follow the record commitments. Appendix B.2 accounts for record-hash collisions, sampling bias, and adaptive attempts.

Zero-knowledge SNARKs. A zero-knowledge SNARK certifies knowledge of a private witness, protects its contents, and supports succinct verification [12]. Write $\mathcal{R}(y; w) = 1$ for statement y and witness w. Setup $\mathsf{SNARK.Setup}(1^\lambda, \mathcal{R})$ returns parameters (pk, vk); $\mathsf{SNARK.Prove}(pk, y; w)$ produces π, checked by $\mathsf{SNARK.Verify}(vk, y, \pi)$. We require completeness, knowledge soundness, zero knowledge, and succinct verification.

Non-uniform IVC. IVC extends a correctness proof as a computation advances; non-uniform IVC supports different step circuits at different steps [15,14]. Setup fixes initialization, a circuit family, and completion. Each step circuit's relation $\Phi_a(y; z, w, z') = 1$ requires its checks and update to hold for public statement y, previous state z, local witness w, and next state z'. The notation $z[f \leftarrow v]$ updates field f. The interfaces $\mathsf{IVC.Init}$, $\mathsf{IVC.ProveStep}$, and $\mathsf{IVC.Compress}$ initialize, extend, and compress. Verification $\mathsf{IVC.Verify}(y, \ell, \pi)$ binds initialization, selected transitions, and completion, disclosing count ℓ. For the execution relations of Section 4, we require complete, succinct final proofs with joint auxiliary-input knowledge extraction and auxiliary-input multi-theorem zero knowledge. Appendices B.3 and B.7 specify these security interfaces; Appendix A.7 gives the proof-system calls, with batching defined in Appendix A.4.

2.2 Privacy-Pool Interface and Finalized Data

Holding evidence comprises notes created by pool transactions and nullifiers revealed when those notes are spent. A *note* represents a transaction output by the private tuple $u = (v, pk^{\mathsf{sp}}, \rho, r)$: value v in asset units, spending public key pk^{sp}, unique note identifier ρ, and commitment randomness r. A spending-key derivation index d selects $sk^{\mathsf{sp}} = \mathsf{DeriveSK}(sk_{\mathsf{u}}, d)$ from private master secret sk_{u}. The predicate $\mathsf{Own}(sk_{\mathsf{u}}, pk^{\mathsf{sp}}, d) = 1$ checks $\mathsf{DerivePK}(sk^{\mathsf{sp}}) = pk^{\mathsf{sp}}$. Here sp means spending. Its commitment and nullifier are

$$cm(u) = \mathsf{H}(\mathtt{note}, v, pk^{\mathsf{sp}}, \rho, r), \tag{1}$$

$$nf(u, sk^{\mathsf{sp}}) = \mathsf{PRF}_{sk^{\mathsf{sp}}}(\mathtt{nullifier}, \rho). \tag{2}$$

The ledger stores $cm(u)$; u is its *note opening*, which reproduces the commitment. Control additionally requires the derived spending key. Write $v(u) = u.v$ and use

dot notation for other fields. Spending consumes the whole note; change is a fresh output. The two input and two output slots are indexed by $0, 1$. Dummy inputs have zero value and nullifier; holding records omit zero-value outputs. The pool checks ownership, output bounds, value conservation, and distinct nonzero input nullifiers absent from its nullifier set. The recipient decrypts the note ciphertext to recover u. Appendix A.1 gives the transfer interface.

Fix a finalized pool log with transactions numbered $1, \ldots, Q$. Let T_q be transaction q, S_q the state after it, and S_0 the initial empty state. The finalized log is $\mathcal{L}_Q = (T_1, \ldots, T_Q)$. Each transaction contains its input nullifiers and output commitments together with the pool's transfer data. Transactions within one block still define separate states.

The transaction Merkle tree has depth d_T and root R_T^Q; it authenticates (q, T_q). The sparse Merkle tree has depth d_N and root R_N^Q; it contains all nonzero nullifiers revealed by spends through Q. Both parties derive these roots from the finalized log [2]. Membership and non-membership paths are private proof witnesses. The pool-data encoding in Appendix A.1 fixes their verification.

2.3 Privacy-Preserving Interval Proof of Holding

We use the finalized states to define the balance and its interval guarantee. Let $\mathcal{U}_{sk_{\mathsf{u}}}(S_q)$ contain the notes created by S_q, still unspent there, and owned under the derivation above. Their balance is $\mathsf{Bal}_{sk_{\mathsf{u}}}(S_q) = \sum_{u \in \mathcal{U}_{sk_{\mathsf{u}}}(S_q)} v(u)$. For inclusive endpoints t_1, t_2 and threshold θ, define

$$\mathsf{Hold}(sk_{\mathsf{u}}, t_1, t_2, \theta) = 1 \iff \min_{q \in [t_1, t_2]} \mathsf{Bal}_{sk_{\mathsf{u}}}(S_q) \geq \theta. \tag{3}$$

The user publishes $C_{\mathsf{u}} = \mathsf{Com}(sk_{\mathsf{u}}; r_{\mathsf{u}})$ with secret randomizer r_{u}. Creation checks bind each counted note's spending key to this same secret. The shared history input is $X_Q = (C_{\mathsf{u}}, Q, R_T^Q, R_N^Q)$, and a request is $\mathsf{query} = (X_Q, t_1, t_2, \theta)$. History commitments bind X_Q throughout the construction. Queries may change their interval and threshold while reusing the same history through Q. Session fields are implicit in this request notation; Appendix A.10 gives their authenticated encoding.

Supported inputs. For history depth d_H, let $N = 2^{d_H}$. Let $V_{\max}$ bound note values and M bound balances and absolute interval summaries. Require $0 \leq t_1 \leq t_2 \leq Q < \min(N, 2^{d_T})$, $0 < v \leq V_{\max} \leq M$, $0 \leq \theta \leq M$, and $8M < p$. Counts, positions, and arithmetic obey the configured integer bounds. Sorting compares complete nullifiers. Appendix A.8 specifies record capacities and encodings. An *admissible* request meets these bounds, uses the correct finalized roots, and is authenticated in a valid fresh session.

Definition 1 (Privacy-preserving interval proof of holding). *A PP-IPoH scheme has polynomial-time algorithms* $\mathcal{S} = (\mathsf{Setup}, \mathsf{ProveHold}, \mathsf{VerifyHold})$ *with interfaces* $pp \leftarrow \mathsf{Setup}(1^\lambda)$, $\Pi \leftarrow \mathsf{ProveHold}(pp, \mathcal{L}_Q, \mathsf{query}; w)$, *and* $b \leftarrow \mathsf{VerifyHold}(pp, \mathcal{L}_Q, \mathsf{query}, \Pi) \in \{0, 1\}$. *Setup fixes the configuration and disclosure* Leak. *Session state and the associated public request fields are implicit, as specified in Appendix A.10. The witness* $w = (sk_{\mathsf{u}}, r_{\mathsf{u}}, \mathsf{notes})$ *includes user-key commitment and note openings, derivation indices, and creation locations. The output* Π *contains*

commitments and proofs. For admissible requests,

$$\mathcal{R}_{\mathsf{hold}}(\mathcal{L}_Q, \mathsf{query}; sk_{\mathsf{u}}, r_{\mathsf{u}}) = 1 \iff C_{\mathsf{u}} = \mathsf{Com}(sk_{\mathsf{u}}; r_{\mathsf{u}}) \ \wedge \ \mathsf{Hold}(sk_{\mathsf{u}}, t_1, t_2, \theta) = 1.$$

The scheme satisfies:

Completeness. *Let $\mathcal{N}$ collect the distinct owned notes selected across the history. At state S_q, only its members still in $\mathcal{U}_{sk_{\mathsf{u}}}(S_q)$ contribute. A witness is sufficient when these openings satisfy $\sum_{u \in \mathcal{N} \cap \mathcal{U}_{sk_{\mathsf{u}}}(S_q)} v(u) \geq \theta$ for every $q \in [t_1, t_2]$ and its full history through Q fits the supported bounds. For every admissible request and sufficient witness,*

$$\Pr[\mathsf{VerifyHold}(pp, \mathcal{L}_Q, \mathsf{query}, \mathsf{ProveHold}(pp, \mathcal{L}_Q, \mathsf{query}; w)) = 1] \geq 1 - \mathsf{negl}(\lambda),$$

over setup and prover randomness. Insufficient witnesses may return $\perp$.

Knowledge soundness. *For every probabilistic polynomial-time prover $\mathcal{A}$ with auxiliary input, there is an expected polynomial-time extractor E such that*

$$\Pr[\mathsf{VerifyHold}(pp, \mathcal{L}_Q, \mathsf{query}, \Pi) = 1 \ \wedge \mathcal{R}_{\mathsf{hold}}(\mathcal{L}_Q, \mathsf{query}; w_{\mathsf{ext}}) = 0] \leq \mathsf{negl}(\lambda).$$

Here $\mathcal{A}$ chooses the request and proof in the ledger experiment. The extractor E obtains $w_{\mathsf{ext}} = (sk_{\mathsf{u}}, r_{\mathsf{u}})$ from that execution. Extraction failure yields $\perp$, whose relation value is zero. Probability covers setup, adversary and extractor randomness, and consistent oracle answers.

Privacy. *For public data* pub *(including commitments) and proof objects* proofs*, an efficient simulator* Sim *satisfies* $(\mathsf{pub}, \mathsf{proofs}) \approx_c (\mathsf{pub}, \mathsf{Sim}(\mathsf{pub}))$*. For every efficient verifier $\mathcal{V}^*$ and two sufficient witness collections with identical declared disclosure* Leak*, require* $\mathsf{View}_0^{\mathcal{V}^*} \approx_c \mathsf{View}_1^{\mathcal{V}^*}$*. Each view includes its world's record and history commitments and proofs; $\approx_c$ denotes computational indistinguishability. Both conditions hold for polynomially many adaptive permitted requests, with the setups and simulation trapdoors of Appendix B.7.*

Adversaries may control users and indexers, omit records, and adapt attempts. We assume ledger finality, correct pool execution, deterministic nullifiers, and prevention of double spending. SPAN discloses the ledger, parameters, user commitment, requests and outcomes, commitment linkage, and both IVC counts. Denote this profile $\mathsf{Leak}_{\mathrm{SPAN}}$. Section 5 states its guarantee under Appendix B's proof assumptions.

3 Authenticated Records and Balance History

To certify a balance through changing notes, the prover records each selected note's creation and spend, pairs them into lifetimes, and stores their net changes in an authenticated history. These objects supply the evidence checked in Section 4.

Alice's transactions illustrate how replacing notes affects aggregate holdings. Alice deposits twelve at position 10, spends it at 20 to pay Bob five and receive seven in change, receives three at 30, then replaces seven plus three with six and four at 40. Her balances are 12, 7, 10, and 10. The minimum on $[15, 40]$ is seven. Both endpoints reach ten, while the balance at 20 falls below ten. Tracking each note's lifetime accounts for these intermediate changes.

3.1 Record Types and List Commitments

The prover records each selected note's identity, value, and creation location to match it to its eventual spend. Given its private opening u, derivation index d, creating transaction position b, and output slot $s \in \{0, 1\}$, define

$$\mathsf{Record}(u, d, b, s) = \mathsf{rec} = (cm(u), nf(u, \mathsf{DeriveSK}(sk_{\mathsf{u}}, d)), v(u), b, s). \tag{4}$$

Fields $\mathsf{rec}.cm, \mathsf{rec}.nf, \mathsf{rec}.v, \mathsf{rec}.b, \mathsf{rec}.s$ give its commitment, nullifier, value, creation position, and output slot.

Appending one such record for each selected output gives the *creation list* $\mathcal{C} = (\mathsf{rec}_i^{\mathsf{C}})_{i=1}^{n_C}$, where n_C is its private length. Entries follow transaction order, then output-slot order. A creation record stays in this list after its note is spent. The consumption is appended separately to the *spend list* $\mathcal{D} = ((\mathsf{rec}_i^{\mathsf{D}}, e_i^{\mathsf{D}}))_{i=1}^{n_D}$. Each entry carries a claimed copy of the note's creation record and its consumption position e_i^{D}. Its private length is n_D. Entries follow transaction order, then input-slot order. Alice's payment appends the consumed twelve-unit note to $\mathcal{D}$ and the seven-unit change note to $\mathcal{C}$.

To check note lifetimes, the prover pairs each creation record with its claimed spend position in the *lifecycle table* $\mathcal{L}_{\mathsf{note}} = ((\mathsf{rec}_i^{\mathsf{L}}, e_i^{\mathsf{L}}))_{i=1}^{n_L}$, with private length n_L. Sorting by complete nullifier allows uniqueness checks between adjacent rows. The superscripts $\mathsf{C}, \mathsf{D}, \mathsf{L}$ indicate the containing list. Their row indices are independent. A note contributes from S_b through the state preceding S_e, where b is its creation position and $e \le Q$ its spend position. The marker $e = Q + 1$ includes S_Q. The table must satisfy

$$\begin{aligned} \mathsf{MSet}(\mathcal{C}) &= \mathsf{MSet}((\mathsf{rec}_i^{\mathsf{L}})_{i=1}^{n_L}), \qquad n_L = n_C, \\ \mathsf{MSet}(\mathcal{D}) &= \mathsf{MSet}((\mathsf{rec}_i^{\mathsf{L}}, e_i^{\mathsf{L}})_{i:e_i^{\mathsf{L}} \le Q}), \\ 1 \le \mathsf{rec}_i^{\mathsf{L}}.b < e_i^{\mathsf{L}} &\le Q + 1, \qquad \mathsf{rec}_i^{\mathsf{L}}.nf < \mathsf{rec}_{i+1}^{\mathsf{L}}.nf. \end{aligned} \tag{5}$$

The equalities preserve creation records, spend pairs, and multiplicities; the inequalities enforce lifetime order and uniqueness. The creation records from rows marked $Q + 1$ form the *remaining-record sequence* $\mathcal{A}$, of length $n_A = |\mathcal{A}|$. Its nullifiers are checked in table order.

The prover commits to the three ordered lists as C_C, C_D, C_L, using independent private random salts ξ_C, ξ_D, ξ_L of at least λ bits. The list-commitment interface of Section 2.1 binds each list's contents, order, and length to X_Q. The running hash-chain values for creations and spends are J_C, J_D. A validating scan starts from the fixed initial hash-chain value, reads the declared number of records, and verifies their commitment. The audit rescans both lists in their committed orders and compares them with the nullifier-sorted table. The lifecycle scan also fixes the length n_A and chain value J_A of $\mathcal{A}$. Appendix A.9 gives the hash-chain encoding.

3.2 Summary Composition and Tree Authentication

At position $q \ge 1$, the selected creations and spends contribute net change x_q. Set $x_0 = 0$; unprocessed positions retain zero. The recorded balance is $B_q^{\mathsf{hist}} = \sum_{i=0}^{q} x_i$.

The audit establishes that each counted note's spend is recorded at its actual position. Together with creation authentication, this makes B_q^{hist} a lower bound on the actual balance, even when other receipts are omitted.

For changes $X = [-5, +3, -4, +2]$, adding entries from left to right gives $0, -5, -2, -6, -4$, including the initial zero. The summary $\Sigma(X) = (\Delta(X), \mu(X)) = (-4, -6)$ records the total and smallest prefix sum. Starting at twelve gives a final balance of eight and a minimum of six. For adjacent sequences L, R,

$$(\Delta_L, \mu_L) \otimes (\Delta_R, \mu_R) = (\Delta_L + \Delta_R, \min\{\mu_L, \Delta_L + \mu_R\}). \tag{6}$$

A prefix ends in L, with minimum μ_L, or enters R after accumulating Δ_L, giving $\Delta_L + \mu_R$. For $L = [-5, +3]$ and $R = [-4, +2]$, summaries $(-2, -5)$ and $(-2, -4)$ combine to $(-4, -6)$: the right minimum shifts to $-2 - 4 = -6$. Composition preserves order and is associative; the empty summary is $(0, 0)$.

A Merkle tree stores the N changes and their subtree summaries. A leaf stores (used, x, τ): a flag marking a processed position, its balance change, and a digest of the transaction and updates. The unused leaf is $(0, 0, 0)$; a written leaf has summary $(x, \min(0, x))$. Setting the bit records note replacements with $x = 0$. Each node record contains its hash, total, and minimum prefix sum. The parent hash binds both complete child records and their combined summary from equation (6). Initialization with the empty history and the proved updates establish summary correctness; Merkle paths authenticate those summaries during queries. Each satisfies $-M \le \Delta \le M$ and $-M \le \mu \le 0$. The private root R_{hist} is hidden by $C_{\mathsf{hist}} = \mathsf{H}(\texttt{hist-commit}, X_Q, \xi_H, R_{\mathsf{hist}})$ with independent secret salt ξ_H.

Authenticated update. For transaction position q, $\mathsf{HistUpdate}$ authenticates the replacement of the unused leaf $(0, 0, 0)$ by $(1, x_q, \tau_q)$ under the old and proposed roots. Both authentication paths use the same sibling node records and summaries. Each ancestor's summary is recomputed by equation (6), and all summaries satisfy the prescribed integer bounds. Section 4.1 binds x_q and τ_q to the selected transaction records. Appendix A.2 gives the exact path relation. Section 4.4 uses the resulting authenticated summaries to check an interval minimum.

4 The SPAN Protocol

SPAN uses three proof-system instantiations. The transaction-record IVC, $\mathsf{IVC}_{\mathsf{tx}}$, authenticates selected creation and claimed-spend records and their balance changes. The lifecycle-consistency IVC, $\mathsf{IVC}_{\mathsf{audit}}$, checks record correspondence and verifies that the remaining notes are unspent, establishing a balance lower bound through Q. The query SNARK, $\mathsf{SNARK}_{\min}$, proves the requested interval threshold. Setup fixes each system's proving and verification parameters for the relations defined below; shared commitments connect their proofs.

Set $y_{\mathsf{tx}} = (X_Q, C_C, C_D, C_{\mathsf{hist}})$, $Y_Q = (X_Q, C_C, C_D, C_L, C_{\mathsf{hist}})$, and $y_{\min} = (Y_Q, \mathsf{query})$ for their public statements. Algorithm 1 prepares the private execution witnesses $W_{\mathsf{tx}}, W_{\mathsf{audit}}$ and query witness $w_{\min}$, using the retained history tree $\mathcal{T}_{\mathsf{hist}}$. Commitments fix the statements before proving, and the lifecycle commitment precedes the multiset-equality challenges. Appendix A.6 specifies witness preparation and the instances' input encodings.

Algorithm 1 SPAN: history certification, interval query, and verification

Require: Request query; finalized log; fixed proof parameters
Witness: Opening of the user-key commitment and selected note openings

procedure ProveHold($\mathsf{query}; sk_{\mathsf{u}}, r_{\mathsf{u}}, \mathsf{notes}$)
  Check the request's roots against the finalized log.
  Prepare $\mathcal{C}, \mathcal{D}, \mathcal{T}_{\mathsf{hist}}, W_{\mathsf{tx}}$ from the selected notes and log.
  Sample independent salts; fix $C_C, C_D, C_{\mathsf{hist}}$ and y_{tx}.
  Complete W_{tx} with $\mathsf{FinishTx}$.
  $\pi_{\mathsf{tx}} \leftarrow \mathsf{IVC}_{\mathsf{tx}}.\mathsf{Prove}(y_{\mathsf{tx}}; W_{\mathsf{tx}})$
  $\mathcal{L}_{\mathsf{note}} \leftarrow \mathsf{Sort}_{nf}(\mathsf{Lifetimes}(\mathcal{C}, \mathcal{D}, Q+1))$
  Commit the table as C_L; fix Y_Q, then derive (γ_C, γ_D).
  Obtain paths for $\mathcal{A}$ under R_N^Q and prepare W_{audit}.
  $\pi_{\mathsf{audit}} \leftarrow \mathsf{IVC}_{\mathsf{audit}}.\mathsf{Prove}(Y_Q; W_{\mathsf{audit}})$
  Prepare and check $y_{\min}, w_{\min}$ from the request and retained history.
  $\pi_{\min} \leftarrow \mathsf{SNARK}_{\min}.\mathsf{Prove}(y_{\min}; w_{\min})$
  return $(C_C, C_D, C_L, C_{\mathsf{hist}}, \pi_{\mathsf{tx}}, \pi_{\mathsf{audit}}, \pi_{\min})$
end procedure
procedure VerifyHold(query, Π)
  Check finalized roots, request bounds, and session validity.
  Unpack Π; form $y_{\mathsf{tx}}, Y_Q, y_{\min}$ with the request's X_Q.
  Recompute (γ_C, γ_D) from Y_Q.
  assert $\mathsf{IVC}_{\mathsf{tx}}.\mathsf{Verify}(y_{\mathsf{tx}}, \pi_{\mathsf{tx}}) = 1$
  assert $\mathsf{IVC}_{\mathsf{audit}}.\mathsf{Verify}(Y_Q, \pi_{\mathsf{audit}}) = 1$
  assert $\mathsf{SNARK}_{\min}.\mathsf{Verify}(y_{\min}, \pi_{\min}) = 1$
  Mark the session request used; **return** 1.
end procedure

All checks run off chain. Each IVC's Prove performs initialization, successive $\mathsf{ProveStep}$ calls, and final compression. Its fixed circuit family checks the permitted operations and phase changes. A phase groups one task; q, k, i index transactions, logical transitions, and scanned records. Unassigned fields retain their values. The integer bounds of Section 2.3 apply throughout, and Appendix A.4 specifies the scan batching.

4.1 Transaction-Record IVC

$\mathsf{IVC}_{\mathsf{tx}}$ uses three step circuits to authenticate selected records and their balance history. InitTx checks the opening of the user-key commitment and initializes empty lists and history. Repeated $\mathsf{ProcessTx}$ calls authenticate selected outputs and published spend nullifiers at increasing transaction positions, updating the records and balance history together. $\mathsf{FinishTx}$ checks the resulting commitments and completes processing. The audit then matches each claimed spend record to its authenticated creation. Under public statement y_{tx} and private witness W_{tx}, the application state is

$$z^{\mathsf{tx}} = (\phi, q_{\mathsf{last}}, B^{\mathsf{hist}}, R_{\mathsf{hist}}, J_C, J_D, n_C, n_D). \tag{7}$$

The phase ϕ and last position q_{last} track progress; the remaining fields carry the balance, history root, and record hash-chain values and counts. InitTx implements $\Phi^{\mathsf{tx}}_{\mathsf{init}}(X_Q; sk_{\mathsf{u}}, r_{\mathsf{u}}, z_0) = 1$: it checks the key-commitment opening and fixes

phase UPDATE, zero balance, position, and counts, and fixed initial hash-chain values and empty history in z_0. $\mathsf{IVC}_{\mathsf{tx}}.\mathsf{Init}$ starts the proof under these conditions (Appendix A.3).

Each subsequent $\mathsf{IVC}_{\mathsf{tx}}.\mathsf{ProveStep}$ certifies $\Phi^{\mathsf{tx}}_{\mathsf{step}}(y_{\mathsf{tx}}; z, w, z') = 1$, the relation for one transition from private state z to z'. The local witness w contains an operation tag selecting ProcessTx or FinishTx and that step circuit's inputs. The relation enforces the selected step circuit's phase, checks, and updates. The two check groups below jointly define ProcessTx.

Transaction authentication. The prover supplies the accepted transaction (q, T_q), its membership path π_T, and the key-commitment opening. The step requires phase UPDATE, $q_{\mathsf{last}} < q \le Q$, the same commitment opening, and $\mathsf{MT.Verify}(R^Q_T, q, T_q; \pi_T) = 1$. Each selected slot contributes once. For a selected output slot j, the witness provides opening u_j and derivation index d_j. The step verifies the opening against that slot's commitment, checks $\mathsf{Own}(sk_{\mathsf{u}}, u_j.pk^{\mathsf{sp}}, d_j) = 1$ and $0 < v(u_j) \le V_{\max}$, and computes $\mathsf{Record}(u_j, d_j, q, j)$ with a nonzero nullifier. For a selected input slot i, the witness supplies a claimed creation record a^{in}_i. Its nullifier must equal the nonzero nullifier published in that slot, and its value must lie in $(0, V_{\max}]$. The resulting spend record is (a^{in}_i, q).

Record and balance-history updates. Append these creation and spend records to their respective lists in slot order. The same selections determine $x_q = \sum_{\text{selected outputs } j} v(u_j) - \sum_{\text{selected inputs } i} a^{\mathsf{in}}_i.v$. The step checks $-M \le x_q \le M$ and $0 \le B^{\mathsf{hist}} + x_q \le M$. The digest τ_q binds (q, T_q), both record-chain updates, and x_q to X_Q. Using the proposed root R'_{hist} and private path π_H, it verifies $\mathsf{HistUpdate}(R_{\mathsf{hist}}, q, x_q, \tau_q, R'_{\mathsf{hist}}; \pi_H) = 1$. It stores the updated balance, root, chains, position, and record counts, checking the count capacities. Equal selected input and output totals give $x_q = 0$; their records still identify the old and replacement notes for the lifecycle audit.

Completion. The FinishTx branch of $\Phi^{\mathsf{tx}}_{\mathsf{step}}$ requires phase UPDATE. It opens C_C, C_D with the computed hash-chain values, lengths, and private salts, and C_{hist} with the final root and its salt. It then returns $\mathbf{0}_{\mathsf{RECORDED}}$, the state with phase RECORDED and all other registers zero; y_{tx} remains public. After this final $\mathsf{IVC}_{\mathsf{tx}}.\mathsf{ProveStep}$, $\mathsf{IVC}_{\mathsf{tx}}.\mathsf{Compress}$ produces π_{tx}. The execution witness W_{tx} contains the commitment opening, initial and successive states z_k, and local witnesses w_k, each with its operation tag and inputs. For ℓ_{tx} transitions, including completion, the complete execution relation is

$$\begin{aligned} &\mathcal{R}_{\mathsf{tx}}(y_{\mathsf{tx}}; W_{\mathsf{tx}}) = 1 \iff \Phi^{\mathsf{tx}}_{\mathsf{init}}(X_Q; sk_{\mathsf{u}}, r_{\mathsf{u}}, z_0) = 1 \\ &\wedge \bigwedge\nolimits_{k=1}^{\ell_{\mathsf{tx}}} \Phi^{\mathsf{tx}}_{\mathsf{step}}(y_{\mathsf{tx}}; z_{k-1}, w_k, z_k) = 1 \wedge z_{\ell_{\mathsf{tx}}} = \mathbf{0}_{\mathsf{RECORDED}}. \end{aligned} \tag{8}$$

The relation links initialization, every transition, and the prescribed terminal state. It authenticates the ordered records and history; the audit establishes their creation–spend correspondence. Appendix A.3 gives the exact transition algorithms.

4.2 Committed Lifecycle Table and Multiset Equality

The lifecycle table associates each deduction with its creation. Its rows pair complete creation records with claimed spend positions or $Q+1$, ordered by full nullifier. The prover fixes its commitment C_L, and hence Y_Q, before deriving

$$\gamma_C = H_{\mathsf{FS}}(\texttt{create-match}, Y_Q), \qquad \gamma_D = H_{\mathsf{FS}}(\texttt{spend-match}, Y_Q). \tag{9}$$

The verifier uses the fixed hash-to-field sampler and distinct domains in equation (9). These challenges depend on the committed history; later intervals and thresholds reuse the same audit statement.

List commitments fix the complete scans; multiset-equality checks compare their records across the two orders. For a complete creation record a and endpoint e, define $f_C(a) = \mathsf{H}(\texttt{create-record}, X_Q, a)$ and $f_D(a,e) = \mathsf{H}(\texttt{spend-record}, X_Q, a, e)$ in $\mathbb{F}_p$. Products P_C, P_D use the creation and spend lists; $P_C^{\mathsf{L}}, P_D^{\mathsf{L}}$ use the corresponding lifecycle entries:

$$\begin{aligned} P_C &= \prod\nolimits_{a\in\mathcal{C}} (\gamma_C - f_C(a)), & P_C^{\mathsf{L}} &= \prod\nolimits_{i=1}^{n_L} (\gamma_C - f_C(\mathsf{rec}_i^{\mathsf{L}})), \\ P_D &= \prod\nolimits_{(a,e)\in\mathcal{D}} (\gamma_D - f_D(a,e)), & P_D^{\mathsf{L}} &= \prod\nolimits_{e_i^{\mathsf{L}}\le Q} (\gamma_D - f_D(\mathsf{rec}_i^{\mathsf{L}}, e_i^{\mathsf{L}})). \end{aligned} \tag{10}$$

The audit checks $P_C = P_C^{\mathsf{L}}$ and $P_D = P_D^{\mathsf{L}}$ with the corresponding lengths. Products start at one and retain multiplicities. Appendix B.2 bounds false acceptance, including zero factors, sampling bias, and adaptive attempts. Each attempt uses its prescribed challenges. The following IVC authenticates complete scans of the compared lists.

4.3 Lifecycle-Consistency IVC

$\mathsf{IVC}_{\mathsf{audit}}$ uses nine step circuits to verify each counted note's lifetime through Q. InitAudit fixes declared lengths, salts, and multiset-equality challenges. ScanCreate and EndCreate authenticate the creation list $\mathcal{C}$; ScanSpend and EndSpend then authenticate the spend list $\mathcal{D}$. ScanLife checks the lifecycle table and derives the remaining sequence $\mathcal{A}$. BeginCheck closes the table scan, verifies record matching, and fixes $\mathcal{A}$ for the final scan. CheckNote verifies each remaining nullifier's non-membership under R_N^Q; FinishCheck checks that the entire sequence was examined. End-of-scan checks follow each complete scan, including empty lists.

Under Y_Q, the fixed private parameters Ω specify the declared lengths and commitment salts. InitAudit implements $\Phi_{\mathsf{init}}^{\mathsf{audit}}(Y_Q; \Omega, z_0) = 1$, checking $n_L = n_C$, $0 \le n_D \le n_C$, capacities, and the challenges derived from Y_Q. It sets phase $\mathsf{SCAN}_{\mathsf{C}}$, products one, counters and previous-value registers zero, and fixed initial hash-chain values. $\mathsf{IVC}_{\mathsf{audit}}$.Init starts the proof under these conditions (Appendix A.4).

Each subsequent $\mathsf{IVC}_{\mathsf{audit}}$.ProveStep certifies $\Phi_{\mathsf{step}}^{\mathsf{audit}}(Y_Q; z, w, z') = 1$ for one of the eight processing and end-of-scan step circuits. The tag in w selects a step circuit permitted by the current phase. The relation checks the selected transition from z to z' under the conditions below. Nonterminal transitions preserve Ω and the challenges.

Creation and spend list scans. The source scans compute P_C, P_D over the lists authenticated by π_{tx}. In $\mathsf{SCAN}_{\mathsf{C}}$, ScanCreate reads the next complete record

within the declared length and updates its chain and product. EndCreate checks exactly n_C entries and the opening of C_C using that chain and fixed salt, then enters $\mathsf{SCAN_D}$. ScanSpend processes complete spend pairs analogously; EndSpend checks length n_D and commitment C_D, then enters $\mathsf{SCAN_L}$. Each new scan starts with its fixed initial hash-chain value and zero count, retaining the products.

Lifecycle scan. In phase $\mathsf{SCAN_L}$, the ScanLife branch of $\Phi^{\mathsf{audit}}_{\mathsf{step}}$ processes the next row (a, e) within the declared table length. The complete creation record a and endpoint e use canonical encodings and satisfy $1 \le a.b < e \le Q+1$, $0 \le a.s < 2$, $0 < a.v \le V_{\max}$, and $a.nf \neq 0$. Consecutive full nullifiers must be strictly increasing. The row advances the lifecycle chain and contributes $\gamma_C - f_C(a)$ to P^{L}_C. For $e \le Q$, it also contributes $\gamma_D - f_D(a, e)$ to P^{L}_D and counts as a spend. For $e = Q+1$, its record a is appended to $\mathcal{A}$. Processing exactly n_L rows determines that sequence's length and chain (n_A, J_A).

Remaining-note checks. The BeginCheck branch closes $\mathsf{SCAN_L}$: it checks the table length, opening of C_L, both product equalities, and $n_L = n_C = n_D + n_A$. The table must contain exactly n_D rows with spending positions at most Q. It fixes (n_A, J_A), enters CHECK, and initializes checked count i_A to zero and hash-chain value J_{check} to the initial value used for J_A. The CheckNote branch uses a complete record a and path π_N to check $\mathsf{SMT.VerifyNonMem}(R^Q_N, a.nf; \pi_N) = 1$, append a to the checked chain, and advance the count, up to n_A. In phase CHECK, the terminal FinishCheck branch requires

$i_A = n_A, \qquad J_{\mathsf{check}} = J_A$, then returns $\mathbf{0}_{\mathsf{DONE}}$, with phase DONE and all other registers zero; Y_Q remains public. The checked equalities bind the records to the entire remaining sequence in order. Creation matching supplies their authenticated ownership and amounts; this final scan establishes their unspentness through Q. The call $\mathsf{IVC_{audit}.Compress}$ then produces π_{audit}.

Audit statement. The execution witness W_{audit} contains Ω, states z_k, and local witnesses w_k with their operation tags and inputs. For ℓ_{audit} transitions, the complete relation joins the initialization and step relations with the prescribed terminal state:

$$\begin{aligned} &\mathcal{R}_{\mathsf{audit}}(Y_Q; W_{\mathsf{audit}}) = 1 \iff \Phi^{\mathsf{audit}}_{\mathsf{init}}(Y_Q; \Omega, z_0) = 1 \\ &\wedge \bigwedge\nolimits_{k=1}^{\ell_{\mathsf{audit}}} \Phi^{\mathsf{audit}}_{\mathsf{step}}(Y_Q; z_{k-1}, w_k, z_k) = 1 \wedge z_{\ell_{\mathsf{audit}}} = \mathbf{0}_{\mathsf{DONE}}. \end{aligned} \tag{11}$$

The verifier checks π_{tx} with $\mathsf{IVC_{tx}.Verify}$ and π_{audit} with $\mathsf{IVC_{audit}.Verify}$ against their shared $C_C, C_D, C_{\mathsf{hist}}$. Section 5 derives the query's balance lower bound from these authenticated records and lifetimes.

4.4 Query Proof and End-to-End Verification

$\mathsf{SNARK}_{\min}$ certifies the interval-query relation $\mathcal{R}_{\min}(y_{\min}; w_{\min})$. Every balance in $[t_1, t_2]$ equals the starting balance plus the changes since t_1. The relation authenticates that starting balance and the minimum subsequent cumulative change under one history root, then checks the threshold. Its public statement is $y_{\min} = (Y_Q, \mathsf{query})$; its witness $w_{\min} = (R_{\mathsf{hist}}, \xi_H, \mathcal{F})$ supplies the root, commitment salt, and joint Merkle multiproof. It checks the same X_Q in the request and Y_Q, bounds $0 \le t_1 \le t_2 \le Q < N$ and $0 \le \theta \le M$, and the opening of C_{hist}.

The public endpoints determine a canonical decomposition of $[0, t_1]$ and $[t_1 + 1, t_2]$ into disjoint tree intervals covering each range exactly. The multiproof authenticates their node records under R_{hist}. Combining each range's summaries by equation (6) in left-to-right order gives the prefix total $\Delta_P = B^{\mathsf{hist}}_{t_1}$ and window minimum prefix sum μ_W. For the certified history, the threshold test is

$$\Delta_P + \mu_W = \min_{q \in [t_1, t_2]} B^{\mathsf{hist}}_q \geq \theta. \tag{12}$$

The relation bounds the minimum minus θ in $[0, M]$. For $t_1 = t_2$, the window summary is $(0, 0)$. After preparing $w_{\min}$, the prover calls $\mathsf{SNARK}_{\min}.\mathsf{Prove}(y_{\min}; w_{\min})$. The verifier checks $\mathsf{SNARK}_{\min}.\mathsf{Verify}(y_{\min}, \pi_{\min}) = 1$. Appendix A.5 specifies the complete relation and witness preparation; Appendix A.10 gives the authenticated input encoding used by both methods.

Algorithm 1 verifies all three proofs on their shared inputs. Once both history proofs have been verified, later requests reuse Y_Q with their own request checks and query proof. Section 5 connects these checks to Definition 1.

5 Security Guarantees

Appendix B proves the guarantee below. The combined bound $\epsilon_{\mathsf{match}}$ covers matching, hashing, binding, sampling, and extraction failures. The term ϵ_{pool} covers pool and finalized-data failures.

Theorem 1 (SPAN realizes PP-IPoH). *With components satisfying Section 2.1 and the pool and finality conditions of Section 2.3, SPAN satisfies Definition 1 with disclosure* $\mathsf{Leak}_{\mathrm{SPAN}}$*. Its knowledge-soundness error is at most* $\epsilon_{\mathsf{match}} + \epsilon_{\mathsf{pool}}$*, with* $\epsilon_{\mathsf{match}}$ *bounded by equation* (17)*, and its privacy is computational. Errors are negligible for exponentially large* p*, polynomial executions and query counts, and negligible component failure terms.*

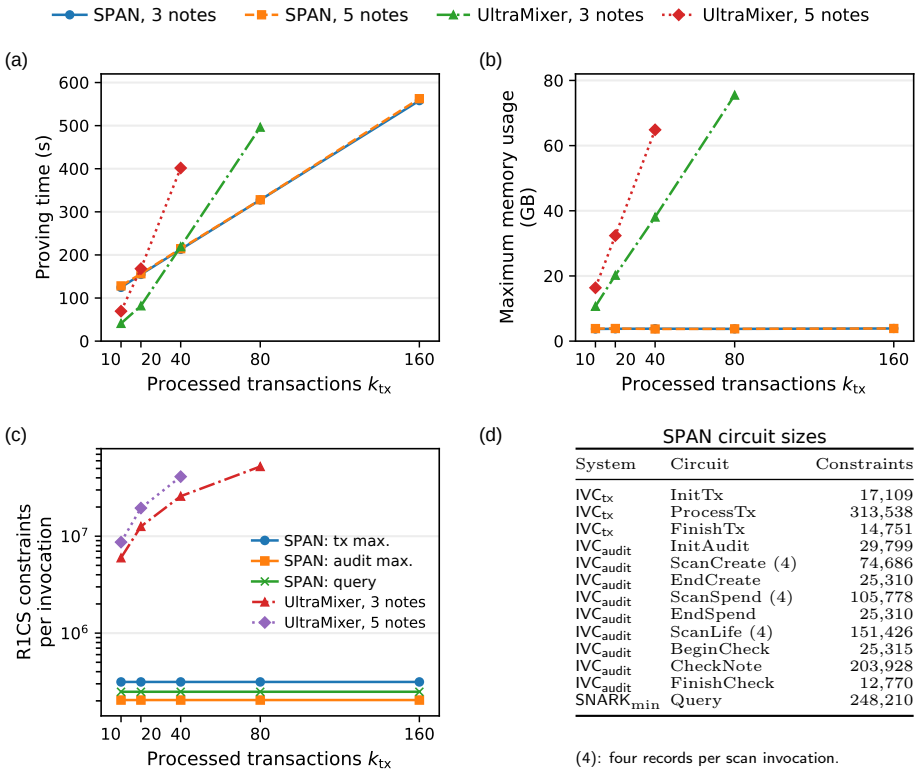
Proof outline. Condition on valid extracted witnesses, binding, and exact record correspondence. Write $v_i = \mathsf{rec}^{\mathsf{L}}_i.v$, $b_i = \mathsf{rec}^{\mathsf{L}}_i.b$, and $e_i = e^{\mathsf{L}}_i$. Creation authentication and strict nullifier order identify distinct owned notes. An endpoint at most Q is the actual spend position. Each row marked $Q + 1$ passes its required nullifier non-membership check at Q. Since the same records write the history changes, every $0 \leq q \leq Q$ satisfies

$$B^{\mathsf{hist}}_q = \sum\nolimits_{b_i \leq q} v_i - \sum\nolimits_{e_i \leq q} v_i = \sum\nolimits_{i=1}^{n_C} v_i \mathbf{1}[b_i \leq q < e_i] \leq \mathsf{Bal}_{sk_u}(S_q). \tag{13}$$

A counted note consumed by Q must have its actual spend endpoint: another endpoint breaks record correspondence, and the marker $Q + 1$ fails the nullifier check. This covers spends in unprocessed gaps and through Q after the last processed position. The query checks that this recorded lower bound meets θ throughout the interval. Honest sufficient selections give completeness. The component simulators establish the first privacy condition; commitment-hiding hybrids establish the second, as shown in Appendix B.7.

6 Implementation and Evaluation

We measure history certification and subsequent queries using the full SPAN implementation: Arecibo 0.1.1 SuperNova over BN256 and Grumpkin, Spartan with IPA compression, and gnark Groth16 over BN254 [18,14,5,12].



System	Circuit	Constraints
IVC_{tx}	InitTx	17,109
IVC_{tx}	ProcessTx	313,538
IVC_{tx}	FinishTx	14,751
IVC_{audit}	InitAudit	29,799
IVC_{audit}	ScanCreate (4)	74,686
IVC_{audit}	EndCreate	25,310
IVC_{audit}	ScanSpend (4)	105,778
IVC_{audit}	EndSpend	25,310
IVC_{audit}	ScanLife (4)	151,426
IVC_{audit}	BeginCheck	25,315
IVC_{audit}	CheckNote	203,928
IVC_{audit}	FinishCheck	12,770
$SNARK_{min}$	Query	248,210

(4): four records per scan invocation.

Fig. 1. Proving time (a), maximum memory usage (b), constraints per invocation (c), and SPAN circuit sizes with their proof systems (d). The upper plots share a legend. All transaction axes are linear; (c) uses a logarithmic vertical scale. IVC entries count application step constraints; the query uses Groth16.

Experiments use an Apple M1 Pro MacBook Pro and four workers. Workloads maintain three or five selected live notes, with $k_{tx} \in \{10, 20, 40, 80, 160\}$ transactions including two or three funding transactions, respectively. Tree depths are $d_T = d_H = 32$ and $d_N = 254$. The three list scans use four-record batches; CheckNote processes one note per invocation. We compare with UltraMixer [17].

History certification. The largest transaction and audit step circuits have 313,538 and 203,928 constraints; the query circuit has 248,210. These per-invocation sizes stay fixed at the configured depths and batch sizes. All n_A remaining notes are checked in n_A CheckNote invocations. Figure 1, bottom left, contrasts these fixed step sizes with UltraMixer's history-dependent complete interval circuit, and Figure 1, bottom right, lists all SPAN step and query circuits.

Proving time ranges from 124.66 to 562.58 seconds. Proving time grows linearly with the number of transactions covered: each additional transaction adds about 2.9 seconds for either live-note count (Figure 1(a)).

Maximum memory usage stays essentially constant at about 3.8 GB as history length increases sixteenfold, well below UltraMixer's growing memory usage (Figure 1(b)). For three notes and 80 transactions, SPAN uses 3.763 GB against UltraMixer's 75.503 GB, a 95.0% reduction. For five notes and 40 transactions, the values are 3.723 and 64.849 GB, a 94.3% reduction.

The comparison shows a time–memory tradeoff. UltraMixer proves faster for three notes with 10 or 20 transactions and five notes with 10, while SPAN uses much less memory. As more transactions are covered, SPAN also gains a growing proving-time advantage. At three notes and 80 transactions, it takes 327.64 seconds against UltraMixer's 496.44; at five notes and 40 transactions, it takes 214.46 against 401.68. These give speedups of 1.52 and 1.87 times. The remaining three UltraMixer configurations exceed the experiment machine's hardware capacity.

Reusable interval queries. For fixed Q and shared commitments, later queries reuse the history proofs and cached verification results. In the three-note series, query proving takes 1.798–1.830 seconds, accounting for 0.33% of proving time at 160 transactions. Both SPAN and UltraMixer have constant-time proof verification.

7 Related Work

Private balance statements differ in their time scope and accounting model. DART's account-based proofs disclose per-asset finalized and pending balances [22]. Provisions combines private Bitcoin assets and liabilities at a snapshot and distinguishes key control from ownership [6]. SPAN's predicate concerns the minimum balance of one asset controlled by a key over a past interval. Authenticity and completeness also arise in private ledger queries: zkLedger enforces auditing completeness and caches selected rows [20], while SPHINX proves computations over authenticated streams and sliding windows [10]. SPAN uses the note model's creation and single-spend semantics to obtain the required historical balance bound.

Within privacy pools, AMR and Tornado Cash associate rewards with holding periods of individual fixed-denomination notes [16,23]. UltraMixer formalizes aggregate interval holding and gives a fixed-note proof [17]. Its multi-nullifier and time-window discussion outlines the extension to changing holdings. SPAN constructs a reusable balance history with spend completeness and independent queries. Derecho studies a different disclosure, the provenance of funds through private transfers [2]. The mechanisms used here build on established IVC [15,14], permutation arguments [11], and accumulation-based memory proofs [4]; the lifecycle checks connect these tools to the interval-holding statement.

8 Conclusion

SPAN realizes PP-IPoH for changing notes under the stated assumptions. Its transaction and lifecycle checks certify a reusable balance history for independent

interval queries. Experiments show linear proving time, adding about 2.9 seconds per additional transaction, and essentially constant maximum memory usage near 3.8 GB. As histories grow, the time–memory tradeoff increasingly favors SPAN. Subsequent queries over the same finalized history take about 1.8 seconds to prove.

References

1. Bank for International Settlements: Blueprint for the future monetary system: Improving the old, enabling the new. In: Annual Economic Report 2023, chap. III. Bank for International Settlements (2023), https://www.bis.org/publ/arpdf/ar2023e3.htm
2. Beal, J., Fisch, B.: Derecho: Privacy pools with proof-carrying disclosures. In: ACM SIGSAC Conference on Computer and Communications Security. pp. 3197–3211 (2024). https://doi.org/10.1145/3658644.3670270, https://eprint.iacr.org/2023/273
3. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: IEEE Symposium on Security and Privacy. pp. 459–474 (2014). https://doi.org/10.1109/SP.2014.36
4. Bünz, B., Chen, J.: Proofs for deep thought: Accumulation for large memories and deterministic computations. Cryptology ePrint Archive, Paper 2024/325 (2024), https://eprint.iacr.org/2024/325, revised March 2025
5. Consensys: gnark: Fast zero-knowledge proofs in Go. Software, https://github.com/Consensys/gnark
6. Dagher, G.G., Bünz, B., Bonneau, J., Clark, J., Boneh, D.: Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges. In: ACM SIGSAC Conference on Computer and Communications Security (2015), https://www.jbonneau.com/doc/DBBCB15-CCS-provisions.pdf
7. Dahlberg, R., Pulls, T., Peeters, R.: Efficient sparse merkle trees: Caching strategies and secure (non-)membership proofs. In: Secure IT Systems—NordSec 2016. LNCS, vol. 10014, pp. 199–215. Springer (2016). https://doi.org/10.1007/978-3-319-47560-8_13
8. Federal Republic of Germany: Aktiengesetz (stock corporation act), section 122. Gesetze im Internet, paragraphs (1)–(2), https://www.gesetze-im-internet.de/aktg/__122.html, accessed 20 September 2026
9. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Advances in Cryptology—CRYPTO '86. LNCS, vol. 263, pp. 186–194. Springer (1987). https://doi.org/10.1007/3-540-47721-7_12
10. Fiore, D., Tucker, I.: Efficient zero-knowledge proofs on signed data with applications to verifiable computation on data streams. In: ACM SIGSAC Conference on Computer and Communications Security. pp. 1067–1080 (2022). https://doi.org/10.1145/3548606.3560630, https://eprint.iacr.org/2022/1393
11. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953 (2019), https://eprint.iacr.org/2019/953
12. Groth, J.: On the size of pairing-based non-interactive arguments. In: Advances in Cryptology—EUROCRYPT 2016. LNCS, vol. 9666, pp. 305–326. Springer (2016). https://doi.org/10.1007/978-3-662-49896-5_11
13. Haber, S., Stornetta, W.S.: How to time-stamp a digital document. Journal of Cryptology **3**(2), 99–111 (1991). https://doi.org/10.1007/BF00196791
14. Kothapalli, A., Setty, S.: SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Paper 2022/1758 (2022), https://eprint.iacr.org/2022/1758
15. Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive zero-knowledge arguments from folding schemes. In: Advances in Cryptology—CRYPTO 2022. pp. 359–388. Springer (2022), https://eprint.iacr.org/2021/370

16. Le, D.V., Gervais, A.: AMR: Autonomous coin mixer with privacy preserving reward distribution. In: ACM Conference on Advances in Financial Technologies. pp. 142–155 (2021). https://doi.org/10.1145/3479722.3480800, https://arxiv.org/abs/2010.01056
17. Li, Z., Kang, H., Liu, X.: UltraMixer: A compliant zero-knowledge privacy layer for tokenized real-world assets. Cryptology ePrint Archive, Paper 2025/1715 (2025), https://eprint.iacr.org/2025/1715
18. Lurk Contributors: Arecibo: Recursive proofs of computation. Version 0.1.1, software, https://github.com/lurk-lang/arecibo
19. Merkle, R.C.: A certified digital signature. In: Advances in Cryptology—CRYPTO '89. LNCS, vol. 435, pp. 218–238. Springer (1990). https://doi.org/10.1007/0-387-34805-0_21
20. Narula, N., Vasquez, W., Virza, M.: zkLedger: Privacy-preserving auditing for distributed ledgers. In: USENIX Symposium on Networked Systems Design and Implementation. pp. 65–80 (2018), https://www.usenix.org/conference/nsdi18/presentation/narula
21. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Advances in Cryptology—CRYPTO '91. LNCS, vol. 576, pp. 129–140. Springer (1992). https://doi.org/10.1007/3-540-46766-1_9
22. Sarencheh, A., Khoshakhlagh, H., Kavousi, A., Kiayias, A.: DART: Decentralized, anonymous, and regulation-friendly tokenization. Cryptology ePrint Archive, Paper 2025/239 (2025), https://eprint.iacr.org/2025/239, balance interface cited from the author manuscript uploaded 15 February 2025
23. Tornado Cash Community: Tornado Cash Classic: Anonymity mining. Protocol documentation (2021), https://github.com/tornadocash/docs/blob/en/tornado-cash-classic/anonymity-mining.md
24. Tornado Cash Community: Tornado Nova. Protocol implementation (2021), https://github.com/tornadocash/tornado-nova
25. U.S. Securities and Exchange Commission: Procedural requirements and resubmission thresholds under exchange act rule 14a-8. Final rule, Release No. 34-89964 (2020), https://www.sec.gov/files/rules/final/2020/34-89964.pdf, 23 September 2020, Section II.A.3.i

A Complete Protocol Algorithms

The algorithms instantiate the relations of Section 4. Each assertion is a circuit constraint; failure returns $\perp$. Updates preserve unassigned fields, and disabled slots use constrained identity transitions. Native witness preparation records the same assignments. A scan call handles one record; Appendix A.4 specifies batching. Brackets $[P]$ return a proposition's Boolean value, $\leftarrow$ assigns a value, and $\|$ concatenates sequences.

A.1 Base Pool Interface

For the pool interface of Section 2.2, an authenticated payment request carries the amount, pool, spending and encryption public keys, nonce, and expiry. The channel authenticates the recipient; spending secrets remain private. Over the two transaction slots, conservation requires $\sum_i v_i^{\mathsf{in}} + D = \sum_j v_j^{\mathsf{out}} + W + F$, where D, W, F are the public deposit, withdrawal, and fee.

The public input is $(R_C, \mathbf{nf}, \mathbf{cm}^{\mathsf{out}}, \mathsf{H}(\mathbf{ct}), D, W, F, addr_{\mathsf{out}}, addr_{\mathsf{rel}})$. Here R_C is a recognized coin-commitment root, $\mathbf{ct}$ contains note ciphertexts, and the addresses receive the withdrawal and fee. The witness contains input openings, Merkle paths and spending keys, output openings, recipient encryption keys, and encryption randomness. For each real input, the relation recomputes the note commitment, authenticates it under R_C, and checks the spending key and published nullifier. It enforces the dummy-input and value rules of Section 2.2, recomputes output commitments and ciphertexts, binds public addresses, and checks $D \cdot W = 0$. Deposits use dummy inputs and $D > 0$; private transfers use $D = W = 0$; withdrawals use $W > 0$ and may create change. After verifying the proof and fresh distinct nullifiers, the contract records them, appends the two commitments, executes the public transfer, and emits T_q.

The finalized transaction order is the lexicographic order of block number, transaction index, and log index. Date-based applications supply the corresponding endpoints. The transaction leaf is $\ell_q^T = \mathsf{H}(\texttt{transaction-leaf}, q, T_q)$; $\mathsf{MT.Verify}(R_T^Q, q, T_q; \pi_T)$ applies this encoding before checking the path. The nullifier tree uses the fixed complete-nullifier-to-key mapping; $\mathsf{SMT.VerifyNonMem}$ authenticates an empty leaf and rejects conflicting occupied values at the same key.

A.2 Balance-History Updates

For the node record $\mathsf{desc}_v = (h_v, \Delta_v, \mu_v)$ of Section 3.2, $\mathsf{Leaf}(\mathsf{used}, x, \tau)$ returns summary $(x, \min(0, x))$ and hash $\mathsf{H}(\texttt{hist-leaf}, \mathsf{used}, x, \tau, x, \min(0, x))$. At height k, $\mathsf{Parent}_k(L, R)$ combines the ordered child summaries by equation (6) and hashes $\mathsf{H}(\texttt{hist-node}, k, \mathsf{desc}_L, \mathsf{desc}_R, \Delta_P, \mu_P)$. Set $\mathsf{Root}(D) = \mathsf{H}(\texttt{hist-root}, N, D)$.

In Algorithm 2, D_k is the sibling record at height k, $\mathsf{Bits}_{d_H}(q)$ lists bits least-significant first, and $\mathsf{Bounds}(D)$ checks $-M \le D.\Delta \le M$ and $-M \le D.\mu \le 0$. Empty-subtree records satisfy $E_0 = \mathsf{Leaf}(0, 0, 0)$ and $E_{k+1} = \mathsf{Parent}_{k+1}(E_k, E_k)$, giving $R_{\mathsf{hist}}^0 = \mathsf{Root}(E_{d_H})$ in $O(d_H)$ work.

A.3 Transaction IVC

Algorithms 3 and 4 implement the processing transitions of Section 4.1, using its prescribed initialization. In Algorithm 3, ω contains the listed local witness; $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ select input and output slots. The functions $nf_i(T_q)$ and $cm_j^{\mathsf{out}}(T_q)$ read the published slot values. Each slot has a canonical encoding. The sums V^+, V^- have enough bits for two note values, and disabled slots preserve chains and counts.

Algorithm 2 $\mathsf{HistUpdate}(R, q, x, \tau, R'; \pi_H)$

Require: Old root R, position q, change x, digest τ, new root R'
Witness: Sibling node records $\pi_H = (D_0, \ldots, D_{d_H-1})$
assert $1 \le q < N \ \wedge\ -M \le x \le M$
$(b_0, \ldots, b_{d_H-1}) \leftarrow \mathsf{Bits}_{d_H}(q)$
$A \leftarrow \mathsf{Leaf}(0,0,0)$; $A' \leftarrow \mathsf{Leaf}(1, x, \tau)$
for $k = 0, \ldots, d_H - 1$ **do**
  assert $\mathsf{Bounds}(D_k) = 1$
  if $b_k = 0$ **then**
    $A \leftarrow \mathsf{Parent}_{k+1}(A, D_k)$; $A' \leftarrow \mathsf{Parent}_{k+1}(A', D_k)$
  else
    $A \leftarrow \mathsf{Parent}_{k+1}(D_k, A)$; $A' \leftarrow \mathsf{Parent}_{k+1}(D_k, A')$
  end if
  assert $\mathsf{Bounds}(A) = \mathsf{Bounds}(A') = 1$
end for
return $[\mathsf{Root}(A) = R] \wedge [\mathsf{Root}(A') = R']$

Algorithm 3 $\mathsf{ProcessTx}(y_{\mathsf{tx}}; z, \omega)$

Require: Public y_{tx}, state z of equation (7)
Witness:$(sk_{\mathsf{u}}, r_{\mathsf{u}}, q, T_q, \pi_T, \boldsymbol{\alpha}, \boldsymbol{\beta}, (a_i^{\mathsf{in}}), (u_j, d_j), R', \pi_H)$
assert $z.\phi = \mathsf{UPDATE} \ \wedge\ z.q_{\mathsf{last}} < q \le Q$
assert $C_{\mathsf{u}} = \mathsf{Com}(sk_{\mathsf{u}}; r_{\mathsf{u}})$
assert $\mathsf{MT.Verify}(R_T^Q, q, T_q; \pi_T) = 1$
assert $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \{0,1\}^2$
$z' \leftarrow z$; $V^+ \leftarrow 0$; $V^- \leftarrow 0$
for $j = 0, 1$ with $\beta_j = 1$ **do**
  assert $cm(u_j) = cm_j^{\mathsf{out}}(T_q) \ \wedge\ \mathsf{Own}(sk_{\mathsf{u}}, u_j.pk^{\mathsf{sp}}, d_j) = 1$
  assert $0 < v(u_j) \le V_{\max}$
  $a \leftarrow \mathsf{Record}(u_j, d_j, q, j)$; **assert** $a.nf \neq 0$
  $z'.n_C \leftarrow z'.n_C + 1$; $z'.J_C \leftarrow \mathsf{Append}_C(z'.J_C, z'.n_C, a)$
  $V^+ \leftarrow V^+ + a.v$
end for
for $i = 0, 1$ with $\alpha_i = 1$ **do**
  assert $a_i^{\mathsf{in}}.nf = nf_i(T_q) \neq 0 \ \wedge\ 0 < a_i^{\mathsf{in}}.v \le V_{\max}$
  $z'.n_D \leftarrow z'.n_D + 1$; $z'.J_D \leftarrow \mathsf{Append}_D(z'.J_D, z'.n_D, (a_i^{\mathsf{in}}, q))$
  $V^- \leftarrow V^- + a_i^{\mathsf{in}}.v$
end for
$x \leftarrow V^+ - V^-$; $B' \leftarrow z.B^{\mathsf{hist}} + x$
assert $0 \le B' \le M \ \wedge\ -M \le x \le M \ \wedge\ z'.n_C \le n_C^{\max} \ \wedge\ z'.n_D \le n_D^{\max}$
$\tau \leftarrow \mathsf{H}(\texttt{history-tx}, X_Q, q, \mathsf{H}(T_q), z.J_C, z.J_D, z'.J_C, z'.J_D, x)$
assert $\mathsf{HistUpdate}(z.R_{\mathsf{hist}}, q, x, \tau, R'; \pi_H) = 1$
return $z'[q_{\mathsf{last}} \leftarrow q, B^{\mathsf{hist}} \leftarrow B', R_{\mathsf{hist}} \leftarrow R']$

Algorithm 4 $\mathsf{FinishTx}(y_{\mathsf{tx}}; z, \xi_C, \xi_D, \xi_H)$

Require: The immutable public y_{tx} and current private state z
Witness: Three independent commitment salts
assert $z.\phi = \mathsf{UPDATE}$
assert $C_C = \mathsf{ListCom}_C(z.n_C, z.J_C; \xi_C)$
assert $C_D = \mathsf{ListCom}_D(z.n_D, z.J_D; \xi_D)$
assert $C_{\mathsf{hist}} = \mathsf{H}(\texttt{hist-commit}, X_Q, \xi_H, z.R_{\mathsf{hist}})$
return $\mathbf{0}_{\mathsf{RECORDED}}$

A.4 Lifecycle IVC

The fixed private parameters are $\Omega = (n_C, n_D, n_L, \xi_C, \xi_D, \xi_L)$; the full audit state is:

$$z^{\mathsf{audit}} = (\phi, \Omega, i, J, P_C, P_D, P_C^{\mathsf{L}}, P_D^{\mathsf{L}}, \mathsf{hasPrev}, nf_{\mathsf{prev}}, d_{\mathsf{seen}}, n_A, J_A, i_A, J_{\mathsf{check}}). \tag{14}$$

The pair (i, J) tracks the current list prefix; $\mathsf{hasPrev}$ and nf_{prev} retain the previous full nullifier. The counter d_{seen} counts rows with spend endpoints. The pairs (n_A, J_A) and $(i_A, J_{\mathsf{check}})$ track the derived and checked remaining sequences. Initialization sets $z_0 = (\mathsf{SCAN_C}, \Omega, 0, J_{C,0}, 1, 1, 1, 1, 0, 0, 0, 0, J_{A,0}, 0, J_{A,0})$, checks $0 \le n_L = n_C \le n_C^{\max}$, $0 \le n_D \le \min(n_C, n_D^{\max})$, canonical salts, and equation (9). Unqualified lengths and salts refer to Ω; other registers refer to the input state. Algorithms 5–7 give the individual transitions.

Algorithm 5 Creation and spend list scans and end-of-scan checks

Require: Y_Q and private audit state z
Witness: One full record for a scan, or $\perp$ for an end-of-scan check
procedure ScanCreate$(z; a)$
  assert $\phi = \mathsf{SCAN_C} \ \wedge \ i < n_C$
  return $z[i \leftarrow i+1, J \leftarrow \mathsf{Append}_C(J, i+1, a), P_C \leftarrow P_C(\gamma_C - f_C(a))]$
end procedure
procedure EndCreate(z)
  assert $\phi = \mathsf{SCAN_C} \ \wedge \ i = n_C \ \wedge \ C_C = \mathsf{ListCom}_C(n_C, J; \xi_C)$
  return $z[\phi \leftarrow \mathsf{SCAN_D}, i \leftarrow 0, J \leftarrow J_{D,0}]$
end procedure
procedure ScanSpend$(z; (a, e))$
  assert $\phi = \mathsf{SCAN_D} \ \wedge \ i < n_D$
  return $z[i \leftarrow i+1, J \leftarrow \mathsf{Append}_D(J, i+1, (a, e)), P_D \leftarrow P_D(\gamma_D - f_D(a, e))]$
end procedure
procedure EndSpend(z)
  assert $\phi = \mathsf{SCAN_D} \ \wedge \ i = n_D \ \wedge \ C_D = \mathsf{ListCom}_D(n_D, J; \xi_D)$
  return $z[\phi \leftarrow \mathsf{SCAN_L}, i \leftarrow 0, J \leftarrow J_{L,0}]$
end procedure

Algorithm 6 $\mathsf{ScanLife}(Y_Q; z, (a, e))$

Require: Y_Q and audit state z
Witness: One canonically encoded lifecycle row (a, e)
assert $z.\phi = \mathsf{SCAN_L} \ \wedge \ z.i < n_L$
assert $1 \le a.b < e \le Q + 1 \ \wedge \ 0 \le a.s < 2 \ \wedge \ 0 < a.v \le V_{\max} \ \wedge \ a.nf \neq 0$
if $z.\mathsf{hasPrev} = 1$ **then**
  assert $z.nf_{\mathsf{prev}} < a.nf$
end if
$z' \leftarrow z$; $\delta \leftarrow \mathbf{1}[e \le Q]$
$z'.i \leftarrow i + 1$; $z'.J \leftarrow \mathsf{Append}_L(J, i + 1, (a, e))$
$z'.P_C^{\mathsf{L}} \leftarrow P_C^{\mathsf{L}}(\gamma_C - f_C(a))$
$z'.P_D^{\mathsf{L}} \leftarrow P_D^{\mathsf{L}}(1 - \delta + \delta(\gamma_D - f_D(a, e)))$
$z'.d_{\mathsf{seen}} \leftarrow d_{\mathsf{seen}} + \delta$
if $\delta = 0$ **then**
  $z'.n_A \leftarrow n_A + 1$; $z'.J_A \leftarrow \mathsf{Append}_A(J_A, n_A + 1, a)$
end if
$z'.\mathsf{hasPrev} \leftarrow 1$; $z'.nf_{\mathsf{prev}} \leftarrow a.nf$
return z'

Algorithm 7 Remaining-note scan and termination

Require: Y_Q and the private audit state
Witness:(a, π_N) for a remaining-note check; $\perp$ for an end-of-scan check
procedure BeginCheck(z)
  assert $\phi = \mathsf{SCAN_L} \ \wedge \ i = n_L = n_C$
  assert $C_L = \mathsf{ListCom}_L(n_L, J; \xi_L)$
  assert $d_{\mathsf{seen}} = n_D \ \wedge \ n_C = n_D + n_A \ \wedge \ P_C = P_C^{\mathsf{L}} \ \wedge \ P_D = P_D^{\mathsf{L}}$
  return $z[\phi \leftarrow \mathsf{CHECK}, i_A \leftarrow 0, J_{\mathsf{check}} \leftarrow J_{A,0}]$
end procedure
procedure CheckNote($z; (a, \pi_N)$)
  assert $\phi = \mathsf{CHECK} \ \wedge \ i_A < n_A$
  assert $\mathsf{SMT.VerifyNonMem}(R_N^Q, a.nf; \pi_N) = 1$
  return $z[i_A \leftarrow i_A + 1, J_{\mathsf{check}} \leftarrow \mathsf{Append}_A(J_{\mathsf{check}}, i_A + 1, a)]$
end procedure
procedure FinishCheck(z)
  assert $\phi = \mathsf{CHECK} \ \wedge \ i_A = n_A \ \wedge \ J_{\mathsf{check}} = J_A$
  return $\mathbf{0}_{\mathsf{DONE}}$
end procedure

Four-record scan batches. A batched ScanCreate, ScanSpend, or ScanLife composes up to four single-record transitions. Enabled slots form a prefix; each checks the current counter against the declared length and processes the next record. Disabled slots preserve every register. The closing checks in Algorithm 5 enforce the exact length and commitment opening. Products are initialized to one in InitAudit and carried through every scan and check preceding FinishCheck. CheckNote handles one note. Each batch advances the logical trace by its number of enabled records, while counting as one backend invocation; initialization and end-of-scan circuits have separate invocations.

A.5 Joint Query Authentication

To implement $\mathcal{R}_{\min}$ from Section 4.4, $\mathsf{BoundaryOpen}$ authenticates the root and the prefix and window node lists $\mathcal{C}_P, \mathcal{C}_W$. The function Fold applies equation (6) left to right from $(0, 0)$. The arithmetic checks are

$$\begin{aligned} (R, \mathcal{C}_P, \mathcal{C}_W) &= \mathsf{BoundaryOpen}(t_1, t_2; \mathcal{F}), \quad R = R_{\mathsf{hist}}, \\ (\Delta_P, \mu_P) &= \mathsf{Fold}(\mathcal{C}_P), \quad (\Delta_W, \mu_W) = \mathsf{Fold}(\mathcal{C}_W), \\ \zeta &= \Delta_P + \mu_W - \theta, \qquad 0 \le \zeta \le M. \end{aligned} \tag{15}$$

The request encoding is enforced as specified in Appendix A.10. In Algorithm 8, $\mathsf{Visit}(k, l, r)$ traverses height k and interval $[l, r]$ of multiproof $\mathcal{F}$. Its lists P, W collect prefix and window records; $[D]$ and $[\,]$ denote singleton and empty lists. Subscripts L, R distinguish child results. The traversal splits nodes crossing $t_1 + 1$ or $t_2 + 1$. For $N = 8$ and request $[2, 5]$, the prefix uses $[0, 1], [2, 2]$, the window uses $[3, 3], [4, 5]$, and $[6, 7]$ contributes only to root reconstruction. The routine $\mathsf{Take}(\mathcal{F}, k, l, r)$ consumes the next record at its prescribed position, rejecting missing or invalid data.

Algorithm 8 $\mathsf{BoundaryOpen}(t_1, t_2; \mathcal{F})$

Require: $0 \le t_1 \le t_2 < N = 2^{d_H}$
Witness: Node-record sequence $\mathcal{F}$ for the joint multiproof
procedure VISIT(k, l, r)
  if $r \le t_1$ **or** $(t_1 < l \wedge r \le t_2)$ **or** $l > t_2$ **then**
    $D \leftarrow \mathsf{Take}(\mathcal{F}, k, l, r)$; **assert** $\mathsf{Bounds}(D) = 1$
    $P \leftarrow [D]$ if $r \le t_1$, otherwise $P \leftarrow [\,]$
    $W \leftarrow [D]$ if $t_1 < l \wedge r \le t_2$, otherwise $W \leftarrow [\,]$
    return (D, P, W)
  end if
  assert $k > 0 \ \wedge \ l < r$
  $m \leftarrow \lfloor (l + r)/2 \rfloor$
  $(D_L, P_L, W_L) \leftarrow$ VISIT$(k - 1, l, m)$
  $(D_R, P_R, W_R) \leftarrow$ VISIT$(k - 1, m + 1, r)$
  $D \leftarrow \mathsf{Parent}_k(D_L, D_R)$; **assert** $\mathsf{Bounds}(D) = 1$
  return $(D, P_L \| P_R, W_L \| W_R)$
end procedure
$(D, \mathcal{C}_P, \mathcal{C}_W) \leftarrow$ VISIT$(d_H, 0, N - 1)$
assert $\mathcal{F}$ is exhausted
return $(\mathsf{Root}(D), \mathcal{C}_P, \mathcal{C}_W)$

The two boundary paths require $O(d_H)$ node records and parent computations, using constrained branches and bounded witness slots. The same traversal covers $t_1 = 0$ and $t_1 = t_2$. The retained tree $\mathcal{T}_{\mathsf{hist}}$ stores these records and root record $\mathsf{desc}_{\mathsf{root}}$. The routine $\mathsf{CoverWitness}$ reads the records selected by Algorithm 8, giving $\mathcal{F}$ and witness $w_{\min} = (\mathsf{Root}(\mathsf{desc}_{\mathsf{root}}), \xi_H, \mathcal{F})$. The calls in Section 4.4 pass

the fixed keys $pk_{\min}, vk_{\min}$ and $y_{\mathsf{enc}} = \mathsf{EncodeQuery}(y_{\min}; \mathsf{session})$ to the generic SNARK interfaces; proving uses the prescribed internal randomness.

A.6 Witness Preparation

Algorithm 1 uses the selected openings and derivation indices, with creation locations from the log. Its root checks call $\mathsf{DeriveRoots}(\mathsf{ledger}, Q)$ for (R_T^Q, R_N^Q). Failed prover checks return $\perp$; failed verifier checks return zero. The preparation routines are:

The call $\mathsf{TraceTx}(X_Q; sk_{\mathsf{u}}, r_{\mathsf{u}}, \mathsf{notes})$ produces $\mathcal{C}, \mathcal{D}, \mathcal{T}_{\mathsf{hist}}, W_{\mathsf{tx}}$. Independent salts $\xi_C, \xi_D, \xi_L, \xi_H \xleftarrow{\$} \{0,1\}^\lambda$ form the commitments. Given y_{tx}, the trace and its salts, $\mathsf{CompleteTx}$ appends Algorithm 4; the retained tree $\mathcal{T}_{\mathsf{hist}}$ supplies later query witnesses. $\mathsf{Lifetimes}$ pairs each creation with its spend or $Q+1$; Sort_{nf} orders the rows, giving $C_L = \mathsf{ListCom}_L(n_L, J_{L,n_L}; \xi_L)$. $\mathsf{Challenge}(Y_Q)$ derives equation (9). Finally, $\mathsf{TraceAudit}(Y_Q; \Omega, \mathcal{C}, \mathcal{D}, \mathcal{L}_{\mathsf{note}}, \mathsf{paths}_N)$ builds W_{audit}, with one path under R_N^Q for each remaining record. Appendix A.7 gives the proof-instance calls; Appendix A.10 specifies the session checks.

A.7 Supplementary Component Interfaces

Witnesses for the checks in Section 2.1 are generated from local authenticated data: $(R, \mathcal{T}) \leftarrow \mathsf{MT.Build}(\mathbf{a})$ constructs the tree, $\mathsf{MT.Open}(\mathcal{T}, i)$ returns one authentication path, and $\mathsf{MT.OpenMulti}(\mathcal{T}, I)$ opens several positions. A position may denote a leaf or subtree root carrying its complete record.

For $s \in \{\mathsf{tx}, \mathsf{audit}\}$, z_k is the application state and A_k the IVC prover state. The call $\mathsf{IVC}_s.\mathsf{Init}(y; z_0, w_0)$ uses initialization witness w_0 to create A_0. Then $\mathsf{IVC}_s.\mathsf{ProveStep}(A_{k-1}; a_k, z_k, w_k)$ returns A_k; the operation tag in w_k must agree with circuit a_k and the current phase. Compression yields $\pi \leftarrow \mathsf{IVC}_s.\mathsf{Compress}(A_\ell)$. The call $\mathsf{IVC}_s.\mathsf{Verify}(y, \ell, \pi)$ binds the public statement and count to the prescribed terminal state: $\mathbf{0}_{\mathsf{RECORDED}}$ for tx and $\mathbf{0}_{\mathsf{DONE}}$ for audit. Its abbreviation is $\mathsf{Verify}(y, \pi)$. The logical traces and batched invocations are related as in Appendix A.4.

A.8 Supported Encodings and Capacities

In addition to the bounds of Section 2.3, capacities $n_C^{\max}, n_D^{\max} < p$ bound selected creations and spends. Canonical encodings and explicit cross-field constraints preserve integer values between the commitment group's scalar field and $\mathbb{F}_p$. The unspent marker $Q+1$ uses $d_H + 1$ bits, allowing $Q = N - 1$ and marker N; query positions stay below N. Signed values use $[-M, M]$ with sufficiently wide intermediates. Nullifier comparisons use all ℓ_{nf} bits of the canonical encoding.

A.9 Fixed Pool Parameters and Hash Encodings

Setup fixes identifiers poolID and ppID for the pool and its complete parameter set. Their encoded public tuples are

$$X_Q^{\mathsf{enc}} = (\mathsf{poolID}, C_{\mathsf{u}}, Q, R_T^Q, R_N^Q), \qquad Y_Q^{\mathsf{enc}} = (X_Q^{\mathsf{enc}}, C_C, C_D, C_L, C_{\mathsf{hist}}).$$

The superscript enc denotes the encoding supplied to the transaction and audit prover and verifier interfaces.

Let H_0 be the hash on canonical encodings. The pool-specific commitment and nullifier in equations (1) and (2) expand to $\mathsf{H}_0(\texttt{note}, \mathsf{poolID}, v, pk^{\mathsf{sp}}, \rho, r)$ and $\mathsf{PRF}_{sk^{\mathsf{sp}}}(\texttt{nullifier}, \mathsf{poolID}, \rho)$, respectively. The transaction leaf expands to $\mathsf{H}_0(\texttt{transaction-leaf}, \mathsf{poolID}, q, T_q)$.

For list type $t \in \{C, D, L, A\}$, $J_{t,i}$ is the chain value after i records, and tag_t is its fixed tag.

The canonical hash encodings are

$$\begin{aligned} J_{t,0} &= \mathsf{H}_0(\texttt{list-init}, (\mathsf{ppID}, X_Q^{\mathsf{enc}}), \mathsf{tag}_t), \\ \mathsf{Append}_t(J, i, a) &= \mathsf{H}_0(\texttt{list-step}, \mathsf{tag}_t, J, i, \mathsf{Encode}(a)), \\ \mathsf{ListCom}_t(n, J; \xi) &= \mathsf{H}_0(\texttt{list-commit}, (\mathsf{ppID}, X_Q^{\mathsf{enc}}), \mathsf{tag}_t, \xi, n, J). \end{aligned} \tag{16}$$

$\mathsf{ListInit}_t$ returns $(0, J_{t,0})$. Record hashes and the history commitment expand as follows:

$$\begin{aligned} f_C(a) &= \mathsf{H}_0(\texttt{create-record}, (\mathsf{ppID}, X_Q^{\mathsf{enc}}), \mathsf{Encode}(a)), \\ f_D(a, e) &= \mathsf{H}_0(\texttt{spend-record}, (\mathsf{ppID}, X_Q^{\mathsf{enc}}), \mathsf{Encode}(a), e), \\ C_{\mathsf{hist}} &= \mathsf{H}_0(\texttt{hist-commit}, (\mathsf{ppID}, X_Q^{\mathsf{enc}}), \xi_H, R_{\mathsf{hist}}). \end{aligned}$$

The digest τ_q in algorithm 3 expands to

$$\tau_q = \mathsf{H}_0\big(\texttt{history-tx}, (\mathsf{ppID}, X_Q^{\mathsf{enc}}), q, \mathsf{H}_0(T_q), J_C, J_D, J'_C, J'_D, x_q\big).$$

The challenge inputs are $(\texttt{create-match}, \mathsf{ppID}, Y_Q^{\mathsf{enc}})$ and $(\texttt{spend-match}, \mathsf{ppID}, Y_Q^{\mathsf{enc}})$. They use the fixed hash-to-field interface of equation (9). Node encodings are fixed in Appendix A.2. The expanded history input binds the user commitment, endpoint, and both finalized roots.

A.10 Authenticated Request Encoding

The mathematical request $\mathsf{query} = (X_Q, t_1, t_2, \theta)$ is exchanged over an authenticated session. The session supplies the public verifier identifier vid, fresh nonce ν, and deadline expiry. The complete request used by the concrete proof interface is

$$\mathsf{request} = (X_Q^{\mathsf{enc}}, t_1, t_2, \theta, \mathsf{vid}, \nu, \mathsf{expiry}).$$

For $y = (Y_Q, \mathsf{query})$, the deterministic routine $\mathsf{EncodeQuery}(y; \mathsf{session})$ produces the public SNARK input

$$\eta = \mathsf{H}_0(\texttt{hist-query}, \mathsf{ppID}, Y_Q^{\mathsf{enc}}, \mathsf{request}), \qquad y_{\mathsf{enc}} = ((Y_Q^{\mathsf{enc}}, \mathsf{request}), \eta).$$

The verifier computes this encoding, and the query circuit enforces its digest together with the checks of Section 4.4. The notation $\mathcal{R}_{\min}(y; w)$ includes these constraints. All session fields belong to the public view in Definition 1.

$\mathsf{SessionValid}(\mathsf{session}, C_{\mathsf{u}}, \mathsf{vid}, \nu, \mathsf{expiry}) = 1$ authenticates the channel and its association with C_{u}, checks the verifier identifier and deadline, and verifies nonce freshness. After all proofs accept, $\mathsf{MarkUsed}(\mathsf{session}, \nu)$ records the nonce. Received identifiers must match the fixed configuration. The reuse rule in Section 4.4 uses the same Y_Q^{enc} and verification keys, with these session checks for each request.

B Security Analysis and Composition

We prove Theorem 1 from record correspondence, randomized matching, and the component proof assumptions. The ledger model is fixed in Section 2.3; extraction failure is bounded by ϵ_{ext}.

B.1 Record Correspondence and the Balance Identity

Lemma 1 (Record authentication). *A witness satisfying $\mathcal{R}_{\mathsf{tx}}$ fixes complete creation and spend lists. Every creation record authenticates its note opening, value, spending-key derivation, creation position, and output slot. Every spend entry contains a nullifier published at its stated position.*

Proof. Membership fixes T_q and its slots. Output constraints fix each creation opening and $(b, s) = (q, j)$. Input constraints fix its published nullifier and endpoint q. Strictly increasing positions and terminal commitments fix the complete lists. The same records determine each history change x_q.

Lemma 2 (Lifecycle correctness and spend completeness). *Assume exact record correspondence in equation* (5)*, record authentication, and an accepting audit witness. Each lifecycle row describes a distinct selected note and its actual lifetime through Q. Every spend of a selected note through Q is recorded at its actual transaction position.*

Proof. Creation matching authenticates each row; strict nullifier order gives distinct notes. An endpoint at most Q matches the unique published spend. For a row marked $Q + 1$, equality of the derived and checked hash-chain values binds its non-membership check under R_N^Q. These checks establish each note's lifetime.

Theorem 2 (Balance-history identity). *Under the conditions of lemma 2, write $v_i = \mathsf{rec}_i^{\mathsf{L}}.v$, $b_i = \mathsf{rec}_i^{\mathsf{L}}.b$, and $e_i = e_i^{\mathsf{L}}$ for the value, creation position, and endpoint of lifecycle row i. For every $0 \le q \le Q$, equation* (13) *holds.*

Proof. Summing the transaction changes gives the first equality because both committed lists use the records that wrote the history leaves. Each row contributes its value after creation. Its spend cancels that value from position e_i onward. With $b_i < e_i$, its remaining contribution is exactly $v_i \mathbf{1}[b_i \le q < e_i]$, giving the second equality. By lemma 2, the contributing rows represent distinct active notes controlled by the same key. Their set is a subset of $\mathcal{U}_{sk_{\mathsf{u}}}(S_q)$, so summing their positive values gives the inequality.

The gap property stated after equation (13) follows from Lemma 2: a spend in any omitted position between consecutive processed transactions, or in the final tail through Q, would require a processed spend at that position. Omitting a receipt preserves the subset inequality in equation (13).

B.2 Randomized Matching and Composed Soundness

The preceding identity uses the exact relationships in equation (5). The audit enforces their randomized product tests in equation (10). We now bound the chance that these tests accept inconsistent records and combine that bound with the query proof.

Lemma 3 (Fixed-list matching). *For fixed committed lists and independent uniform $\gamma_C, \gamma_D \in \mathbb{F}_p$, false exact correspondence passes both product checks with probability at most $(n_C + n_D)/p$, outside record-hash collisions.*

Proof. Distinct equal-length multisets of field elements define distinct monic product polynomials. Their difference has degree at most the corresponding list length and thus at most that many roots. Apply this to the creation and spend comparisons and take a union bound. Record hashing preserves the compared identities outside its collision event. This reasoning includes a zero factor.

Let q_{FS} bound distinct public tuples Y_Q used in challenge attempts and q_H bound hash queries, including extraction. The random-oracle argument in Section B.4 binds the lists before each challenge except with probability $O(q_H^2/p)$. Let ϵ_{bind} bound key, note, and tree-binding failures, and $\epsilon_{\mathsf{sample}}$ bound challenge-sampling bias. Together with lemma 3, these give

$$\epsilon_{\mathsf{match}} \leq \frac{(q_{\mathsf{FS}} + 1)(n_C^{\max} + n_D^{\max})}{p} + O(q_H^2/p) + \epsilon_{\mathsf{ext}} + \epsilon_{\mathsf{bind}} + \epsilon_{\mathsf{sample}}. \tag{17}$$

The factor $q_{\mathsf{FS}} + 1$ accounts for Fiat–Shamir challenge grinding. The bound assumes auxiliary-input extraction preserving a common oracle transcript, as specified in Section B.3.

Theorem 3 (PP-IPoH knowledge soundness). *Under the stated extraction, binding, random-oracle, and pool assumptions, SPAN satisfies the knowledge-soundness condition of Definition 1. An accepted proof yields a master-key commitment opening w_{ext} satisfying $\mathcal{R}_{\mathsf{hold}}(\mathcal{L}_Q, \mathsf{query}; w_{\mathsf{ext}}) = 1$, except with probability at most $\epsilon_{\mathsf{match}} + \epsilon_{\mathsf{pool}}$, where ϵ_{pool} bounds failure of the pool and finalized-data assumptions.*

Proof. Outside the failure events, joint extraction and shared commitments identify the same lists and history root across the proofs. Theorem 2 establishes the lower bound on actual balances through Q. History-update induction fixes every summary under that root. The authenticated cover summaries identify its minimum on $[t_1, t_2]$, and equation (15) proves it is at least θ. The verifier requires $t_2 \leq Q$.

Completeness. For every sufficient witness in Definition 1, the honest user processes the selected notes in order and builds their true lifecycle table. Exact record matching holds for every challenge, remaining notes pass non-membership, and a supported selection meeting the threshold satisfies the query relation, giving the definition's completeness property. Selecting all controlled notes realizes the full balance. A smaller selection gives its lower bound.

B.3 Joint Knowledge Extraction

Let $\mathcal{I} \subseteq \{\mathsf{tx}, \mathsf{audit}, \min\}$ be the accepted components in an adversarially chosen statement. For each accepted component a, let y_a, π_a, vk_a be its public statement, proof, and verification key. Probabilities include setup, adversarial randomness, and oracle answers. The joint extraction interface required in Section 2.1 provides an extractor E such that

$$\Pr\left[\begin{array}{l}\bigwedge_{a\in\mathcal{I}} \mathsf{Verify}_a(vk_a, y_a, \pi_a) = 1 \\ \wedge\ E \text{ fails to extract all required valid witnesses}\end{array}\right] \le \epsilon_{\mathsf{ext}}. \tag{18}$$

The extractor uses one consistent random-oracle assignment and leaves accepted public statements fixed. It supports adaptively selected statements and auxiliary input. The extracted transaction witness contains the initial master-key commitment opening and complete transition sequence. The audit witness contains Ω, its complete scans, and final nullifier paths. The query witness contains the hidden history root, salt, and authenticated node records. Queries performed by extraction and relation checking are included in q_H. This joint interface supplies compatible witnesses for the composed theorem.

Commitment binding associates the extracted witnesses through C_C, C_D, C_L, C_{hist}. The master-key commitment binds the keys used in transaction steps. The two IVCs use their respective proving keys under fixed parameters and shared public inputs. The query relation compares the shared fields of X_Q and binds the complete request digest.

B.4 Lists Fixed Before the Challenge

The fixed-list matching bound requires the compared records to be determined before the challenge. Commitment binding fixes their contents, while the Fiat–Shamir analysis must also establish this ordering in the adversary's oracle transcript. We therefore trace the accepted list openings back to queries made before the first challenge for their public tuple Y_Q. Model the independent list-hash, record-hash, and challenge domains as random oracles with uniform outputs in $\mathbb{F}_p$, and fix an attempted Y_Q immediately before its first creation- or spend-challenge query. For each of C_C, C_D, C_L, search the preceding list-commit queries with that output and the prescribed public inputs and tag. Such a query supplies a salt, declared length, and final hash-chain value. Follow the corresponding list-step queries backwards, checking their tag and counter, until reaching the fixed initial hash-chain value. A complete chain identifies the entire ordered list.

Consider two bad events. First, two distinct queried encodings have the same hash output. Second, a fresh hash answer equals a previously fixed target used as a commitment output or a missing chain link. Condition on neither event occurring. If a valid extracted opening is completed only after the first challenge for Y_Q, consider its first missing link in the backward traversal. Its output was already fixed in the public commitment or in an earlier link's input. Producing this missing link later requires the latter bad event. Likewise, if the outer list-commit query did not exist before the challenge, its subsequent answer must hit the already chosen public commitment. Across at most q_H relevant queries and targets, the union bound is $O(q_H^2/p)$. A final record-hash collision is charged to the same oracle-collision analysis or to the separately stated binding term.

Thus, outside these events, any successful extracted opening is already represented by the prechallenge query transcript. Its records cannot be chosen as functions of the challenges for Y_Q. The order-sensitive chain authenticates the list and its length. For the product argument, conceptually sample the hashes of these fixed records at this point. They belong to separate domains and are independent of the matching challenges, which have not yet been answered. This handles records whose hash query is delayed until after the challenge without permitting their contents to change.

The extractor of Appendix B.3 supplies the complete openings under the same oracle transcript.

B.5 Multiple Fiat–Shamir Challenge Attempts

Apply Lemma 3 to the lists fixed by Appendix B.4. Empty lists are covered by their count and commitment checks.

For each new Y_Q submitted to the challenge oracle, the preceding subsection fixes its candidate lists. Applying the polynomial bound conditionally on the transcript and taking a union bound over at most q_{FS} distinct tuples gives the grinding term in equation (17). The extra one covers a final proof whose Y_Q is first submitted to that oracle during verification. Either challenge domain may be queried first; their outputs remain independent. All queries for one Y_Q share the same pair of challenge values and constitute one attempt.

Together with extraction, commitment binding, and the events where fresh oracle outputs hit fixed targets, this gives the conditional bound in equation (17). The term $\epsilon_{\mathsf{sample}}$ accounts for accumulated bias in the fixed hash-to-field sampler of Section 4.2.

B.6 PP-IPoH Knowledge-Soundness Experiment

The experiment $\mathsf{KS}_{\mathcal{S},\mathcal{A},E}(1^\lambda)$ instantiates Definition 1 on a ledger recording accepted transactions and their note witnesses. It samples $pp \leftarrow \mathsf{Setup}(1^\lambda)$, runs $\mathcal{A}$ with auxiliary input and ledger and oracle access to obtain (query, Π), and derives the verifier's roots from the finalized log. The joint extractor E acts on this execution and returns $w_{\mathsf{ext}} = (sk_{\mathsf{u}}, r_{\mathsf{u}})$ or $\perp$. The experiment returns the

indicator of the definition's knowledge-soundness event, using $\mathcal{R}_{\mathsf{hold}}(\cdot;\bot) = 0$. Theorem 3 bounds its success probability by $\epsilon_{\mathsf{match}} + \epsilon_{\mathsf{pool}}$.

B.7 PP-IPoH Privacy Experiments

For the first condition in Definition 1, pub contains the three public statements, prescribed terminal states, exposed counts, ledger and request transcript. Honest setup and proofs define the real distribution; the simulated setup and its trapdoor generate the corresponding proof transcripts for the same public statements. The auxiliary-input multi-theorem zero-knowledge interface supports these simulations under adaptive requests, including a fixed common user commitment.

For the second condition, $\mathsf{Priv}^{b}_{\mathcal{S},\mathcal{V}^*}(1^\lambda)$ samples honest setup, then takes two sufficient witness collections and auxiliary input from $\mathcal{V}^*$. They share the ledger, user commitment, configuration and disclosure $\mathsf{Leak}_{\mathrm{SPAN}}$ of Section 2.3. With bit $b \in \{0,1\}$, the challenger prepares honest commitments and proofs from collection b using independent fresh salts. Further adaptive requests must be supported by both collections and have the same disclosed outcome and invocation counts; otherwise both experiments terminate with the same fixed output. Reusing a history reuses its commitments and proofs within that world. Both worlds have the same public request fields and reuse pattern, while their randomized record and history commitments may differ. The view includes every transmitted commitment and proof, and the experiment returns the verifier's final bit. For every efficient $\mathcal{V}^*$ and admissible witnesses, require

$$\left|\Pr[\mathsf{Priv}^{0}_{\mathcal{S},\mathcal{V}^*}(1^\lambda) = 1] - \Pr[\mathsf{Priv}^{1}_{\mathcal{S},\mathcal{V}^*}(1^\lambda) = 1]\right| \le \mathsf{negl}(\lambda).$$

First replace the component proofs by their adaptive multi-theorem zero-knowledge simulations for the same public statements and exposed counts. Then use commitment hiding to replace the messages in the randomized record and history commitments. In the random-oracle model, recognizing a low-entropy record or balance through its commitment requires guessing the independent secret salt or reaching a hash-collision event. The domain-separated challenges are deterministic functions of Y_Q, so their disclosure is determined by the public commitments and shared inputs. The Pedersen commitment is common to the two histories. These hybrids bound the distinguishing advantage by the proof-simulation advantages, hash-commitment hiding terms, and relevant collision probabilities.

The proof interfaces protect the remaining private states and witnesses under the declared disclosure. Repeated queries using Y_Q are linked, and accepted thresholds reveal their inequalities. The transcript model excludes proving time, remote-provider access patterns and local-memory side channels. Obtaining nullifier paths through a local index or a privacy-preserving retrieval channel satisfies its data-access assumption.

Together with the completeness argument in Section B.2 and Theorem 3, these privacy arguments establish Theorem 1 for Algorithm 1.