ra:st

A Non-Custodial Privacy Protocol with Provable Compliance via Association Sets

Technical Whitepaper — Version 0.1

ra:st protocol

2026

Abstract

Public blockchains impose a stark privacy cost: every transfer, balance, and counterparty relation is recorded permanently and legibly for any observer. This transparency, while foundational to the trust model of open ledgers, is incompatible with ordinary financial confidentiality. Existing anonymity systems resolve the tension by hiding all participants indiscriminately, which places them in direct conflict with anti-money-laundering obligations and has led to their prohibition. This document specifies *ra:st*, a non-custodial privacy protocol deployed on Robinhood Chain that severs the on-chain link between the point at which value enters a shielded pool and the point at which it leaves, while requiring every withdrawal to carry a zero-knowledge proof that the exiting funds belong to an *approved association set*. The construction follows the Privacy Pools paradigm of Buterin, Illum, Nadler, Fabian, and Soleimani, and builds directly upon the open-source Privacy Pools reference implementation of 0xbow. We give a complete formal treatment: the algebraic commitment scheme instantiated over the Poseidon hash function on the BN254 scalar field, the incremental Merkle accumulator, the Groth16 argument system used for succinct verification, and the arithmetic circuits enforcing deposit and withdrawal correctness. We define the protocol's security goals—unlinkability, balance soundness, and double-spend resistance—as cryptographic games and sketch reductions to standard hardness assumptions. We conclude with a description of the deployed system architecture, an explicit threat model, and a discussion of the protocol's limitations. The protocol is non-custodial in the strongest sense: no operator holds user funds or keys, no administrative key can freeze or seize a position, and a single client-held secret is the sole authority over any deposited value.

prove everything. reveal nothing.

Contents

Part I — Overview

1 Executive Summary

ra:st is a privacy layer for on-chain money. It lets anyone move value on a public blockchain without broadcasting their financial life to the world, while still being able to prove—to anyone who requires it—that the money is clean. It is deployed on Robinhood Chain, it is non-custodial, and it holds neither your funds nor your keys.

The problem it solves is simple to state. On a public blockchain, everything is visible: how much you hold, who pays you, what you buy, and when. That transparency is permanent and searchable by anyone, forever. For a settlement system meant to carry ordinary economic life—salaries, savings, business revenue—this is an extraordinary and involuntary disclosure. No bank statement works this way. No cash transaction works this way. Yet every on-chain transfer does.

Earlier attempts to fix this hid everyone indiscriminately. By giving identical cover to every participant, they also gave cover to illicit funds, and they gave honest users no way to distinguish themselves. Regulators responded by shutting them down. The lesson was not that privacy is incompatible with compliance; it was that *indiscriminate* privacy is.

ra:st takes a different route. When you deposit, the deposit is public—anyone can see that an address funded the pool. When you withdraw, the withdrawal is private—nobody can tell which deposit it came from. And at the moment you withdraw, you attach a zero-knowledge proof that your funds belong to an approved set: you prove your money is clean without revealing anything else about it. Honest users keep their privacy. Funds that cannot prove their provenance simply do not get the anonymity.

The result is a system with three properties that used to be thought mutually exclusive:

- **Unlinkable.** No on-chain observer can connect your deposit to your withdrawal.
- **Provably clean.** Every withdrawal proves, in zero knowledge, that it belongs to an approved association set. This is not a mixer.
- **Self-custodial.** No operator ever holds your funds or your keys. A single secret—your recovery phrase—is the only thing that can move your money. We cannot freeze it, seize it, or recover it. Neither can anyone else.

The remainder of this document explains, first in plain terms and then in full technical detail, what ra:st is, whom it is for, how it works, and why its guarantees hold.

2 The Problem

2.1 A public ledger is a permanent disclosure

Consider what it means for a ledger to be public. When you receive two ETH from a client, the blockchain now records, forever and for everyone: your address, the client's address, the exact amount, and the precise time. Anyone—a competitor, an employer, an insurer, a stranger who once sold you an NFT, a party not yet born—can read it. They can trace where the money came from and where it goes next. They can assemble, from thousands of such records, a complete and permanent picture of your financial behavior.

This is not an incidental flaw; it is how public blockchains are built. Their integrity depends on every participant being able to verify the whole history, which means the whole history is legible to everyone. Privacy was traded away for verifiability at the foundation.

In traditional finance the default is reversed. A bank statement is private; obtaining someone else's requires legal process. Cash reveals nothing at all. We take this for granted precisely

because it has always been the default. On-chain, the default is the opposite, and most users do not realize how much they disclose until it is used against them.

2.2 Who this hurts, concretely

The transparency problem is not abstract. It has victims:

- **The individual paid on-chain.** Your salary, its source, and everything you do with it become public the moment you are paid. Your net worth is legible in real time to anyone who learns one of your addresses.
- **The business accepting crypto.** Your revenue, your suppliers, your margins, and your treasury movements are visible to competitors as they happen.
- **The trader.** Your positions and your strategy are readable on a public ledger. Your edge disappears the moment it is on-chain; copytrading is not a strategy, it is just reading.
- **The counterparty who never consented.** When someone pays you on-chain, the record ties *their* address to yours, whether or not they wanted that link exposed. On-chain privacy is not selfish; it protects everyone you transact with.

A whole industry of chain-analysis firms exists to exploit this legibility. The question is not whether anyone is looking. On a public chain, everyone is always looking.

2.3 Why not just hide everyone?

The obvious fix—pool everyone's funds together so inputs and outputs cannot be matched—was tried. It works, cryptographically: it provides strong unlinkability. But it hides the honest and the illicit behind the same curtain, indistinguishably. It offers no way for an honest user to show her funds are clean, and equal service to funds that are not. That indiscriminate design is why such systems were sanctioned and shut down.

The failure was specific: not privacy, but privacy *without accountability*. The insight behind ra:st is that zero-knowledge proofs dissolve this trade-off. You can prove a fact about your money—that it comes from an approved source—without revealing the money itself. Privacy and provable compliance stop being opposites.

3 What ra:st Is

ra:st is a non-custodial shielded pool with a compliance proof built in. Using it is three moves, and there is nothing else to learn.

1. Deposit, in the open. You put value into the pool. This transaction is fully public: anyone can see that your address deposited. What they cannot see is the secret behind it. Before depositing, your device generates a secret note and publishes only its commitment—a hash. The secret never leaves your device.

2. Prove, in zero knowledge. To take your funds out, you prove two things at once, revealing neither: that you own a note in the pool, and that your note belongs to an approved association set. The proof is generated locally, in your browser. It convinces the network without disclosing which note is yours.

3. Withdraw, in private. Your funds arrive at a fresh address with no on-chain link to your deposit. The chain shows that a deposit happened and that a withdrawal happened; it cannot show that they were the same person.

That is the entire protocol. Deposit in public, withdraw in private, and prove your funds are clean. No mixer. No custody. The note is the only key.

4 Who It Is For

Individuals who do not want their salary, savings, and every purchase to be a permanent public record searchable by anyone. Privacy here is not secrecy; it is the ordinary financial confidentiality that every other payment system already provides.

Businesses that cannot operate with their revenue, supplier relationships, and treasury movements broadcast in real time to competitors, but that also cannot use tools which would compromise their regulatory standing. ra:st's compliance proof is what makes it usable in this setting.

Traders whose alpha evaporates the moment their positions are legible on a public ledger, and who need to move capital without signaling it to the entire market.

Institutions for whom privacy and compliance are both non-negotiable. The association-set mechanism is precisely the primitive that lets an institution demonstrate clean provenance while retaining confidentiality.

In every case the pitch is the same: the confidentiality that traditional finance takes for granted, brought to on-chain money, without asking anyone to trust an operator and without hiding illicit funds alongside honest ones.

5 Design Philosophy

Three commitments shape every decision in the protocol.

We hold nothing. There is no operator account that custodies your deposit, no administrative key that can freeze or seize a position, and no upgrade button that can change the rules on funds already in the pool. Your recovery phrase is the sole authority over your money. This is a hard guarantee, and it cuts both ways: because we built the system so that we cannot move your funds, we also cannot recover them if you lose your phrase. Self-custody is the whole point, not a bug.

Your secrets never leave your device. Every computation that touches a secret—generating your note, building your proof—happens locally, in your browser. The servers in this system see only public data. Outsourcing proof generation would mean handing over your witness and defeating the entire purpose; we do not do it, even though it would be easier.

Trust as little as possible. We reduce what you must trust to a small body of auditable code and a few well-studied cryptographic assumptions. Where trust in a human process is unavoidable—curating the association set—we confine it to one explicit, externally auditable input and make its influence legible. We do not ask you to trust us. We ask you to trust math.

Part II — Technical Specification

6 Introduction

6.1 The transparency problem

A public, permissionless blockchain is, by design, a globally replicated and append-only record of every state transition it has ever processed. This property is what allows any participant to independently verify the ledger's integrity without trusting a central authority. It is also the source of a privacy problem that has no analogue in traditional finance. When a user transacts on such a ledger, the following facts become permanently and publicly legible: the addresses involved, the exact amounts transferred, the precise ordering and timing of transfers, and—through graph analysis over the entire transaction history—rich behavioral and relational metadata that frequently suffices to deanonymize the pseudonymous addresses entirely.

A bank statement is private by default; access to it requires legal process. A blockchain balance is public by default and will remain readable indefinitely, by present and future adversaries alike, including parties who were not yet capable of analysis at the time the transaction was recorded. The asymmetry is severe. A user who receives a salary on-chain thereby discloses that salary, its source, and its subsequent disposition to the entire world. A merchant who accepts on-chain payment reveals its revenue in real time to competitors. The problem is not hypothetical; a mature industry of chain-analysis firms exists precisely to exploit this legibility.

6.2 The compliance tension

The naive solution—an anonymity pool that mixes funds so that inputs and outputs cannot be correlated—was realized in systems such as fixed-denomination mixers. These constructions provide strong unlinkability by hiding *all* participants behind a common anonymity set. Precisely because they hide all participants indiscriminately, they provide equally strong cover to funds of illicit origin, and they offer no mechanism by which an honest user can distinguish herself from a malicious one. This indiscriminate design has placed such systems in direct conflict with regulatory obligations, and has in several jurisdictions led to their sanction or prohibition.

The central insight of the Privacy Pools framework [1] is that these two goals—privacy for honest users and exclusion of illicit funds—are not fundamentally opposed. Zero-knowledge proofs permit a user to demonstrate a *property* of her funds without revealing the funds themselves. In particular, a user may prove, in zero knowledge, that her withdrawal originates from a deposit belonging to a curated set of addresses deemed acceptable—an *association set*—without revealing *which* deposit. Honest users whose funds trace to an approved set retain full privacy; funds that cannot be so proven are excluded from the anonymity set altogether. Privacy and provable compliance become simultaneously achievable.

6.3 Contributions and scope

ra:st is an instantiation of this paradigm, deployed on Robinhood Chain, built directly upon the open-source Privacy Pools reference implementation of 0xbow. This document is a complete technical specification. Our contributions in this whitepaper are expository and systematizing rather than a claim of novel cryptographic primitives: the underlying scheme is that of [1] and its reference realization. Specifically, we provide:

1. A self-contained formal treatment of the algebraic objects underlying the protocol (Sections 8 and 9): the BN254 scalar field, the Poseidon permutation and the sponge hash it induces, the commitment scheme, and the incremental Merkle accumulator.

2. A precise specification of the Groth16 argument system as instantiated here, including the relation it proves and the trusted-setup assumptions it requires (Section 10).

3. A complete description of the protocol state machine (Section 11): the deposit, association, and withdrawal procedures, the derivation of commitments, nullifiers, and labels, and the accounting invariants they maintain.

4. A specification of the association-set mechanism and its Merkle-inclusion semantics (Section 12).

5. The full arithmetic-circuit constraint systems for deposit and withdrawal (Section 13).

6. A security analysis (Section 14) defining unlinkability, balance soundness, and double-spend resistance as games, with reductions sketched to the underlying assumptions.

7. A description of the deployed architecture, a threat model, and an honest account of limitations (Sections 16 to 18).

6.4 Design principles

Three principles govern the protocol's design and recur throughout this document.

Non-custody. No party other than the user ever controls user funds. There is no operator account that holds deposits in escrow, no administrative key capable of freezing or seizing a position, and no upgrade mechanism that can alter the rules governing already-deposited value. The sole authority over a deposited note is a secret held by the depositor. This is a strong guarantee: it implies that the protocol operator *cannot* comply with a demand to freeze specific user funds, because the capability to do so does not exist in the system.

Client-side proving. All secret-dependent computation—note generation, witness construction, and zero-knowledge proof generation—occurs on the user's own device. No secret ever crosses the network boundary. The server-side components of the system observe only public data.

Minimal trust surface. The protocol reduces the set of things a user must trust to the correctness of a small, auditable body of code and the soundness of well-studied cryptographic assumptions. Where trust in a human process is unavoidable—specifically, in the curation of the association set—the protocol confines that trust to a single, explicit, and externally auditable input, and makes its influence on the system legible.

6.5 Document conventions

Throughout, $\lambda \in \mathbb{N}$ denotes the security parameter. A function $\mu : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is *negligible*, written $\mu(\lambda) = \mathsf{negl}(\lambda)$, if for every $c \in \mathbb{N}$ there exists λ_0 such that $\mu(\lambda) < \lambda^{-c}$ for all $\lambda > \lambda_0$. We write $x \xleftarrow{\$} S$ for sampling x uniformly from a finite set S. Probabilistic polynomial-time is abbreviated PPT. All adversaries are PPT unless stated otherwise. We reserve p for the base field characteristic of the BN254 curve and r for its scalar-field (group) order; the arithmetic circuits and all in-circuit hashing are defined over $\mathbb{F}_r$, and we write $\mathbb{F}_r := \mathbb{F}_r$ for this working field. The constant

$$r = 21888242871839275222246405745257275088548364400416034343698204186575808495617$$

is the BN254 scalar field order and is referred to throughout as the SNARK scalar field modulus.

6.6 How to read this document

This whitepaper is organized in three parts, addressed to overlapping but distinct audiences. Part I (Overview) is non-technical: it explains what ra:st is, the problem it solves, whom it serves, and the principles behind it, and it is self-contained for a reader who wants the product without the mathematics. Part II (Technical Specification) is the formal core: it develops the cryptographic primitives, the protocol, the circuits, and the security analysis, and it presumes familiarity with basic cryptography. Part III (Ecosystem) returns to the practical: the user journey, the economic model, the roadmap, operational guidance, and honest risk disclosure. A reader may take Part I alone for understanding, Part I and III for a product-and-plan view, or all three for the complete treatment. The appendices collect reference material—notation, deployment specifics, detailed algorithms, and a glossary—for implementers and auditors.

We have tried, throughout, to be precise about what is proven and what is assumed, and about what the protocol guarantees versus what remains the user's or operator's responsibility. Where a guarantee is cryptographic we state the assumption it rests on; where a property is operational we say so plainly rather than dressing it as a theorem. The intent is a document that is honest about its own limits, because a privacy system that overstates its guarantees is worse than one that states them exactly.

7 Background and Related Work

This section situates ra:st in the lineage of on-chain privacy constructions, expanding the brief comparison of Section 13.7 into a fuller account of what came before and how each line of work informs the present design.

7.1 The privacy problem in three generations

First generation: address unlinkability by mixing. The earliest on-chain privacy tools broke the deposit-withdrawal link by pooling many equal-value deposits and letting each depositor later withdraw to a fresh address, with a zero-knowledge proof of membership standing in for a spend authorization. These constructions established the core template that ra:st inherits—commitment on deposit, nullifier on withdrawal, Merkle membership in zero knowledge—and demonstrated that strong unlinkability was achievable on a public chain at practical cost. Their limitation was structural: by admitting every deposit unconditionally, they provided identical cover to lawful and unlawful funds, and offered honest users no means of distinguishing themselves. This is the specific property that drew regulatory sanction.

Second generation: shielded chains and account-based privacy. A parallel line embedded privacy at the base layer, maintaining an entire shielded state in which values, senders, and receivers are hidden by default and every transaction carries a zero-knowledge proof of its own validity. These systems achieve comprehensive privacy and pioneered much of the note-and-nullifier machinery in production. They too, however, lack an intrinsic mechanism by which a user can prove that her funds originate from an approved set: privacy is total and undifferentiated, which reproduces the compliance tension at the level of an entire chain rather than a single pool.

Third generation: privacy with provable compliance. The Privacy Pools framework [1] introduced the decisive idea that a withdrawal can carry, alongside its membership proof, a proof of membership in a curated *association set*—so that honest users prove clean provenance in zero knowledge while illicit funds are excluded from the anonymity set. This dissolves the tension that defeated the first two generations. ra:st is an instantiation of this third-generation design, built on the reference implementation of the framework and deployed on Robinhood Chain.

7.2 The cryptographic toolbox

The construction stands on a small number of now-standard cryptographic advances: pairing-based succinct arguments, which made on-chain verification of complex statements economical; arithmetic-friendly hashing, which made those statements small enough to prove on a laptop; and incremental Merkle accumulators, which made membership provable against a compact, cheaply-updatable commitment to a growing set. Each is used here as a mature building block rather than a research contribution, and the protocol's novelty— to the extent it claims any beyond the framework it instantiates—is in their integration and deployment, not their invention.

7.3 Regulatory context

The regulatory trajectory of on-chain privacy is the backdrop against which the association-set design should be read. Indiscriminate mixing tools have faced sanction in multiple jurisdictions on the ground that they materially facilitate the laundering of illicit proceeds by providing undifferentiated anonymity. The association-set mechanism is a direct technical response: it preserves privacy for participants who can prove membership in an approved set while denying anonymity to those who cannot, converting compliance from an external policy bolted onto a private system into an intrinsic precondition of every private withdrawal. We are careful to claim only what is true: the mechanism makes provable compliance *possible*; whether a given deployment *achieves* regulatory compliance depends on the curation policy it adopts and on the law of the relevant jurisdiction, both of which lie outside the protocol and are the responsibility of the operator. The protocol supplies the primitive; it does not, and cannot, supply a legal conclusion.

8 Preliminaries

We fix notation and recall the standard cryptographic objects on which the protocol depends. A reader familiar with pairing-based zero-knowledge and algebraic hashing may skim this section and Section 9, returning as needed.

8.1 Finite fields and the BN254 curve

Definition 8.1 (Finite field)**.** For a prime power q, $\mathbb{F}_q$ denotes the finite field with q elements. When $q = p$ is prime, $\mathbb{F}_p \cong \mathbb{Z}/p\mathbb{Z}$ with arithmetic modulo p. The multiplicative group $\mathbb{F}_q^\times = \mathbb{F}_q \setminus \{0\}$ is cyclic of order $q - 1$.

The protocol is instantiated over the Barreto–Naehrig curve BN254 (also called BN128 or `alt_bn128`), the pairing-friendly elliptic curve for which efficient precompiled operations are available in the Ethereum execution environment. BN254 is defined by

$$E : y^2 = x^3 + 3 \quad \text{over } \mathbb{F}_p,$$

where p is a 254-bit prime. The group $E(\mathbb{F}_p)$ has prime order r, the scalar field modulus fixed above. The curve admits a non-degenerate bilinear pairing

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T,$$

where $\mathbb{G}_1, \mathbb{G}_2$ are order-r subgroups of $E(\mathbb{F}_p)$ and $E(\mathbb{F}_{p^{12}})$ respectively and $\mathbb{G}_T \subseteq \mathbb{F}_{p^{12}}^\times$. Bilinearity means $e(a \cdot P, b \cdot Q) = e(P, Q)^{ab}$ for all scalars $a, b \in \mathbb{F}_r$ and generators $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$. The pairing is the algebraic engine of the Groth16 verifier (Section 10).

Remark 8.2. All "field elements" manipulated inside the protocol's circuits—commitments, nullifiers, labels, Merkle nodes, secrets—are elements of $\mathbb{F}_r$, i.e. integers reduced modulo r. Values that originate outside the field, such as the Keccak-256 digests used to derive labels, are reduced modulo r before entering any circuit; this reduction is made explicit wherever it occurs.

8.2 Hash functions and random oracles

Definition 8.3 (Collision resistance)**.** A family of functions $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\ell$ is *collision resistant* if for every PPT adversary $\mathcal{A}$,

$$\Pr\Big[(x, x') \leftarrow \mathcal{A}(1^\lambda) : x \neq x' \wedge \mathsf{H}(x) = \mathsf{H}(x')\Big] = \mathsf{negl}(\lambda).$$

For hashing over the field we require, in addition, a notion suited to algebraic inputs. An *arithmetic hash* is a function $\mathsf{H} : \mathbb{F}_r^k \to \mathbb{F}_r$ for a fixed arity k. We treat the Poseidon family (Section 9.1) both as collision resistant in the above sense and, in the analysis of Section 14, as a random oracle: a function sampled uniformly from all functions of its signature, queryable by all parties including the adversary. The random-oracle idealization is standard in the analysis of hash-based commitment and nullifier constructions and we adopt it explicitly as an assumption rather than a theorem.

8.3 Commitment schemes

Definition 8.4 (Commitment scheme)**.** A (non-interactive) commitment scheme over message space $\mathcal{M}$ and randomness space $\mathcal{R}$ is a function $\mathsf{Com} : \mathcal{M} \times \mathcal{R} \to \mathcal{C}$ satisfying:

- **Hiding.** For all $m_0, m_1 \in \mathcal{M}$, the distributions $\{\mathsf{Com}(m_0; \rho) : \rho \xleftarrow{\$} \mathcal{R}\}$ and $\{\mathsf{Com}(m_1; \rho) : \rho \xleftarrow{\$} \mathcal{R}\}$ are computationally indistinguishable.
- **Binding.** For every PPT $\mathcal{A}$,

$$\Pr\Big[(m, \rho, m', \rho') \leftarrow \mathcal{A}(1^\lambda) : (m, \rho) \neq (m', \rho') \wedge \mathsf{Com}(m; \rho) = \mathsf{Com}(m'; \rho')\Big] = \mathsf{negl}(\lambda).$$

The protocol instantiates commitments through Poseidon hashing, treating a subset of the hashed inputs as the committed message and the remainder as the hiding randomness. The precise construction is given in Section 9.2; its hiding property reduces to the random-oracle idealization of Poseidon and its binding property to Poseidon's collision resistance.

8.4 Zero-knowledge arguments of knowledge

Let $\mathcal{R}$ be a polynomial-time-decidable binary relation, and let $\mathcal{L}_\mathcal{R} = \{x : \exists\, w,\ (x, w) \in \mathcal{R}\}$ be its associated language. We call x the *statement* (or public input) and w the *witness*.

Definition 8.5 (NARK)**.** A non-interactive argument of knowledge for $\mathcal{R}$ is a triple of PPT algorithms $(\mathrm{Setup}, \mathrm{Prove}, \mathrm{Verify})$:

- $\mathrm{Setup}(1^\lambda, \mathcal{R}) \to \mathsf{crs}$ produces a common reference string;
- $\mathrm{Prove}(\mathsf{crs}, x, w) \to \pi$ produces a proof;
- $\mathrm{Verify}(\mathsf{crs}, x, \pi) \to \{0,1\}$ accepts or rejects.

It must satisfy:

- **Completeness.** For all $(x, w) \in \mathcal{R}$, $\mathrm{Verify}(\mathsf{crs}, x, \mathrm{Prove}(\mathsf{crs}, x, w)) = 1$.
- **Knowledge soundness.** For every PPT prover $\mathcal{A}$ there exists a PPT extractor $\mathcal{E}$ such that, whenever $\mathcal{A}$ produces an accepting proof π for a statement x, $\mathcal{E}$ (given $\mathcal{A}$'s randomness) outputs w with $(x, w) \in \mathcal{R}$, except with negligible probability.
- **Zero knowledge.** There exists a PPT simulator $\mathcal{S}$ that, given only $x \in \mathcal{L}_\mathcal{R}$ and a trapdoor for crs, produces proofs computationally indistinguishable from those of the honest prover.

The protocol uses the Groth16 argument system, a pairing-based NARK with a constant-size proof (three group elements) and constant-time verification, at the cost of a relation-specific trusted setup. Section 10 makes this precise.

8.5 Merkle accumulators

Definition 8.6 (Merkle tree)**.** Let $\mathsf{H} : \mathbb{F}_r^2 \to \mathbb{F}_r$ be an arithmetic hash. A Merkle tree over a sequence of leaves $(\ell_0, \ldots, \ell_{n-1}) \in \mathbb{F}_r^n$ is the balanced binary tree whose leaves are the ℓ_i and whose every internal node is the hash of its two children. Its *root* is the value at the apex. A *membership path* (or Merkle proof) for leaf ℓ_i is the sequence of sibling nodes along the path from ℓ_i to the root, together with the index i that fixes left/right orientation at each level.

Given a leaf ℓ, an index i, and a path σ, one recomputes a candidate root by iterated hashing; the leaf is a member of a tree with root R iff this recomputation yields R. Collision resistance of H implies that producing two distinct leaves with valid paths to the same root is infeasible; this is the security property on which the protocol's inclusion proofs rest. The specific accumulator used—an incremental, append-only variant with a compact update rule—is defined in Section 9.4.

9 Cryptographic Primitives

This section specifies the concrete primitives from which the protocol is built: the Poseidon arithmetic hash, the commitment and nullifier constructions it induces, and the incremental Merkle accumulator (lean-IMT) used to maintain protocol state.

9.1 The Poseidon hash function

Poseidon is a family of arithmetic hash functions designed for efficiency inside zero-knowledge circuits. Unlike bit-oriented hashes such as SHA-256 or Keccak, whose Boolean operations translate into a large number of arithmetic constraints, Poseidon operates natively on field elements and is expressible in a small number of multiplication gates. This efficiency is decisive: every commitment, nullifier, and Merkle node in the protocol is a Poseidon output, and the withdrawal circuit evaluates Poseidon dozens of times.

9.1.1 The permutation

Poseidon is built from a fixed permutation π_{P} acting on the state space $\mathbb{F}_r^t$, where t is the *width*. The permutation is a sequence of rounds, each of the form

$$\mathsf{state} \mapsto \mathsf{MDS} \cdot \big(S(\mathsf{state} + \mathsf{RC}_i)\big),$$

where $\mathsf{RC}_i \in \mathbb{F}_r^t$ is a round-constant vector, $\mathsf{MDS} \in \mathbb{F}_r^{t \times t}$ is a fixed maximum-distance-separable matrix providing linear diffusion, and S is the non-linear layer. The S-box is the low-degree power map

$$S_{\mathsf{full}}(x_1, \ldots, x_t) = (x_1^\alpha, \ldots, x_t^\alpha), \qquad S_{\mathsf{partial}}(x_1, \ldots, x_t) = (x_1^\alpha, x_2, \ldots, x_t),$$

with $\alpha = 5$ for the BN254 scalar field (the smallest exponent coprime to $r - 1$). Poseidon interleaves R_F *full* rounds (applying the S-box to every state element) with R_P *partial* rounds (applying it to a single element). The full rounds provide resistance to statistical attacks; the partial rounds provide algebraic degree at low constraint cost. The counts (R_F, R_P) and the constants $(\mathsf{RC}_i, \mathsf{MDS})$ are fixed per width t by the Poseidon parameter generation procedure and are not security-sensitive secrets.

9.1.2 The sponge and fixed-arity hashing

A hash on k field elements is obtained by absorbing the inputs into the permutation's state and squeezing one element out. For the fixed, small arities used by the protocol, this reduces to a

single permutation call on a state of width $t = k + 1$: the capacity element is initialized to a domain-separating constant, the k input elements occupy the rate positions, the permutation is applied once, and the first state element is output. We write

$$\mathsf{Poseidon}_k : \mathbb{F}_r^k \to \mathbb{F}_r$$

for the resulting fixed-arity hash. The protocol uses exactly three arities: $\mathsf{Poseidon}_1$, $\mathsf{Poseidon}_2$, and $\mathsf{Poseidon}_3$, corresponding to permutation widths $t = 2, 3, 4$. In the reference implementation these are exposed as the contracts `PoseidonT2`, `PoseidonT3`, and `PoseidonT4` respectively, whose on-chain evaluation is bit-for-bit identical to the in-circuit and client-side evaluations—a compatibility property we rely on and which is verified empirically against the deployed contracts.

Assumption 9.1 (Poseidon security)**.** For each arity $k \in \{1, 2, 3\}$ used by the protocol, $\mathsf{Poseidon}_k$ is collision resistant, and, where required by the analysis of Section 14, behaves as a random oracle over $\mathbb{F}_r$.

9.2 The commitment and nullifier scheme

The protocol's fundamental object is a *note*: a hidden record of value that a user controls. A note is defined by four field elements,

$$\mathsf{note} = (\mathsf{value}, \mathsf{label}, \mathsf{nullifier}, \mathsf{secret}) \in \mathbb{F}_r^4,$$

where value is the denominated value (in the pool's base unit, after fees), label is a public label binding the note to a particular deposit event (Section 11.2), and $(\mathsf{nullifier}, \mathsf{secret}) \in \mathbb{F}_r^2$ is the secret key material known only to the note's owner. From these, three derived quantities are defined.

Definition 9.2 (Precommitment)**.** The *precommitment* of a note binds its secret key material:

$$\mathsf{precommitment} := \mathsf{Poseidon}_2(\mathsf{nullifier}, \mathsf{secret}).$$

Definition 9.3 (Commitment)**.** The *commitment* of a note binds its value, label, and precommitment:

$$\mathsf{cm} := \mathsf{Poseidon}_3(\mathsf{value}, \mathsf{label}, \mathsf{precommitment}) = \mathsf{Poseidon}_3\big(\mathsf{value}, \mathsf{label}, \mathsf{Poseidon}_2(\mathsf{nullifier}, \mathsf{secret})\big).$$

Definition 9.4 (Nullifier hash)**.** The *nullifier hash* of a note is the single-arity hash of its nullifier:

$$\mathsf{nf} := \mathsf{Poseidon}_1(\mathsf{nullifier}).$$

The design separates concerns cleanly. The commitment cm is the public object inserted into the shielded pool at deposit time; it hides $(\mathsf{nullifier}, \mathsf{secret})$ behind two layers of Poseidon and hides nothing an observer should not already learn (value and label are public consequences of a deposit). The nullifier hash nf is the public object revealed at withdrawal time to prevent double-spending: it is a deterministic, one-way function of the secret nullifier, so that spending the same note twice would reveal the same nf twice and be detected, while nf itself leaks nothing about which commitment it corresponds to.

Proposition 9.5 (Binding)**.** *Under the collision resistance of* $\mathsf{Poseidon}_2$ *and* $\mathsf{Poseidon}_3$ *(Assumption 9.1), the commitment map* $(\mathsf{value}, \mathsf{label}, \mathsf{nullifier}, \mathsf{secret}) \mapsto \mathsf{cm}$ *is binding: no PPT adversary can produce two distinct note tuples with the same commitment except with negligible probability.*

Proof sketch. A collision in the commitment map is either a collision in the outer $\mathsf{Poseidon}_3$ on distinct $(\mathsf{value}, \mathsf{label}, \mathsf{precommitment})$ triples, or an agreement on that triple with distinct $(\mathsf{nullifier}, \mathsf{secret})$, which forces a collision in the inner $\mathsf{Poseidon}_2$. Either event contradicts Assumption 9.1. A standard hybrid over the two layers gives the bound. □

Proposition 9.6 (Hiding)**.** *Modeling* $\mathsf{Poseidon}_2$ *as a random oracle (Assumption 9.1) and treating* $(\mathsf{nullifier}, \mathsf{secret})$ *as sampled with sufficient min-entropy, the commitment* cm *reveals no information about* $(\mathsf{nullifier}, \mathsf{secret})$ *beyond what is implied by* value *and* label*: for any two secret pairs, the induced commitment distributions are indistinguishable to a PPT adversary that does not query the oracle at the underlying preimage.*

Proof sketch. $\mathsf{precommitment} = \mathsf{Poseidon}_2(\mathsf{nullifier}, \mathsf{secret})$ is, under the random-oracle model, uniform and independent of all other quantities until the adversary queries the oracle at exactly $(\mathsf{nullifier}, \mathsf{secret})$. With min-entropy $\omega(\log \lambda)$ in the secret pair, the probability of such a query is negligible. Conditioned on its absence, $\mathsf{precommitment}$ is a fresh uniform field element and $\mathsf{cm} = \mathsf{Poseidon}_3(\mathsf{value}, \mathsf{label}, \mathsf{precommitment})$ is distributed identically for any secret pair. □

9.3 Deterministic note derivation

To spare users from managing independent random secrets for every note, the protocol derives all of a user's note secrets deterministically from a single master seed—a standard BIP-39 mnemonic. This yields a wallet-like recovery model: one backed-up phrase suffices to reconstruct every note the user has ever created.

Construction 9.7 (Master-key derivation)**.** From a mnemonic m, derive two master field elements

$$\mathsf{msk}_{\mathsf{nullifier}} := \mathsf{Poseidon}_1(\kappa_1), \qquad \mathsf{msk}_{\mathsf{secret}} := \mathsf{Poseidon}_1(\kappa_2),$$

where κ_1, κ_2 are two field-reduced sub-keys obtained from m via the standard hierarchical-deterministic derivation. The pair $(\mathsf{msk}_{\mathsf{nullifier}}, \mathsf{msk}_{\mathsf{secret}})$ constitutes the master keys.

Construction 9.8 (Per-deposit secret derivation)**.** For a pool of scope scope (Section 11.2) and a monotonically increasing deposit index $i \in \mathbb{N}$, the note secrets are

$$\mathsf{nullifier}_i := \mathsf{Poseidon}_3(\mathsf{msk}_{\mathsf{nullifier}}, \mathsf{scope}, i), \qquad \mathsf{secret}_i := \mathsf{Poseidon}_3(\mathsf{msk}_{\mathsf{secret}}, \mathsf{scope}, i).$$

The derivation binds each note to both the pool (scope) and a per-pool counter (i), so that distinct deposits yield distinct, unlinkable secrets, while the entire family is recoverable from the seed alone. The index i is not secret; it is recovered at scan time by trial derivation (Section 11.5). Because the master keys never leave the client and the derivation is one-way, exposure of any individual note's secrets does not compromise the seed or any other note.

Remark 9.9. The determinism of Construction 9.8 is a usability feature, not a security requirement of the withdrawal argument: the circuit of Section 13 accepts *any* consistent $(\mathsf{value}, \mathsf{label}, \mathsf{nullifier}, \mathsf{secret})$ tuple and does not assume the secrets were seed-derived. A user may equally supply independently sampled secrets. The deterministic construction is the recommended default because it collapses backup to a single phrase.

9.4 The incremental Merkle accumulator (lean-IMT)

The shielded pool maintains the set of all deposited commitments as an append-only Merkle accumulator. The protocol uses the *lean incremental Merkle tree* (lean-IMT), an optimization of the classic fixed-depth incremental Merkle tree that avoids precomputed zero-subtrees and grows its depth dynamically with the number of leaves.

9.4.1 Structure and update rule

A lean-IMT over hash $\mathsf{H} = \mathsf{Poseidon}_2$ maintains a list of leaves and a current root. Unlike a fixed-depth tree, which pads to a constant number of levels using canonical "zero" subtree hashes, the lean-IMT stores only the nodes actually induced by present leaves and defines the root as the hash-folding of the current frontier. Insertion of a new leaf updates only the $O(\log n)$ nodes along its path. Formally, writing n for the number of leaves and $d = \lceil \log_2 n \rceil$ for the current depth:

- **Insertion.** Appending leaf ℓ_n recomputes the path from position n to the root, hashing ℓ_n upward against its existing left-siblings; where a right-sibling is absent, the node is promoted unchanged. The depth increases by one exactly when n crosses a power of two.
- **Root.** The root is the apex value after folding the frontier; for a single leaf ($n = 1$) the root equals the leaf itself, reflecting a depth-zero tree.
- **Membership.** A proof for leaf ℓ_i consists of its index i, the realized sibling list σ along its path, and the current depth d. The verifier recomputes the root by folding ℓ_i against σ according to the bits of i and checks equality against the claimed root.

Definition 9.10 (lean-IMT inclusion relation)**.** For a claimed root R, leaf ℓ, index i, depth d, and sibling list $\sigma = (\sigma_0, \ldots, \sigma_{d-1})$, define the recomputation

$$h_0 := \ell, \qquad h_{j+1} := \begin{cases} \mathsf{Poseidon}_2(h_j, \sigma_j) & \text{if bit } j \text{ of } i \text{ is } 0, \\ \mathsf{Poseidon}_2(\sigma_j, h_j) & \text{if bit } j \text{ of } i \text{ is } 1, \end{cases} \quad j = 0, \ldots, d-1,$$

and say (ℓ, i, σ, d) is a valid inclusion witness for root R iff $h_d = R$.

The circuit implements exactly Definition 9.10; the same recomputation is performed on-chain to validate that a submitted proof refers to a known state root. To bound circuit size, the sibling list is padded with zero elements to a fixed maximum depth $D_{\max}$, and the effective depth d is supplied as a public signal so that the circuit folds only the first d levels.

Proposition 9.11 (Accumulator soundness)**.** *Under the collision resistance of* $\mathsf{Poseidon}_2$*, no PPT adversary can produce, for a root R committed to by an honest insertion sequence, a valid inclusion witness for a leaf ℓ that was never inserted, except with negligible probability.*

Proof sketch. A valid witness for a non-member leaf yields, along the first level at which the recomputed path diverges from the honest tree, two distinct $\mathsf{Poseidon}_2$ preimages mapping to the same node value—a collision. The reduction guesses the divergence level (a $1/D_{\max}$ loss) and outputs the colliding pair. □

9.4.2 Two accumulators

The protocol maintains *two* lean-IMTs, with different leaf semantics:

- The **state tree** accumulates *commitments*: its leaves are the cm values of all deposits. Membership in the state tree is what a withdrawal proves about the note being spent.
- The **association-set tree** accumulates *labels*: its leaves are the label values of deposits deemed acceptable by the association-set provider (Section 12). Membership in this tree is what a withdrawal proves about the note's compliance.

The distinction in leaf type—commitment versus label—is essential and is enforced by the circuit (Section 13): the withdrawal argument proves state-tree membership of the *commitment* and association-tree membership of the *label*, simultaneously.

10 Succinct Arguments: Groth16

The withdrawal procedure requires a user to convince the pool contract of a compound statement—correct note structure, membership in two accumulators, and correct accounting—while revealing none of the underlying secret witness. The protocol realizes this with the Groth16 zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK), whose constant-size proofs and constant-time pairing-based verification are well suited to on-chain use.

10.1 From circuits to constraint systems

A zk-SNARK proves satisfiability of an arithmetic constraint system. The protocol's statements are expressed as arithmetic circuits over $\mathbb{F}_r$; a compiler translates each circuit into a *rank-1 constraint system* (R1CS).

Definition 10.1 (R1CS)**.** An R1CS over $\mathbb{F}_r$ with m constraints and n variables is a triple of matrices $(A, B, C) \in \mathbb{F}_r^{m \times n}$. A vector $z \in \mathbb{F}_r^n$, whose first entry is the constant 1 and whose remaining entries are partitioned into public inputs and private witness, *satisfies* the system iff

$$(Az) \circ (Bz) = (Cz),$$

where $\circ$ denotes the entrywise (Hadamard) product. Each row encodes one bilinear constraint $\langle a_i, z\rangle \cdot \langle b_i, z\rangle = \langle c_i, z\rangle$.

Every arithmetic circuit compiles to such a system: each multiplication gate becomes a row, additions and constant scalings are folded into the linear combinations, and each Poseidon evaluation expands into the rows encoding its round function. The R1CS is in turn arithmetized as a *quadratic arithmetic program* (QAP) by interpolating the constraint matrices into polynomials over an evaluation domain; satisfiability becomes a polynomial divisibility check, which is what the pairing ultimately verifies.

10.2 The Groth16 argument

Construction 10.2 (Groth16, abridged)**.** For a relation $\mathcal{R}$ with public input x and witness w:

- $\mathsf{Setup}(1^\lambda, \mathcal{R})$ samples a secret trapdoor $\tau = (\alpha, \beta, \gamma, \delta, s)$, publishes a structured reference string of $\mathbb{G}_1/\mathbb{G}_2$ elements encoding the QAP polynomials evaluated at s and masked by the trapdoor scalars, and discards τ.
- $\mathsf{Prove}(\mathsf{crs}, x, w)$ outputs $\pi = (A, B, C) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1$, a randomized encoding of a satisfying assignment.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi)$ accepts iff the single pairing equation

$$e(A, B) = e(\alpha_{\mathbb{G}_1}, \beta_{\mathbb{G}_2}) \cdot e\Big(\textstyle\sum_j x_j \cdot \mathsf{IC}_j,\ \gamma_{\mathbb{G}_2}\Big) \cdot e(C, \delta_{\mathbb{G}_2})$$

  holds, where the $\mathsf{IC}_j \in \mathbb{G}_1$ are the public-input commitment bases from crs.

The proof is three group elements, independent of circuit size; verification is three pairings plus a multi-scalar multiplication over the public inputs, independent of witness size. These succinctness properties are what make on-chain verification economical.

Theorem 10.3 (Groth16 security, informal)**.** *Under the generic-group model (equivalently, suitable knowledge-of-exponent and q-type assumptions on BN254), the construction of Construction 10.2 is a non-interactive zero-knowledge argument of knowledge in the sense of Definition 8.5: it is complete, knowledge-sound, and zero-knowledge.*

We use Theorem 10.3 as a black box. Its knowledge soundness is what lets the security analysis of Section 14 *extract* a valid note witness from any adversary who produces an accepting withdrawal proof; its zero-knowledge is what guarantees the proof leaks nothing about which note was spent.

10.3 The trusted setup

Groth16 requires a relation-specific structured reference string whose generation involves the secret trapdoor τ. If τ is known to an adversary, that adversary can forge proofs for false statements; soundness therefore rests on τ having been destroyed. The protocol's reference string is produced by a multi-party computation ("powers-of-tau" followed by a circuit-specific phase-two ceremony) in which each participant contributes randomness and the trapdoor remains unknown so long as at least one participant behaved honestly and discarded their contribution. The resulting proving and verifying keys—a proving key of roughly 18 MB and a compact verifying key—are fixed public parameters. The verifying key is embedded in the on-chain verifier contract; the proving key is served to clients for local proof generation.

Assumption 10.4 (Setup integrity)**.** The phase-two ceremony for the withdrawal relation had at least one honest participant, so that the trapdoor τ is unknown to any PPT adversary.

Assumption 10.4 is the one setup-dependent trust assumption in the system. It is a one-time, publicly auditable event, and its "1-of-N honest" threshold is the weakest such assumption available for a ceremony of N participants.

10.4 Poseidon in full

We give the concrete structure of the permutation used by the protocol, at the level of detail needed to reproduce it. Fix a width $t \in \{2, 3, 4\}$ and the S-box exponent $\alpha = 5$. The permutation $\pi_{\mathsf{P}} : \mathbb{F}_r^t \to \mathbb{F}_r^t$ is a composition of R_F full rounds and R_P partial rounds, with the full rounds split evenly before and after the partial block:

$$\pi_{\mathsf{P}} = \underbrace{\mathcal{F} \circ \cdots \circ \mathcal{F}}_{R_F/2} \circ \underbrace{\mathcal{P} \circ \cdots \circ \mathcal{P}}_{R_P} \circ \underbrace{\mathcal{F} \circ \cdots \circ \mathcal{F}}_{R_F/2}.$$

A full round $\mathcal{F}$ and a partial round $\mathcal{P}$ act on a state $x \in \mathbb{F}_r^t$ as

$$\mathcal{F}(x) = M \cdot \big(\rho_{\mathsf{full}}(x + c)\big), \qquad \mathcal{P}(x) = M \cdot \big(\rho_{\mathsf{part}}(x + c)\big),$$

where $c \in \mathbb{F}_r^t$ is the round-specific constant vector, $M \in \mathbb{F}_r^{t \times t}$ is the MDS matrix, and the non-linear layers are

$$\rho_{\mathsf{full}}(x_1, \ldots, x_t) = (x_1^5, x_2^5, \ldots, x_t^5), \qquad \rho_{\mathsf{part}}(x_1, \ldots, x_t) = (x_1^5, x_2, \ldots, x_t).$$

The MDS matrix. M is a fixed maximum-distance-separable matrix over $\mathbb{F}_r$, chosen so that the linear layer achieves full diffusion in a single round: every output coordinate depends on every input coordinate, and the branch number is maximal. A standard choice is a Cauchy matrix $M_{ij} = (u_i + v_j)^{-1}$ for fixed distinct $\{u_i\}, \{v_j\}$ with $u_i + v_j \neq 0$; MDS-ness follows from the non-vanishing of every square submatrix determinant, a property of Cauchy matrices.

Round counts. The counts (R_F, R_P) are chosen to defeat statistical and algebraic cryptanalysis with a security margin. Full rounds dominate resistance to differential and linear attacks; partial rounds inflate the polynomial degree of the permutation—each raising it by a factor of α—at the cost of a single S-box rather than t. For the BN254 field and the widths used here, the reference parameters provide 128-bit security. The concrete constants (c, M, R_F, R_P) are produced by the Poseidon parameter-generation script from the field modulus and width, are fully public, and are not security-sensitive.

From permutation to hash. For a fixed-arity hash $\mathsf{Poseidon}_k$, the state of width $t = k+1$ is initialized as $(\iota, m_1, \ldots, m_k)$, where ι is a domain-separation constant fixed by the arity and the m_j are the inputs; the permutation is applied once; and the first output coordinate is returned:

$$\mathsf{Poseidon}_k(m_1, \ldots, m_k) = \big[\pi_{\mathsf{P}}(\iota, m_1, \ldots, m_k)\big]_1.$$

Because each S-box is a degree-5 map and the linear layers are free in the constraint system, the number of R1CS constraints contributed by one $\mathsf{Poseidon}_k$ evaluation is proportional to $t \cdot R_F + R_P$ multiplication gates for the S-boxes, plus the fixed cost of the constant additions, which is the quantity that dominates the circuit sizes of Section 13.

10.5 From R1CS to QAP

We expand the arithmetization sketched in Section 10, since it is what the pairing check ultimately certifies. Let the R1CS be $(A, B, C) \in \mathbb{F}_r^{m \times n}$ with satisfying assignment $z \in \mathbb{F}_r^n$. Choose m distinct interpolation points $\{r_1, \ldots, r_m\} \subseteq \mathbb{F}_r$ and define the vanishing polynomial

$$Z(X) = \prod_{i=1}^{m} (X - r_i).$$

For each variable $j \in \{1, \ldots, n\}$, define polynomials $A_j, B_j, C_j \in \mathbb{F}_r[X]$ of degree $< m$ by Lagrange interpolation through the j-th columns of the constraint matrices:

$$A_j(r_i) = A_{ij}, \qquad B_j(r_i) = B_{ij}, \qquad C_j(r_i) = C_{ij}, \qquad i = 1, \ldots, m.$$

Set

$$A(X) = \sum_{j=1}^{n} z_j A_j(X), \quad B(X) = \sum_{j=1}^{n} z_j B_j(X), \quad C(X) = \sum_{j=1}^{n} z_j C_j(X).$$

Proposition 10.5 (QAP satisfiability)**.** *The assignment z satisfies the R1CS (A, B, C) if and only if the polynomial $A(X)\,B(X) - C(X)$ is divisible by $Z(X)$; equivalently, there exists a quotient $H(X) \in \mathbb{F}_r[X]$ with*

$$A(X)\,B(X) - C(X) = H(X)\,Z(X).$$

Proof. Evaluate at each interpolation point r_i. By construction $A(r_i) = \langle a_i, z\rangle$, $B(r_i) = \langle b_i, z\rangle$, and $C(r_i) = \langle c_i, z\rangle$, where a_i, b_i, c_i are the i-th rows. The R1CS is satisfied iff $\langle a_i, z\rangle\langle b_i, z\rangle = \langle c_i, z\rangle$ for every i, i.e. iff $A(r_i)B(r_i) - C(r_i) = 0$ for all i, i.e. iff every r_i is a root of $AB - C$. A univariate polynomial vanishes at all r_i iff it is divisible by $Z(X) = \prod_i (X - r_i)$. The two conditions coincide. □

The Groth16 prover commits, in the exponent of the trusted-setup encoding, to the polynomials A, B, C, H evaluated at the secret point s; the verifier's pairing equation (Construction 10.2) checks exactly the divisibility identity of Proposition 10.5 at s, which by the Schwartz–Zippel lemma holds for a random s only with negligible error unless the identity holds as polynomials. This is the sense in which a single pairing certifies satisfaction of all m constraints at once.

10.6 Public-signal encoding

The withdrawal verifier consumes a fixed, ordered vector of public signals. The protocol fixes the following order, which the on-chain proof library and the circuit agree upon exactly:

$$\begin{aligned}
\mathsf{pub}[0] &= \mathsf{cm}^{\mathsf{new}} && \text{(new commitment hash)}\\
\mathsf{pub}[1] &= \mathsf{nf} && \text{(existing nullifier hash)}\\
\mathsf{pub}[2] &= \mathsf{value}^{\mathsf{wd}} && \text{(withdrawn value)}\\
\mathsf{pub}[3] &= R_{\mathsf{state}} && \text{(state root)}\\
\mathsf{pub}[4] &= d_{\mathsf{state}} && \text{(state tree depth)}\\
\mathsf{pub}[5] &= R_{\mathsf{ASP}} && \text{(association-set root)}\\
\mathsf{pub}[6] &= d_{\mathsf{ASP}} && \text{(association tree depth)}\\
\mathsf{pub}[7] &= \mathsf{ctx} && \text{(context).}
\end{aligned}$$

The semantics of each signal are defined in Sections 11 and 13. The context signal ctx (Section 11.4) binds the proof to the specific withdrawal parameters, preventing a valid proof from being replayed against different recipient or fee data.

11 Protocol Design

We now specify the protocol as a state machine: the objects it maintains, the deposit and withdrawal procedures, and the invariants they preserve. The presentation is agnostic to the underlying asset; the deployed instantiation is a native-currency (ETH) pool, and the extension to fungible tokens is noted where relevant.

11.1 System objects and roles

Definition 11.1 (Protocol state)**.** The on-chain state of a pool comprises:

- the **state tree**, a lean-IMT of commitments with current root R_{state} and a bounded history of recent roots;
- the **nullifier set** $\mathcal{N} \subseteq \mathbb{F}_r$, the set of nullifier hashes already spent;
- the **association-set registry**, an append-only list of published association roots with the latest R_{ASP};
- static configuration: the pool scope, the minimum deposit amount minDep, and the vetting-fee rate $\phi \in [0, 1)$ in basis points.

Three roles interact with the state. A **depositor** adds commitments. A **spender** (the owner of a note, possibly acting through a relayer) removes value by proving and nullifying. An **association-set provider** (ASP) publishes association roots. Crucially, none of these roles is a custodian: the contract holds pooled value, but no role can move a specific user's value without that user's note secrets.

11.2 Scope and label

Definition 11.2 (Scope)**.** The *scope* of a pool is a fixed field element derived from the pool's deployment identity (its address and chain), serving as a domain separator so that notes and contexts are bound to a specific pool:

$$\mathsf{scope} \in \mathbb{F}_r.$$

Definition 11.3 (Label)**.** At deposit time the pool assigns the deposit a *label*, derived from the scope and a monotonically increasing per-pool nonce η:

$$\mathsf{label} := \mathsf{keccak256}(\mathsf{scope}, \eta) \bmod r.$$

The label is public and is emitted in the deposit event. It serves two purposes. First, it individuates deposits within a pool, so that the association set can reference specific deposits by label. Second, it enters the commitment (Definition 9.3), binding the note to its deposit event. Because η increases with every deposit, labels are unique within a pool. The reduction modulo r maps the 256-bit Keccak digest into the scalar field so that the label may appear as a circuit signal.

Remark 11.4. The label is assigned by the contract, not chosen by the depositor, and is not known to the client until the deposit transaction is mined. Consequently, the client's pre-deposit computation commits only to the precommitment; the full commitment is reconstructed after the deposit by reading label from the emitted event. This ordering is reflected in the deposit procedure below.

11.3 The deposit procedure

Depositing is a fully public operation. Its privacy contribution is prospective: the deposit enlarges the anonymity set into which a later, unlinkable withdrawal will blend.

Construction 11.5 (Deposit)**.** To deposit value $v \geq \mathsf{minDep}$:

1. **(Client)** Derive note secrets $(\mathsf{nullifier}, \mathsf{secret})$ for the next index i via Construction 9.8, and compute the precommitment $\mathsf{precommitment} = \mathsf{Poseidon}_2(\mathsf{nullifier}, \mathsf{secret})$.
2. **(Client)** Submit the transaction $\mathsf{deposit}(\mathsf{precommitment})$ carrying value v to the pool's entrypoint.
3. **(Contract)** Charge the vetting fee: the net value entering the pool is

   $$\mathsf{value} := v \cdot (1 - \phi),$$

   computed as $\mathsf{value} = v \cdot (10^4 - \phi_{\mathsf{bps}})/10^4$ in basis points.
4. **(Contract)** Assign the label $\mathsf{label} = \mathsf{keccak256}(\mathsf{scope}, \eta) \bmod r$ and increment η.
5. **(Contract)** Form the commitment $\mathsf{cm} = \mathsf{Poseidon}_3(\mathsf{value}, \mathsf{label}, \mathsf{precommitment})$ and insert it as a new leaf in the state tree, updating R_{state}.
6. **(Contract)** Emit the event $\mathsf{Deposited}(\mathsf{depositor}, \mathsf{cm}, \mathsf{label}, \mathsf{value}, \mathsf{precommitment})$.
7. **(Client)** On observing the event, record the completed note $(\mathsf{value}, \mathsf{label}, \mathsf{nullifier}, \mathsf{secret})$ and increment the local index counter.

Remark 11.6 (Fund-safety invariant)**.** The commitment inserted by the contract is $\mathsf{Poseidon}_3(\mathsf{value}, \mathsf{label}, \mathsf{precon}$ using the *contract-charged* net value value and the *contract-assigned* label label. For the deposited value to be later withdrawable, the client's reconstructed note must use exactly these quantities. A note reconstructed with an incorrect value or label would yield a commitment with no preimage the client can prove, and the funds would be permanently unspendable. The protocol therefore treats the correctness of the commitment derivation—in particular, the bit-for-bit agreement between the client, circuit, and on-chain Poseidon evaluations—as a fund-safety invariant, verified against the deployed hash contracts rather than assumed.

11.4 The context binding

A withdrawal specifies where value is to be sent and, if relayed, what fee is paid to the relayer. These parameters are not secret, but they must be bound to the proof so that an adversary who observes a valid proof in the mempool cannot re-target it to a different recipient. The binding is the *context*.

Definition 11.7 (Context)**.** For a withdrawal descriptor $W = (\mathsf{processor}, \mathsf{data})$—where $\mathsf{processor}$ is the address authorized to submit the withdrawal and data encodes recipient and fee parameters—the context is

$$\mathsf{ctx} := \mathsf{keccak256}\big(\mathsf{encode}(W), \mathsf{scope}\big) \bmod r.$$

The context is a public signal of the withdrawal proof. The on-chain verifier recomputes ctx from the submitted W and the pool scope and rejects unless it matches the proof's context signal. Because the circuit binds the proof to ctx, and ctx commits to W, a valid proof is usable only with the exact withdrawal parameters under which it was generated: altering the recipient changes W, hence ctx, and invalidates the proof.

11.5 The withdrawal procedure

Withdrawal is where privacy is produced. The spender proves, in zero knowledge, that she owns a note whose commitment is in the state tree and whose label is in the association set, that she has correctly computed its nullifier hash, and that value is conserved; she reveals only the nullifier hash, the withdrawn amount, the roots, and the context.

Construction 11.8 (Withdrawal)**.** To withdraw value $v_{\mathsf{wd}} \le \mathsf{value}$ from a note $(\mathsf{value}, \mathsf{label}, \mathsf{nullifier}, \mathsf{secret})$ with commitment cm:

1. **(State proof)** Reconstruct the state tree from the pool's deposit events, locate cm at index j_{state}, and extract its inclusion witness $(\sigma_{\mathsf{state}}, d_{\mathsf{state}})$ for the current root R_{state}.

2. **(Association proof)** Obtain the current association-set tree (published by the ASP), locate label at index j_{ASP}, and extract its inclusion witness $(\sigma_{\mathsf{ASP}}, d_{\mathsf{ASP}})$ for the latest root R_{ASP}.

3. **(Fresh secrets)** Derive fresh withdrawal secrets $(\mathsf{nullifier}', \mathsf{secret}')$ for the change note via a derivation analogous to Construction 9.8, and form the new commitment

   $$\mathsf{cm}^{\mathsf{new}} := \mathsf{Poseidon}_3\big(\mathsf{value} - v_{\mathsf{wd}},\ \mathsf{label},\ \mathsf{Poseidon}_2(\mathsf{nullifier}', \mathsf{secret}')\big),$$

   representing the residual value retained in the pool (zero in a full withdrawal).

4. **(Context)** Fix the withdrawal descriptor W and compute ctx per Definition 11.7.

5. **(Prove)** Generate a Groth16 proof π for the withdrawal relation (Section 13) on public signals $(\mathsf{cm}^{\mathsf{new}}, \mathsf{nf}, v_{\mathsf{wd}}, R_{\mathsf{state}}, d_{\mathsf{state}}, R_{\mathsf{ASP}}, d_{\mathsf{ASP}}, \mathsf{ctx})$ and private witness $(\mathsf{value}, \mathsf{label}, \mathsf{nullifier}, \mathsf{secret}, \mathsf{nullifier}', \mathsf{secret}', \sigma$

6. **(Submit)** The authorized processor submits (W, π, pub) to the pool.

7. **(Contract)** The pool verifies: (a) the caller equals $W.\mathsf{processor}$; (b) the recomputed context matches $\mathsf{pub}[7]$; (c) $R_{\mathsf{state}} = \mathsf{pub}[3]$ is a known state root and $R_{\mathsf{ASP}} = \mathsf{pub}[5]$ is the latest association root; (d) the tree depths are within bounds; (e) the nullifier hash $\mathsf{nf} = \mathsf{pub}[1]$ is not in $\mathcal{N}$; and (f) the Groth16 proof verifies.

8. **(Contract)** On success: mark nf spent ($\mathcal{N} \leftarrow \mathcal{N} \cup \{\mathsf{nf}\}$); insert $\mathsf{cm}^{\mathsf{new}}$ into the state tree; transfer v_{wd} to the recipient encoded in W; and emit $\mathsf{Withdrawn}(\mathsf{processor}, v_{\mathsf{wd}}, \mathsf{nf}, \mathsf{cm}^{\mathsf{new}})$.

11.5.1 Note discovery

Step 1 of Construction 11.8 presumes the spender can locate her note. Because secrets are seed-derived (Construction 9.8), discovery is a local trial procedure: for each index $i = 0, 1, 2, \ldots$, the client derives $(\mathsf{nullifier}_i, \mathsf{secret}_i)$, computes the candidate precommitment, and checks whether it appears among the pool's deposit events. Matches are the user's notes; their spent status is determined by testing whether $\mathsf{Poseidon}_1(\mathsf{nullifier}_i)$ lies in the on-chain nullifier set. A gap limit terminates the scan after a run of consecutive misses.

11.5.2 Double-spend prevention

The nullifier hash $\mathsf{nf} = \mathsf{Poseidon}_1(\mathsf{nullifier})$ is a deterministic function of the note's secret nullifier. Spending a note reveals nf and the contract records it. A second attempt to spend the same note necessarily reveals the same nf—the nullifier is fixed by the note—and is rejected by the membership check in step 7(e). No two distinct notes share a nullifier except with negligible probability (collision resistance of $\mathsf{Poseidon}_1$), so the mechanism neither permits double-spends nor blocks legitimate distinct notes.

11.6 Partial withdrawals and value conservation

The withdrawal relation enforces the accounting identity

$$\mathsf{value} = v_{\mathsf{wd}} + v_{\mathsf{residual}}, \qquad v_{\mathsf{residual}} \geq 0,$$

where v_{residual} is the value of the freshly created change note $\mathsf{cm}^{\mathsf{new}}$. A full withdrawal sets $v_{\mathsf{residual}} = 0$; a partial withdrawal retains the remainder in a new, unlinkable note under fresh secrets. The circuit enforces non-negativity via a range check on v_{wd} and on v_{residual}, preventing the creation of value through underflow.

12 The Association Set Provider

The feature distinguishing this construction from an indiscriminate mixer is the requirement that every withdrawal prove membership of its note in an *association set*: a curated collection of deposits deemed acceptable. This section specifies the association mechanism, its trust model, and the semantics of the compliance proof.

12.1 Motivation

A mixer hides all participants behind a common anonymity set and thereby provides uniform cover to honest and illicit funds alike. This uniformity is both its strength (strong unlinkability) and its fatal weakness (no honest user can distinguish herself, and the system offers equal service to illicit value). The association-set mechanism breaks the uniformity in a privacy-preserving way: it partitions the potential anonymity set into an *approved* subset and its complement, and requires each withdrawal to prove membership in the approved subset—without revealing which member it is.

Definition 12.1 (Association set)**.** An *association set* is a subset $\mathcal{S} \subseteq \mathbb{F}_r$ of deposit labels. It is represented as a lean-IMT whose leaves are the labels in $\mathcal{S}$, with root R_{ASP}. Publication of R_{ASP} (together with an off-chain pointer, e.g. an IPFS content identifier, to the full leaf set) constitutes an assertion by the ASP that every deposit whose label lies in $\mathcal{S}$ is approved.

12.2 Publication and on-chain registry

The ASP is the sole role authorized to update the association root, gated by an on-chain access-control role (`ASP_POSTMAN`). Publication appends a record

$$(R_{\mathsf{ASP}},\ \mathsf{cid},\ \mathsf{timestamp})$$

to an append-only on-chain registry, where cid is an off-chain pointer to the leaf set (subject to a length validity check) and the root becomes the new *latest* association root. Withdrawals are validated against the latest root only. The append-only registry preserves the full history of published sets, so that the provenance of any accepted withdrawal is externally auditable after the fact.

12.3 The compliance proof

The withdrawal circuit (Section 13) enforces association-set membership by an inclusion proof of the note's *label* against R_{ASP}, exactly as in Definition 9.10, with the constraint

$$R_{\mathsf{ASP}} \stackrel{!}{=} \mathrm{root}\big(\mathsf{label},\ j_{\mathsf{ASP}},\ \sigma_{\mathsf{ASP}},\ d_{\mathsf{ASP}}\big).$$

Because the leaf is the label—a public per-deposit identifier—and not the commitment or any secret, the association proof reveals nothing about the note's value or ownership; it establishes only that the note's deposit is among the approved set. The separation of leaf types between the two trees (commitments in the state tree, labels in the association tree) is what allows a single proof to simultaneously demonstrate "this note exists in the pool" and "this note's deposit is approved" without cross-leaking.

12.4 Trust model and policy neutrality

The ASP is the protocol's one point of human-curated trust, and we are explicit about its scope and its limits.

What the ASP can do. By choosing which labels to include in $\mathcal{S}$, the ASP determines which deposits are withdrawable with privacy. Excluding a label prevents the corresponding note from being spent through the shielded path (it does not seize the funds; see below).

What the ASP cannot do. The ASP holds no custody. It cannot move, freeze, or seize any user's funds: withdrawal still requires the note secrets, which the ASP never learns. It cannot forge membership for a label it did not include, nor retroactively alter a published set (the registry is append-only). It cannot deanonymize a withdrawal: the compliance proof is zero-knowledge with respect to which approved member is spending.

Curation policy is out of protocol scope. The *criterion* by which the ASP admits or excludes labels—whether it screens against sanctions lists, applies a provenance heuristic, or admits all deposits unconditionally—is a policy decision external to the cryptographic protocol. The protocol fixes only the *mechanism* (membership must be proven) and is neutral as to the *policy* (what is admitted). Different deployments may adopt different policies; the security properties of Section 14 hold regardless. We stress that a deployment's choice of curation policy carries legal and regulatory significance that is beyond the scope of this document and is the responsibility of the deploying operator.

Remark 12.2 (Non-seizure under exclusion)**.** Excluding a label does not destroy or capture the associated value. The funds remain in the pool under the exclusive control of the note secret. If

a subsequently published association set includes the label, the note becomes spendable again. The ASP's influence is thus confined to the *privacy* of a withdrawal, never to the *ownership* of funds.

13 Circuit Specification

This section gives the constraint systems of the two arithmetic circuits: the deposit (commitment) circuit and the withdrawal circuit. Constraints are stated over $\mathbb{F}_r$; we write $a \overset{!}{=} b$ for an enforced equality (a rank-1 constraint) and describe range checks and hash sub-circuits at the level of their input/output relations.

13.1 Signal conventions

Each circuit distinguishes *public* signals (the statement x, visible to the verifier) from *private* signals (the witness w, known only to the prover). All Poseidon evaluations $\mathsf{Poseidon}_k$ expand in-circuit into the round constraints of Section 9.1; we treat them as constraint-generating gadgets with the stated input/output relation. Merkle inclusion is the gadget of Definition 9.10, folding to a fixed maximum depth $D_{\max}$ with an active-depth signal.

13.2 The commitment circuit

The commitment circuit certifies that a claimed commitment is correctly formed from its note components. It is used both as a standalone check and as a sub-circuit of the withdrawal circuit.

Construction 13.1 (Commitment circuit)**.** *Public:* cm, nf. *Private:* value, label, $\mathsf{nullifier}$, secret.
Constraints:

$$\mathsf{precommitment} \overset{!}{=} \mathsf{Poseidon}_2(\mathsf{nullifier}, \mathsf{secret}), \tag{1}$$

$$\mathsf{cm} \overset{!}{=} \mathsf{Poseidon}_3(\mathsf{value}, \mathsf{label}, \mathsf{precommitment}), \tag{2}$$

$$\mathsf{nf} \overset{!}{=} \mathsf{Poseidon}_1(\mathsf{nullifier}). \tag{3}$$

Constraint (1) binds the secret material into the precommitment; (2) binds value and label into the commitment; (3) exposes the nullifier hash. A satisfying witness demonstrates knowledge of a full note whose commitment is cm and whose nullifier hash is nf, without revealing $(\mathsf{nullifier}, \mathsf{secret})$.

13.3 The withdrawal circuit

The withdrawal circuit is the protocol's core statement. It proves, in zero knowledge, ownership of an existing note that is present in the state tree and approved in the association set, correct nullification, and value conservation into a fresh change note.

Construction 13.2 (Withdrawal circuit)**.** *Public signals* (in the order of Section 10.6): $\mathsf{cm}^{\mathsf{new}}$, nf, v_{wd}, R_{state}, d_{state}, R_{ASP}, d_{ASP}, ctx.
Private signals: value, label, $\mathsf{nullifier}$, secret (existing note); $\mathsf{nullifier}'$, secret' (fresh note); σ_{state}, j_{state} (state path); σ_{ASP}, j_{ASP} (association path).

Constraints:

$$\text{(existing commitment)} \quad \mathsf{cm} \overset{!}{=} \mathsf{Poseidon}_3\big(\mathsf{value}, \mathsf{label}, \mathsf{Poseidon}_2(\mathsf{nullifier}, \mathsf{secret})\big), \tag{4}$$

$$\text{(nullifier)} \quad \mathsf{nf} \overset{!}{=} \mathsf{Poseidon}_1(\mathsf{nullifier}), \tag{5}$$

$$\text{(state inclusion)} \quad R_{\mathsf{state}} \overset{!}{=} \mathrm{root}(\mathsf{cm}, j_{\mathsf{state}}, \sigma_{\mathsf{state}}, d_{\mathsf{state}}), \tag{6}$$

$$\text{(association inclusion)} \quad R_{\mathsf{ASP}} \overset{!}{=} \mathrm{root}(\mathsf{label}, j_{\mathsf{ASP}}, \sigma_{\mathsf{ASP}}, d_{\mathsf{ASP}}), \tag{7}$$

$$\text{(value range)} \quad 0 \le v_{\mathsf{wd}} \le \mathsf{value} < 2^L, \tag{8}$$

$$\text{(fresh commitment)} \quad \mathsf{cm}^{\mathsf{new}} \overset{!}{=} \mathsf{Poseidon}_3\big(\mathsf{value} - v_{\mathsf{wd}}, \mathsf{label}, \mathsf{Poseidon}_2(\mathsf{nullifier}', \mathsf{secret}')\big), \tag{9}$$

$$\text{(context binding)} \quad \mathsf{ctx} \text{ is a declared public input, bound into the proof.} \tag{10}$$

We comment on each constraint.

- (4) reconstructs the existing commitment from the private note, establishing that the prover knows a well-formed note.
- (5) exposes the nullifier hash; combined with the on-chain spent-set check, this is the double-spend guard.
- (6) proves the existing commitment is a leaf of the state tree at the claimed root. The leaf is the *commitment*.
- (7) proves the note's label is a leaf of the association tree at the latest root. The leaf is the *label*. This is the compliance statement.
- (8) bounds both the withdrawn amount and the note value to L bits (with L chosen so that $2^L < r$), ruling out modular wraparound that could manufacture value; it also enforces $v_{\mathsf{wd}} \le \mathsf{value}$.
- (9) constructs the change note with the residual value $\mathsf{value} - v_{\mathsf{wd}}$ under fresh secrets, conserving value.
- (10) binds the proof to the withdrawal descriptor via the context signal, defeating mempool re-targeting (Section 11.4).

Proposition 13.3 (Circuit soundness of accounting)**.** *Any witness satisfying Construction 13.2 conserves value: the sum of the withdrawn amount* v_{wd} *and the residual value encoded in* $\mathsf{cm}^{\mathsf{new}}$ *equals the value* value *of the spent note, and neither is negative or exceeds field range. Consequently no satisfying witness creates net value.*

Proof sketch. By (9) the residual value is exactly $\mathsf{value} - v_{\mathsf{wd}}$, and by (8) both v_{wd} and value lie in $[0, 2^L)$ with $v_{\mathsf{wd}} \le \mathsf{value}$, so the residual lies in $[0, \mathsf{value}]$ and the identity $\mathsf{value} = v_{\mathsf{wd}} + (\mathsf{value} - v_{\mathsf{wd}})$ holds over the integers (no modular wraparound, as $2^L < r$). Value is therefore conserved. □

13.4 Constraint-count considerations

The dominant cost in both circuits is Poseidon evaluation and Merkle folding. Each $\mathsf{Poseidon}_k$ contributes a fixed number of constraints determined by its round counts; each Merkle level contributes one $\mathsf{Poseidon}_2$ plus selector logic; the range check contributes L Boolean constraints plus a summation. Because the folding depth is padded to $D_{\max}$, the withdrawal circuit's size is dominated by $2D_{\max}$ Poseidon evaluations (two inclusion proofs), independent of the actual tree occupancy. This fixed size is what permits a single trusted setup to serve pools of any population up to $2^{D_{\max}}$ deposits.

13.5 Algorithms

For completeness we give the client-side procedures in pseudocode. These are the operations a conforming client performs; all secret-dependent steps run locally.

Input: mnemonic m, pool scope scope, next index i, amount v
Output: deposit transaction; recorded note
$(\mathsf{msk}_{\mathsf{nullifier}}, \mathsf{msk}_{\mathsf{secret}}) \leftarrow \textsc{MasterKeys}(m)$
$\mathsf{nullifier} \leftarrow \mathsf{Poseidon}_3(\mathsf{msk}_{\mathsf{nullifier}}, \mathsf{scope}, i)$
$\mathsf{secret} \leftarrow \mathsf{Poseidon}_3(\mathsf{msk}_{\mathsf{secret}}, \mathsf{scope}, i)$
$\mathsf{precommitment} \leftarrow \mathsf{Poseidon}_2(\mathsf{nullifier}, \mathsf{secret})$
submit $\mathsf{deposit}(\mathsf{precommitment})$ with value v
`// on observing` $\mathsf{Deposited}(\cdot, \mathsf{cm}, \mathsf{label}, \mathsf{value}, \mathsf{precommitment})$:
record note $(\mathsf{value}, \mathsf{label}, \mathsf{nullifier}, \mathsf{secret})$; $i \leftarrow i + 1$

Algorithm 1: Deposit (client side)

Input: mnemonic m, scope scope, deposit events $\mathcal{D}$, gap limit g
Output: list of owned notes with spent status
$(\mathsf{msk}_{\mathsf{nullifier}}, \mathsf{msk}_{\mathsf{secret}}) \leftarrow \textsc{MasterKeys}(m)$; $notes \leftarrow []$; $miss \leftarrow 0$
for $i \leftarrow 0, 1, 2, \ldots$ ***while*** $miss < g$ **do**
  $\mathsf{nullifier} \leftarrow \mathsf{Poseidon}_3(\mathsf{msk}_{\mathsf{nullifier}}, \mathsf{scope}, i)$; $\mathsf{secret} \leftarrow \mathsf{Poseidon}_3(\mathsf{msk}_{\mathsf{secret}}, \mathsf{scope}, i)$
  $\mathsf{precommitment} \leftarrow \mathsf{Poseidon}_2(\mathsf{nullifier}, \mathsf{secret})$
  if $\exists\, (\mathsf{cm}, \mathsf{label}, \mathsf{value}, \mathsf{precommitment}) \in \mathcal{D}$ **then**
    $\mathsf{nf} \leftarrow \mathsf{Poseidon}_1(\mathsf{nullifier})$; $spent \leftarrow \textsc{NullifierUsed}(\mathsf{nf})$
    append $(\mathsf{value}, \mathsf{label}, \mathsf{nullifier}, \mathsf{secret}, i, spent)$ to $notes$; $miss \leftarrow 0$
  else
    $miss \leftarrow miss + 1$
  end
end
return *notes*

Algorithm 2: Note discovery (client side)

Input: note $(\mathsf{value}, \mathsf{label}, \mathsf{nullifier}, \mathsf{secret})$, amount v_{wd}, descriptor W, mnemonic m
Output: withdrawal proof and public signals
$\mathsf{cm} \leftarrow \mathsf{Poseidon}_3(\mathsf{value}, \mathsf{label}, \mathsf{Poseidon}_2(\mathsf{nullifier}, \mathsf{secret}))$
build state tree from deposit commitments; $(\sigma_{\mathsf{state}}, j_{\mathsf{state}}, d_{\mathsf{state}}) \leftarrow \textsc{Path}(\mathsf{cm})$
fetch association leaves; $(\sigma_{\mathsf{ASP}}, j_{\mathsf{ASP}}, d_{\mathsf{ASP}}) \leftarrow \textsc{Path}(\mathsf{label})$
$(\mathsf{nullifier}', \mathsf{secret}') \leftarrow \textsc{WithdrawSecrets}(m, \mathsf{label}, \cdot)$
$\mathsf{cm}^{\mathsf{new}} \leftarrow \mathsf{Poseidon}_3(\mathsf{value} - v_{\mathsf{wd}}, \mathsf{label}, \mathsf{Poseidon}_2(\mathsf{nullifier}', \mathsf{secret}'))$
$\mathsf{nf} \leftarrow \mathsf{Poseidon}_1(\mathsf{nullifier})$; $\mathsf{ctx} \leftarrow \mathsf{keccak256}(\mathsf{encode}(W), \mathsf{scope}) \bmod r$
$\mathsf{pub} \leftarrow (\mathsf{cm}^{\mathsf{new}}, \mathsf{nf}, v_{\mathsf{wd}}, R_{\mathsf{state}}, d_{\mathsf{state}}, R_{\mathsf{ASP}}, d_{\mathsf{ASP}}, \mathsf{ctx})$
$\pi \leftarrow \textsc{Groth16.Prove}(\mathsf{pub},\ \mathrm{witness})$
return (π, pub)

Algorithm 3: Withdraw (client side)

13.6 A worked example

To make the objects concrete, we trace a single note through the protocol with illustrative small values (the true values are field elements of full size; here we use short numbers purely to show the data flow).

Suppose a user deposits and, after the vetting fee, the net note value is $\mathsf{value} = 9950000000000000$ (i.e. 0.00995 in base units at 18 decimals, from a 0.01 deposit at a 50 basis-point fee). The contract assigns a label label from the scope and nonce. The user's device has derived, for index

$i = 0$, a nullifier $\mathsf{nullifier}$ and secret secret; from these it forms

$$\mathsf{precommitment} = \mathsf{Poseidon}_2(\mathsf{nullifier}, \mathsf{secret}), \qquad \mathsf{cm} = \mathsf{Poseidon}_3(\mathsf{value}, \mathsf{label}, \mathsf{precommitment}).$$

The contract inserts cm as the first leaf of the state tree. With a single leaf, the state root equals the leaf: $R_{\mathsf{state}} = \mathsf{cm}$, at depth 0.

For the association set, the provider builds a tree whose leaves are the approved labels—say three prior labels and this user's label, giving four leaves, depth 2. The user's label sits at index 3; its inclusion witness is the two siblings along its path, and the provider publishes the root R_{ASP}.

To withdraw the full amount, the user sets $v_{\mathsf{wd}} = \mathsf{value}$, so the change note has value 0; she derives fresh $(\mathsf{nullifier}', \mathsf{secret}')$ and forms $\mathsf{cm}^{\mathsf{new}} = \mathsf{Poseidon}_3(0, \mathsf{label}, \mathsf{Poseidon}_2(\mathsf{nullifier}', \mathsf{secret}'))$. She computes $\mathsf{nf} = \mathsf{Poseidon}_1(\mathsf{nullifier})$ and the context from her chosen recipient. The circuit checks: that cm reconstructs from her note; that nf is its nullifier hash; that cm is in the state tree at R_{state}; that label is in the association tree at R_{ASP}; that $0 \le v_{\mathsf{wd}} \le \mathsf{value}$; and that $\mathsf{cm}^{\mathsf{new}}$ carries the residual $\mathsf{value} - v_{\mathsf{wd}} = 0$. The proof verifies, the contract records nf as spent, inserts $\mathsf{cm}^{\mathsf{new}}$, and releases value to the recipient. The chain now shows a deposit and a withdrawal with no link between them; the spent nullifier ensures the note cannot be used again.

13.7 Comparison to prior systems

We situate the design relative to three prior approaches, along the axes of privacy, compliance, and custody.

System	Unlinkable	Provable compliance	Non-custodial
Fixed-denomination mixer	yes	no	yes
Shielded-chain (note-based)	yes	no	yes
Custodial privacy service	partial	operator-dependent	no
ra:st (this work)	yes	yes	yes

The mixer achieves unlinkability and non-custody but hides all participants indiscriminately, offering no way to prove clean provenance—the property that led to sanction. A shielded chain provides strong note-based privacy but likewise lacks a built-in mechanism to prove membership in an approved set. A custodial service can gesture at compliance but does so by holding user funds and trust, forfeiting non-custody and re-introducing exactly the counterparty risk that on-chain systems exist to remove. ra:st's contribution, inherited from the Privacy Pools framework, is to occupy the remaining cell: unlinkable *and* provably compliant *and* non-custodial, by replacing indiscriminate hiding with a zero-knowledge proof of association-set membership.

14 Security Analysis

We define the protocol's three principal security goals as cryptographic games and sketch reductions of each to the assumptions already introduced: collision resistance and the random-oracle behavior of Poseidon (Assumption 9.1), the knowledge soundness and zero-knowledge of Groth16 (Theorem 10.3), and setup integrity (Assumption 10.4). The goals are *unlinkability*, *balance soundness*, and *double-spend resistance*. Throughout, the adversary controls the network, may act as depositor and spender, and observes all public state; it does not control the ASP's inclusion decisions except as noted.

14.1 Unlinkability

Unlinkability is the core privacy guarantee: an observer cannot correlate a withdrawal with the deposit that funded it, beyond the unavoidable leakage of public amounts and the anonymity-set size.

Definition 14.1 (Withdrawal unlinkability)**.** Consider the game between challenger $\mathcal{C}$ and PPT adversary $\mathcal{A}$:

1. $\mathcal{C}$ honestly creates two notes N_0, N_1 of equal value and equal label-approval status, inserting their commitments into the state tree and their labels into the association set, interleaved with adversarially chosen deposits.

2. $\mathcal{C}$ samples $b \xleftarrow{\$} \{0,1\}$ and performs a full withdrawal of note N_b under a fixed context, publishing the resulting proof and public signals.

3. $\mathcal{A}$, given the entire public transcript, outputs a guess b'.

The protocol is *withdrawal-unlinkable* if $\left|\Pr[b' = b] - \frac{1}{2}\right| = \mathsf{negl}(\lambda)$.

Theorem 14.2 (Unlinkability)**.** *Under the zero-knowledge property of Groth16 (Theorem 10.3) and the random-oracle idealization of Poseidon (Assumption 9.1), the protocol is withdrawal-unlinkable in the sense of Definition 14.1.*

Proof sketch. We bound $\mathcal{A}$'s advantage by a hybrid argument. In the real game the published proof π is generated from note N_b. Replace π by a simulated proof produced by the Groth16 simulator on the public signals alone; by zero-knowledge this change is indistinguishable. The remaining public signals are the nullifier hash $\mathsf{nf}_b = \mathsf{Poseidon}_1(\mathsf{nullifier}_b)$, the withdrawn value, the roots, the context, and the fresh change commitment $\mathsf{cm}^{\mathsf{new}}$. Under the random-oracle model, nf_b is a uniform field element independent of which note was spent (the two notes' nullifiers are independent random-oracle preimages of equal-status notes), and $\mathsf{cm}^{\mathsf{new}}$ is a commitment under fresh secrets, hence uniform and independent of b by Proposition 9.6. The withdrawn value, roots, and context are identical for $b = 0$ and $b = 1$ by the equal-value, fixed-context setup. Thus, after the hybrid, the transcript is statistically independent of b, and $\mathcal{A}$'s advantage is negligible. □

Remark 14.3 (Scope of the guarantee)**.** Theorem 14.2 isolates the cryptographic contribution to unlinkability. It does *not* claim protection against side channels outside the protocol: correlation by transaction timing, by amount when amounts are distinctive, by network-level metadata (IP linkage of deposit and withdrawal), or by an anonymity set of size one. The practical anonymity a user obtains is bounded by the number of indistinguishable deposits present in the pool at withdrawal time; a withdrawal from a pool containing a single deposit is trivially linkable regardless of the proof's zero-knowledge. These operational considerations are discussed in Section 18.

14.2 Balance soundness

Balance soundness is the guarantee that the pool cannot be drained beyond what was deposited: no adversary can withdraw value it did not deposit, nor manufacture value through malformed proofs.

Definition 14.4 (Balance soundness)**.** For any PPT adversary $\mathcal{A}$ interacting with the pool through the deposit and withdrawal interfaces, let D be the total net value deposited and W the total value successfully withdrawn over the interaction. The protocol satisfies *balance soundness* if $\Pr[W > D] = \mathsf{negl}(\lambda)$.

Theorem 14.5 (Balance soundness)**.** *Under the knowledge soundness of Groth16 (Theorem 10.3), the collision resistance of Poseidon (Assumption 9.1), the accumulator soundness of the state tree (Proposition 9.11), and setup integrity (Assumption 10.4), the protocol satisfies balance soundness.*

Proof sketch. Consider any accepted withdrawal. By knowledge soundness there is an extractor producing a witness satisfying Construction 13.2. By (4) the witness contains a note $(\mathsf{value}, \mathsf{label}, \mathsf{nullifier}, \mathsf{secret})$ whose commitment is cm; by (6) and Proposition 9.11, cm is a genuine leaf of the state tree, i.e. the note corresponds to an actual prior deposit (or a prior change note, which by induction corresponds to earlier deposited value). By Proposition 13.3 the withdrawal conserves value: $v_{\mathsf{wd}} + v_{\mathsf{residual}} = \mathsf{value}$. The on-chain nullifier check ensures each note's value is consumed at most once (Theorem 14.7). Summing the conservation identity over all accepted withdrawals and telescoping the change-note chain, total withdrawn value cannot exceed total deposited net value except on the negligible-probability events of a Poseidon collision, a forged inclusion, or a forged proof (each excluded by the cited assumptions). □

14.3 Double-spend resistance

Definition 14.6 (Double-spend resistance)**.** The protocol is *double-spend resistant* if no PPT adversary can cause two accepting withdrawals that both consume the same note, i.e. that reveal the same nullifier hash, except with negligible probability.

Theorem 14.7 (Double-spend resistance)**.** *Under the collision resistance of* $\mathsf{Poseidon}_1$ *(Assumption 9.1) and knowledge soundness of Groth16 (Theorem 10.3), the protocol is double-spend resistant.*

Proof sketch. A note's nullifier hash $\mathsf{nf} = \mathsf{Poseidon}_1(\mathsf{nullifier})$ is fixed by the note. By (5) every accepted withdrawal exposes the nullifier hash of the note it extracts. The contract records each accepted nf and rejects any withdrawal whose nf is already recorded. Hence two accepted withdrawals of the same note would require two distinct accepted transactions with the same nf, contradicting the on-chain check; while two *different* notes producing the same nf would be a $\mathsf{Poseidon}_1$ collision, excluded by assumption. Either case occurs only with negligible probability. □

14.4 Compliance soundness

Finally, the association mechanism is sound in the sense that a withdrawal cannot be accepted for a note whose label is outside the published set.

Proposition 14.8 (Compliance soundness)**.** *Under accumulator soundness of the association tree (Proposition 9.11) and knowledge soundness of Groth16, any accepted withdrawal corresponds to a note whose label is a member of the association set at the latest published root, except with negligible probability.*

Proof sketch. By extraction and (7), the witness contains a valid inclusion path for the note's label against R_{ASP}; the on-chain check binds R_{ASP} to the latest published root. By Proposition 9.11, a valid path for a non-member label would yield a Poseidon collision. Hence the label is a genuine member. □

14.5 Anonymity set: a quantitative view

Theorem 14.2 establishes cryptographic unlinkability but, as its remark notes, the *practical* anonymity of a withdrawal is governed by the size and composition of the set of deposits it could plausibly have come from. We make this precise.

Definition 14.9 (Effective anonymity set)**.** Fix a withdrawal w of amount v_{wd} at time θ. Its *effective anonymity set* $\mathcal{A}(w)$ is the set of unspent, association-approved deposits present at time θ whose value is consistent with having funded w (in a full-withdrawal setting, those of value exactly v_{wd}; in the partial setting, those of value at least v_{wd}). The anonymity of w is measured by $|\mathcal{A}(w)|$ and by the entropy of the a-posteriori distribution an observer assigns over $\mathcal{A}(w)$.

An idealized observer, given only the protocol's public transcript and the zero-knowledge proof, has no information distinguishing members of $\mathcal{A}(w)$: by Theorem 14.2 the posterior is uniform over $\mathcal{A}(w)$, giving anonymity entropy $\log_2 |\mathcal{A}(w)|$ bits. Two forces erode this ideal:

- **Value fingerprinting.** If v_{wd} is distinctive—an amount matched by few deposits—then $|\mathcal{A}(w)|$ is small regardless of pool size. Round, common denominations maximize the set; idiosyncratic amounts shrink it. This motivates denomination discipline, and, in future work, fixed-denomination sub-pools.
- **Temporal correlation.** If w occurs immediately after a single fresh deposit, the posterior concentrates on that deposit despite a nominally large pool, because the observer conditions on arrival times. Delay between deposit and withdrawal, drawn from a distribution decorrelated from deposit timing, restores the uniform posterior.

Proposition 14.10 (Anonymity floor)**.** *Absent value and timing side information, a withdrawal from a pool in which k association-approved, value-compatible, unspent deposits are present enjoys anonymity entropy exactly $\log_2 k$ bits against an idealized observer. In particular a withdrawal against $k = 1$ has zero anonymity, and the cryptographic guarantee provides no cover in that degenerate case.*

The proposition is a caution as much as a result: the protocol's mathematics guarantees that nothing *beyond* membership in $\mathcal{A}(w)$ leaks, but the size of $\mathcal{A}(w)$ is an operational property the user co-determines by her choice of amount and timing. Operational guidance (Section 18) follows directly.

14.6 The relayer protocol

Direct withdrawal requires the recipient address to submit its own transaction and thus to hold gas, which may itself create a linkage (the gas must come from somewhere) and requires the recipient to be online. The relayer protocol removes both frictions while preserving the trust model.

Construction 14.11 (Relayed withdrawal)**.** A relayer $\mathcal{R}$ is an address willing to submit withdrawals for a fee.

1. The spender constructs the withdrawal proof with a descriptor W whose recipient is the desired fresh address and whose fee field authorizes a payment f to $\mathcal{R}$, bounded by the protocol's maximum relay-fee rate. The context ctx binds W, including recipient and f.
2. The spender transmits (π, pub, W) to $\mathcal{R}$ off-chain.
3. $\mathcal{R}$ submits the withdrawal, paying gas. The contract pays the recipient $v_{\mathsf{wd}} - f$ and the relayer f, per the descriptor.

Proposition 14.12 (Relayer safety)**.** *A relayer executing Construction 14.11 cannot redirect funds, inflate its fee beyond the authorized f, or learn the spender's note secrets, except with negligible probability.*

Proof sketch. The recipient and fee are fields of W, and $\mathsf{ctx} = \mathsf{keccak256}(\mathsf{encode}(W), \mathsf{scope}) \bmod r$ is a public signal bound into the proof and re-checked on-chain. Any alteration of W by the relayer changes ctx and invalidates the proof (Section 11.4). The relayer receives only (π, pub, W), all of which are public or zero-knowledge with respect to the secrets; by the zero-knowledge property of Groth16 it learns nothing of $(\mathsf{nullifier}, \mathsf{secret})$. Hence it can neither redirect, overcharge, nor deanonymize. □

The relayer is therefore a pure liveness/usability aid with no additional trust: worst case, a malicious relayer declines to submit, and the spender routes to another. Decentralizing the relayer set removes even this liveness dependency.

14.7 Full proofs of the core theorems

We expand two of the proof sketches of Section 14 into full arguments, for completeness. The others follow the same pattern.

Full proof of Theorem 14.7. Let $\mathcal{A}$ be a PPT adversary and suppose, for contradiction, that with non-negligible probability ϵ it causes two accepting withdrawals w_1, w_2 that consume the same note N. Let $\mathsf{nf}_1, \mathsf{nf}_2$ be the nullifier-hash signals of w_1, w_2.

Case 1: $\mathsf{nf}_1 = \mathsf{nf}_2$. The contract records nf_1 upon accepting w_1 (step 8 of Construction 11.8) and, upon processing w_2, executes the membership check of step 7(e), which rejects any withdrawal whose nullifier hash is already recorded. Thus w_2 cannot be accepted, contradicting the assumption that both are accepting. (If w_1, w_2 are in the same block, the state-transition function serializes them; the second to execute sees the first's recorded nullifier.)

Case 2: $\mathsf{nf}_1 \neq \mathsf{nf}_2$. Since both consume the same note N with nullifier $\mathsf{nullifier}$, knowledge soundness (Theorem 10.3) yields extractors producing witnesses for w_1, w_2; by (5) each witness satisfies $\mathsf{nf}_b = \mathsf{Poseidon}_1(\mathsf{nullifier})$ for the *same* $\mathsf{nullifier}$ (the note is fixed). But $\mathsf{Poseidon}_1$ is a function, so $\mathsf{Poseidon}_1(\mathsf{nullifier})$ is single-valued and $\mathsf{nf}_1 = \mathsf{nf}_2$, contradicting the case hypothesis. The only escape is an extraction failure, which occurs with negligible probability.

Combining, the probability that both withdrawals are accepting and consume N is bounded by the extraction-failure probability, which is negligible; hence $\epsilon = \mathsf{negl}(\lambda)$, a contradiction. Therefore no such $\mathcal{A}$ exists. □

Full proof of Proposition 14.8. Let w be any accepting withdrawal with association-root signal R_{ASP} and label label extracted from its witness. The on-chain validation (step 7(c)) enforces that R_{ASP} equals the latest published association root $R^\star$. By knowledge soundness the extracted witness satisfies (7), i.e. $R_{\mathsf{ASP}} = \mathrm{root}(\mathsf{label}, j_{\mathsf{ASP}}, \sigma_{\mathsf{ASP}}, d_{\mathsf{ASP}})$ for some index and sibling list, so $(\mathsf{label}, j_{\mathsf{ASP}}, \sigma_{\mathsf{ASP}}, d_{\mathsf{ASP}})$ is a valid inclusion witness for root $R^\star$. Suppose $\mathsf{label} \notin \mathcal{S}$, the leaf set underlying $R^\star$. Then by Proposition 9.11 the existence of a valid inclusion witness for the non-member label implies a $\mathsf{Poseidon}_2$ collision, obtainable by the reduction of that proposition, contradicting Assumption 9.1. Hence $\mathsf{label} \in \mathcal{S}$: the note's deposit is a genuine member of the association set at the latest root, except with negligible probability. □

14.8 On the random-oracle idealization

Several results (Proposition 9.6 and Theorem 14.2) model Poseidon as a random oracle. We comment on the strength and necessity of this idealization. The random-oracle model (ROM) treats the hash as an external uniformly-random function to which all parties have oracle access; proofs in the ROM are heuristic evidence of security when the oracle is instantiated by a concrete hash. For the hiding and unlinkability arguments, the ROM is used only to conclude that a commitment under high-entropy secrets is distributed independently of those secrets until the exact preimage is queried—a property that a collision-resistant, preimage-resistant algebraic

hash is believed to satisfy in practice. The soundness-type results (balance, double-spend, compliance) do *not* rely on the ROM; they require only collision resistance (Assumption 9.1) and the standard-model knowledge soundness of Groth16. Thus the protocol's *integrity* guarantees rest on standard-model assumptions, and only its *privacy* guarantees invoke the ROM, in line with the broader literature on note-based privacy systems.

14.9 Formal state-transition semantics

We give the pool as a labeled transition system, so that the invariants of Appendix G can be read directly off the transition rules. Let a pool configuration be

$$\Gamma = \big(T_{\mathsf{state}},\ \mathcal{N},\ \mathcal{H},\ R^{\star}_{\mathsf{ASP}}\big),$$

where T_{state} is the state tree (with root R_{state} and its insertion history), $\mathcal{N} \subseteq \mathbb{F}_r$ is the spent-nullifier set, $\mathcal{H}$ is the bounded window of historical state roots, and $R^{\star}_{\mathsf{ASP}}$ is the latest association root. Two transition rules act on Γ.

Deposit transition. On $\mathsf{deposit}(\mathsf{precommitment})$ with value v:

$$\Gamma \xrightarrow{\ \mathsf{dep}(v,\mathsf{precommitment})\ } \Gamma',$$

where, writing $\mathsf{value} = v(1-\phi)$, $\mathsf{label} = \mathsf{keccak256}(\mathsf{scope}, \eta) \bmod r$, and $\mathsf{cm} = \mathsf{Poseidon}_3(\mathsf{value}, \mathsf{label}, \mathsf{precommitme}$ the successor configuration Γ' has $T'_{\mathsf{state}} = T_{\mathsf{state}} \Vdash \mathsf{cm}$ (append), R'_{state} the new root, $\mathcal{H}' = \mathcal{H} \cup \{R_{\mathsf{state}}\}$, and $\mathcal{N}, R^{\star}_{\mathsf{ASP}}$ unchanged. The rule has no precondition beyond $v \geq \mathsf{minDep}$; deposits never fail on state grounds.

Withdrawal transition. On $\mathsf{withdraw}(W, \pi, \mathsf{pub})$ with $\mathsf{pub} = (\mathsf{cm}^{\mathsf{new}}, \mathsf{nf}, v_{\mathsf{wd}}, R, d_{\mathsf{s}}, R_{\mathsf{ASP}}, d_{\mathsf{ASP}}, \mathsf{ctx})$:

$$\Gamma \xrightarrow{\ \mathsf{wd}(W,\pi,\mathsf{pub})\ } \Gamma' \quad \text{iff all guards hold:}$$

$$\begin{aligned}
&\textsc{caller} = W.\mathsf{processor}, \quad \mathsf{ctx} = \mathsf{keccak256}(\mathsf{encode}(W), \mathsf{scope}) \bmod r,\\
&R \in \mathcal{H} \cup \{R_{\mathsf{state}}\}, \quad R_{\mathsf{ASP}} = R^{\star}_{\mathsf{ASP}}, \quad d_{\mathsf{s}}, d_{\mathsf{ASP}} \leq D_{\max},\\
&\mathsf{nf} \notin \mathcal{N}, \quad \textsc{Groth16.Verify}(\mathsf{vk}, \mathsf{pub}, \pi) = 1.
\end{aligned}$$

When the guards hold, Γ' has $\mathcal{N}' = \mathcal{N} \cup \{\mathsf{nf}\}$, $T'_{\mathsf{state}} = T_{\mathsf{state}} \Vdash \mathsf{cm}^{\mathsf{new}}$, updated $R'_{\mathsf{state}}, \mathcal{H}'$, and $R^{\star}_{\mathsf{ASP}}$ unchanged; the transition also emits the transfer of v_{wd} per W. If any guard fails the transition does not fire and Γ is unchanged.

Proposition 14.13 (Guard completeness)**.** *The withdrawal guards jointly enforce every property required by Theorems 14.5 and 14.7 and Proposition 14.8: caller and context guards enforce integrity of the withdrawal descriptor; the root guards enforce authenticity of the two accumulator roots; the nullifier guard enforces no-double-spend; and the verification guard enforces the circuit relation of Construction 13.2.*

The transition system makes explicit that the only way to remove value from the pool is a withdrawal whose guards hold, and that each such transition consumes exactly one fresh nullifier and conserves value—the content of the invariants.

14.10 Consolidated assumptions

For reference we collect the assumptions on which the analysis rests, and record which guarantees depend on each.

Assumption	Type	Guarantees relying on it
Poseidon collision resistance	standard model	balance, double-spend, compliance
Poseidon as random oracle	idealized	hiding, unlinkability
Groth16 knowledge soundness	GGM / knowledge	balance, double-spend, compliance
Groth16 zero-knowledge	standard (given crs)	unlinkability
Trusted-setup integrity (1-of-N)	one-time ceremony	all soundness guarantees
Contract correctness	implementation	all guarantees
Chain liveness / bounded reorg	network	liveness, root authenticity

The separation is deliberate: the protocol's *soundness* (nobody steals or double-spends, every exit is compliant) rests only on standard-model assumptions plus the one-time ceremony; the *privacy* guarantees additionally invoke the random-oracle idealization. A reader who rejects the ROM still obtains all integrity properties.

14.11 Cost and performance

We summarize the concrete costs that determine feasibility.

Proving. The withdrawal proof is generated client-side. Its cost is dominated by the two Merkle inclusions (each $D_{\max}$ Poseidon evaluations) and the commitment/nullifier hashing, compiled into an R1CS whose size fixes the number of multi-scalar multiplications in Groth16 proving. The proving key is on the order of tens of megabytes and is downloaded once and cached; proving itself completes in seconds on a commodity machine and runs entirely in the browser.

Verification. On-chain verification is three pairings and a multi-scalar multiplication over the eight public inputs—constant cost, independent of pool size or tree occupancy. This is what keeps withdrawal gas bounded as the pool grows.

Proof size. A Groth16 proof is three group elements: two in $\mathbb{G}_1$ and one in $\mathbb{G}_2$, a few hundred bytes total, independent of the statement's complexity.

Note discovery. Discovery scans deposit events and trial-derives secrets per index up to a gap limit; its cost is linear in the number of scanned indices and in the event count, and is the one operation whose cost grows with pool history. Caching and indexing mitigate this and are an engineering, not a protocol, concern.

The upshot is that the two costs which must scale well with adoption—on-chain verification and proof size—are constant, while the costs that grow (proving key download, discovery) are one-time or client-local and amenable to standard optimization.

15 Design Rationale

We collect, in one place, the reasoning behind the protocol's less-obvious design choices, so that a reader can distinguish essential structure from incidental convention.

15.1 Why commitments hide precommitments, not raw secrets

The commitment is $\mathsf{Poseidon}_3(\mathsf{value}, \mathsf{label}, \mathsf{precommitment})$ where $\mathsf{precommitment} = \mathsf{Poseidon}_2(\mathsf{nullifier}, \mathsf{secret})$, rather than a single hash of all four note fields. The two-layer structure serves the deposit ordering of Section 11.2: the client can compute and submit precommitment *before* the contract assigns label, and the contract then forms the full commitment using its assigned label without needing the secrets. A single-layer commitment would require the label at derivation time, which the client does not yet have. The layering thus reconciles client-side secret custody with contract-side label assignment.

15.2 Why the nullifier is hashed with a distinct arity

The nullifier hash uses $\mathsf{Poseidon}_1$—a distinct arity from the precommitment's $\mathsf{Poseidon}_2$ and the commitment's $\mathsf{Poseidon}_3$. Distinct arities carry distinct domain separations (Appendix A), ensuring the nullifier hash cannot be confused with a commitment or precommitment. This matters because the nullifier hash is public while the nullifier is secret; a shared hash function across roles could let an adversary relate the public nullifier hash to other public hashes of the same secret.

15.3 Why labels, not commitments, seed the association tree

As argued in Appendix E, keying the association tree on labels lets the approved set be published in full without revealing ownership-bearing data. A label is a public, per-deposit identifier with no link to the note's value or secrets; publishing the set of approved labels leaks nothing a deposit event did not already reveal. Keying on commitments, by contrast, would publish the approved commitments and could aid the very linkage the protocol prevents.

15.4 Why context binds the descriptor into the proof

Without the context binding, a valid withdrawal proof observed in the mempool could be lifted and resubmitted by an adversary with a different recipient, stealing the funds. By making $\mathsf{ctx} = \mathsf{keccak256}(\mathsf{encode}(W), \mathsf{scope}) \bmod r$ a public signal that the circuit binds and the contract re-derives from the submitted W, the proof is cryptographically welded to its intended descriptor: change the recipient, change W, change ctx, invalidate the proof. This is the standard defense against proof malleability and front-running in the withdrawal path.

15.5 Why a bounded root history

The contract accepts withdrawals against any state root in a bounded recent history, not only the latest. This tolerates the natural race in which a user builds a proof against the current root and, before her transaction lands, another deposit advances the root. Without a history window, every concurrent deposit would invalidate in-flight withdrawal proofs; with it, proofs remain valid for a reasonable window. The window is bounded so that the set of accepted roots stays small and the anonymity implications remain controlled.

15.6 Why proving is client-side despite its cost

Client-side proving imposes real costs—a large proving-key download and seconds of computation—that a server could absorb. The protocol nonetheless insists on client-side proving because the witness contains the note secrets, and outsourcing proof generation would mean disclosing those secrets to the prover. There is no partial measure: either the secrets stay on the device, or the privacy guarantee is forfeited. The cost is the price of the guarantee, and the roadmap's performance work aims to reduce the cost, never to relocate the secret.

16 System Architecture

We describe the deployed realization of the protocol, separating on-chain from off-chain components and identifying where each cryptographic object of the preceding sections lives.

16.1 On-chain components

- **Entrypoint (upgradeable proxy).** The user-facing contract. It accepts deposits, charges the vetting fee, assigns labels, forwards net value and precommitments to the pool, and exposes the association-root registry and its update method. It holds the access-control roles governing configuration and ASP publication.
- **Pool.** Maintains the state tree of commitments, the nullifier set, and the pooled value; validates withdrawals (caller authorization, context match, known state root, latest association root, unspent nullifier) and invokes the verifier.
- **Withdrawal verifier.** A Groth16 verifier contract for the withdrawal relation, embedding the verifying key; evaluates the pairing check of Construction 10.2 over the submitted proof and public signals.
- **Commitment verifier.** A Groth16 verifier for the commitment relation, used where standalone commitment validity is required.
- **Poseidon contracts.** On-chain evaluators $\mathsf{Poseidon}_1, \mathsf{Poseidon}_2, \mathsf{Poseidon}_3$ (as `PoseidonT2`, `PoseidonT3`, `PoseidonT4`), used for label formation, commitment reconstruction, and Merkle updates. Their outputs are required to match the client and circuit evaluations bit-for-bit.

16.2 Off-chain and client components

- **Client (browser).** Performs all secret-dependent computation: seed management and note derivation (?? 9.7?? 9.8), note discovery, Merkle-witness construction, and Groth16 proof generation. Proving runs locally against the served proving key and circuit WASM; no secret leaves the device.
- **SDK.** A library exposing note derivation, precommitment and commitment computation, Merkle-proof generation, and withdrawal-proof orchestration, consumed by the client.
- **Association-set service.** Maintains the association tree, publishes the root through the ASP role, and serves the leaf set to clients so that they can construct association-inclusion witnesses.
- **Relayer (optional).** Submits withdrawal transactions on behalf of a spender so that the recipient address need not hold gas or reveal a funding link, strengthening operational unlinkability. The context binding (Section 11.4) ensures a relayer cannot alter the recipient or overcharge the fee.

16.3 The client-side proving pipeline

Because Groth16 proving for the withdrawal relation runs entirely in the browser, the client must load the circuit WASM and the proving key (on the order of a few and tens of megabytes respectively) and execute a multi-scalar-multiplication-heavy proving routine. This is the most computationally demanding user-facing operation in the system. Its feasibility in a commodity browser is what makes the non-custodial, no-secret-leaves-device model practical rather than merely aspirational: the alternative—outsourcing proving to a server—would require disclosing the witness and defeat the privacy guarantee.

17 Threat Model

We state the adversary's capabilities and the trust assumptions on which each guarantee depends, consolidating the assumptions used piecemeal above.

17.1 Adversary capabilities

The adversary may: observe all on-chain state and all network traffic; submit arbitrary deposits and withdrawals; act as one or more users; reorder, delay, or censor transactions subject to the underlying chain's liveness; and adaptively choose its actions based on observed state. It may corrupt the relayer. It runs in probabilistic polynomial time.

17.2 Trust assumptions

1. **Cryptographic hardness.** Poseidon is collision resistant and behaves as a random oracle where invoked (Assumption 9.1); the pairing assumptions underlying Groth16 hold on BN254 (Theorem 10.3).

2. **Setup integrity.** The withdrawal-circuit trusted setup had at least one honest participant (Assumption 10.4).

3. **Contract correctness.** The deployed contracts faithfully implement the specified checks; this is an implementation assumption addressed by audit and by the empirical Poseidon-compatibility verification.

4. **Chain liveness and integrity.** The underlying chain does not reorganize beyond the pool's root-history window and eventually includes submitted transactions.

17.3 Explicitly out of scope

The protocol does *not* defend against: compromise of the user's device or seed; network-level deanonymization linking a deposit and a withdrawal by IP or timing; statistical linkage when amounts or timing are distinctive or the anonymity set is small; and legal or policy consequences of the ASP's curation choices. These are addressed, where possible, by operational guidance rather than by the cryptographic protocol.

18 Limitations and Discussion

We close with an honest account of the protocol's limitations.

Anonymity-set size. The privacy a withdrawal enjoys is upper-bounded by the number of indistinguishable deposits in the pool at withdrawal time. Early in a pool's life, or for unusual denominations, this set may be small, and Theorem 14.2's guarantee—though cryptographically intact—provides little practical cover. Users seeking strong anonymity should withdraw into a large, active anonymity set and avoid distinctive amounts.

Timing and network metadata. The cryptographic unlinkability of Theorem 14.2 says nothing about correlation by timing or network origin. A deposit and a withdrawal issued from the same IP in close succession may be linked by an observer with network vantage, entirely outside the proof system. A relayer and delay between deposit and withdrawal mitigate but do not eliminate this.

Association-set trust. The ASP is a real trust dependency. While it cannot seize funds or deanonymize withdrawals (Section 12.4), it does determine which deposits are privately withdrawable, and the integrity and availability of its published sets are operational assumptions. A deployment must decide, and be accountable for, its curation policy; that decision carries legal weight beyond this document's scope.

Setup and implementation trust. Soundness rests on the trusted setup (Assumption 10.4) and on the faithful implementation of the contracts and circuits. Both are one-time, auditable dependencies, but they are dependencies nonetheless. The proving key and ceremony transcript are public artifacts subject to independent verification.

Client performance. In-browser Groth16 proving imposes a nontrivial memory and time cost and a large one-time download of the proving key. On constrained devices this may be a usability barrier, though it is the price of keeping secrets on-device.

Fund-safety on derivation. As noted in Section 11.3, a note whose commitment is derived with an incorrect value or label—or against a mismatched Poseidon implementation—is permanently unspendable. Correctness of the derivation pipeline is a fund-safety property and must be validated against the deployed hash evaluators, not assumed.

Scope of novelty. We reiterate that the cryptographic design is that of the Privacy Pools framework [1] and its reference implementation; this document specifies and analyzes a deployment of that design, not a new primitive.

A Extended Circuit Notes

We record additional detail on the constraint systems, useful to an implementer or auditor.

A.1 Depth handling and padding

Both inclusion proofs fold to a fixed maximum depth $D_{\max}$, with the effective depth supplied as a public signal and the sibling list zero-padded above it. The folding gadget must ensure that padded levels do not alter the recomputed root: above the active depth, the accumulated hash is carried through unchanged rather than folded against a padding sibling. Correct depth handling is a common source of implementation error, and it is checked both in-circuit (the active-depth signal gates the fold) and on-chain (the depth signals are bounded by $D_{\max}$ in the withdrawal guards).

A.2 Range checks and field safety

The value range check (Equation (8)) decomposes v_{wd} and value into L bits and asserts each bit Boolean, with L chosen so that $2^L < r$. This prevents two distinct failure modes: negative values (which in a prime field appear as large positives near r) and modular wraparound in the conservation identity. Without the range check, an adversary could set $\mathsf{value} - v_{\mathsf{wd}}$ to wrap the modulus and manufacture value in the change note; the check forecloses this by confining all values to a subinterval where field and integer arithmetic agree.

A.3 Domain separation

Each Poseidon arity uses a distinct capacity initialization, providing domain separation between the precommitment hash ($\mathsf{Poseidon}_2$), the commitment hash ($\mathsf{Poseidon}_3$), the nullifier

hash ($\mathsf{Poseidon}_1$), and the Merkle node hash ($\mathsf{Poseidon}_2$ over node pairs). Domain separation ensures that a value valid in one role cannot be reinterpreted in another—for instance, that a Merkle node cannot be passed off as a precommitment— closing a class of cross-protocol confusion attacks.

A.4 Public-signal integrity

The eight public signals (Section 10.6) are the sole channel through which the proof communicates with the chain. Their ordering is fixed and shared by the circuit, the proof-formatting library, and the on-chain verifier; a mismatch in ordering between any two of these would cause either spurious rejection or, worse, acceptance of a proof about different quantities than intended. The ordering is therefore treated as part of the protocol specification rather than an implementation detail, and is fixed in Section 10.6.

B Notation Summary

Symbol	Meaning
r	BN254 scalar field order (SNARK modulus)
$\mathbb{F}_r$	scalar field $\mathbb{Z}/r\mathbb{Z}$, the in-circuit working field
$\mathsf{Poseidon}_k$	fixed-arity Poseidon hash, $\mathbb{F}_r^k \to \mathbb{F}_r$
value	note value (net of fees)
label	deposit label, $\mathsf{keccak256}(\mathsf{scope}, \eta) \bmod r$
$\mathsf{nullifier}, \mathsf{secret}$	note nullifier and secret
$\mathsf{precommitment}$	precommitment, $\mathsf{Poseidon}_2(\mathsf{nullifier}, \mathsf{secret})$
cm	commitment, $\mathsf{Poseidon}_3(\mathsf{value}, \mathsf{label}, \mathsf{precommitment})$
nf	nullifier hash, $\mathsf{Poseidon}_1(\mathsf{nullifier})$
scope	pool domain separator
R_{state}	state-tree root (leaves: commitments)
R_{ASP}	association-tree root (leaves: labels)
ctx	context, $\mathsf{keccak256}(\mathsf{encode}(W), \mathsf{scope}) \bmod r$
$D_{\max}$	maximum Merkle folding depth
ϕ	vetting-fee rate

C Glossary

Association set
A curated set of deposit labels deemed acceptable, represented as a Merkle tree whose root is published on-chain. Membership is proven in zero knowledge at withdrawal.

Commitment
The public per-deposit value $\mathsf{Poseidon}_3(\mathsf{value}, \mathsf{label}, \mathsf{precommitment})$ inserted into the state tree; hides the note's secrets.

Context
A hash binding a withdrawal proof to its recipient and fee parameters, preventing re-targeting.

Label
A public per-deposit identifier $\mathsf{keccak256}(\mathsf{scope}, \eta) \bmod r$ assigned by the contract; the leaf of the association tree.

Nullifier
A secret component of a note; its hash $\mathsf{Poseidon}_1(\mathsf{nullifier})$ is revealed at withdrawal to prevent double-spending.

Note
A hidden record of value $(\mathsf{value}, \mathsf{label}, \mathsf{nullifier}, \mathsf{secret})$ controlled by its owner.

Precommitment
The inner hash $\mathsf{Poseidon}_2(\mathsf{nullifier}, \mathsf{secret})$ binding a note's secret material.

Scope
A per-pool domain separator binding notes and contexts to a specific pool.

State tree
The Merkle accumulator of all deposit commitments.

Vetting fee
A fee charged on deposit; the note value is the deposit net of it.

zk-SNARK
A zero-knowledge succinct non-interactive argument of knowledge; here, Groth16, used to prove the withdrawal relation.

D Deployment Reference

The reference deployment targets Robinhood Chain (an Ethereum-compatible execution environment, chain identifier 4663). The instantiated pool is a native-currency (ETH) pool with a configured minimum deposit and vetting-fee rate. The deployment comprises the Entrypoint proxy and its implementation, the ETH pool, the withdrawal and commitment Groth16 verifiers, and the Poseidon evaluator contracts, together with an association-root registry administered through the ASP role. Concrete contract addresses, the pool scope, and the current configuration parameters are published separately as deployment metadata and are intentionally omitted here so that this specification remains valid across redeployments and parameter updates. The on-chain Poseidon evaluators are verified, against client and circuit evaluations, to agree bit-for-bit on representative inputs, discharging the fund-safety compatibility requirement of Section 11.3.

E Detailed Accumulator Semantics

We give the lean-IMT insertion and proof algorithms in full, since the accumulator's correctness underlies both soundness (Proposition 9.11) and the fund-safety requirement that client-reconstructed roots match on-chain roots.

E.1 Insertion

Input: current leaves $\ell_0, \ldots, \ell_{n-1}$, frontier nodes, new leaf ℓ_n
Output: updated root
$h \leftarrow \ell_n$; $pos \leftarrow n$
for *level* $\leftarrow 0$ ***to*** $d-1$ **do**
  if *pos is odd* **then**
    $h \leftarrow \mathsf{Poseidon}_2(leftSibling[level],\ h)$
  else
    `// no right sibling yet: promote unchanged, record as frontier`
    record h as frontier at *level*; **break**
  end
  $pos \leftarrow \lfloor pos/2 \rfloor$
end
recompute apex from frontier; **return** new root

Algorithm 4: lean-IMT insert

The insertion updates only the $O(\log n)$ nodes along the new leaf's path and promotes a node unchanged wherever its right sibling is absent, which is the mechanism by which the lean variant avoids materializing canonical zero-subtrees. The depth d increases by one exactly when n crosses a power of two.

E.2 Proof generation and verification

Input: leaves, target index i, claimed root R
Output: sibling list σ; verification bit
$\sigma \leftarrow []$; $pos \leftarrow i$
for *level* $\leftarrow 0$ ***to*** $d-1$ **do**
  $s \leftarrow$ sibling of node at $(level, pos)$, or 0 if absent
  append s to σ; $pos \leftarrow \lfloor pos/2 \rfloor$
end
pad σ with zeros to length $D_{\max}$
`// verification:`
$h \leftarrow \ell_i$
for $j \leftarrow 0$ ***to*** $d-1$ **do**
  if *bit* j *of* i *is* 0 **then**
    $h \leftarrow \mathsf{Poseidon}_2(h, \sigma_j)$
  else
    $h \leftarrow \mathsf{Poseidon}_2(\sigma_j, h)$
  end
end
return σ; $[h = R]$

Algorithm 5: lean-IMT prove and verify

The verification loop is exactly Definition 9.10, and is the computation performed in-circuit by the inclusion gadget of Section 13. The client, the circuit, and the on-chain root validation all execute this same fold; their agreement is the fund-safety compatibility property emphasized throughout.

E.3 Why two accumulators, revisited

The protocol's use of two accumulators with different leaf types is worth restating as an accumulator-level design choice. The state tree accumulates commitments and answers "does this note exist in the pool?"; the association tree accumulates labels and answers "is this note's deposit ap-

proved?". A single tree cannot serve both roles without leaking: if the association tree keyed on commitments, publishing it would reveal the set of approved commitments and, combined with the state tree, could aid linkage; keying on labels—public per-deposit identifiers that carry no ownership information—lets the approval set be published in full without compromising the privacy of any withdrawal. The separation is thus not incidental but load-bearing for the privacy guarantee.

F Symbol and Function Reference

Object	Signature	Definition
$\mathsf{Poseidon}_1$	$\mathbb{F}_r \to \mathbb{F}_r$	nullifier hash
$\mathsf{Poseidon}_2$	$\mathbb{F}_r^2 \to \mathbb{F}_r$	precommitment, Merkle node
$\mathsf{Poseidon}_3$	$\mathbb{F}_r^3 \to \mathbb{F}_r$	commitment
keccak256	$\{0,1\}^* \to \{0,1\}^{256}$	label, context (then mod r)
root	(leaf, idx, sibs, depth) $\to \mathbb{F}_r$	Merkle fold (Definition 9.10)
MasterKeys	mnemonic $\to \mathbb{F}_r^2$	master key derivation
Groth16.Prove	(pub, wit) $\to \pi$	proof generation
Groth16.Verify	(vk, pub, π) $\to \{0,1\}$	proof verification

The three Poseidon arities, the two Keccak-derived reductions, the Merkle fold, and the Groth16 pair constitute the complete set of cryptographic operations in the protocol. Everything else—the state machine, the guards, the accounting—is composed from these.

G Protocol Invariants

For reference we collect the invariants maintained across all honest and adversarial executions.

1. **Commitment well-formedness.** Every state-tree leaf equals $\mathsf{Poseidon}_3(\mathsf{value}, \mathsf{label}, \mathsf{Poseidon}_2(\mathsf{nullifier}, \mathsf{secre}$ for some note.

2. **Nullifier uniqueness.** Each nullifier hash appears in the spent set at most once; acceptance implies prior absence.

3. **Value conservation.** Every withdrawal satisfies $\mathsf{value} = v_{\mathsf{wd}} + v_{\mathsf{residual}}$ with all terms in $[0, 2^L)$.

4. **Root authenticity.** Every accepted proof references a state root in the pool's history and the latest association root.

5. **Context integrity.** Every accepted withdrawal's context equals $\mathsf{keccak256}(\mathsf{encode}(W), \mathsf{scope}) \bmod r$ for its submitted descriptor W.

6. **Non-custody.** No state transition moves a specific note's value without a witness containing that note's secrets.

Part III — Ecosystem

H User Journey

To make the abstract concrete, we trace a full round trip from the user's point of view, with no cryptographic detail.

Setting up. On first use, the app generates a twelve-word recovery phrase. This phrase is the user's entire identity in the system: every note derives from it, and it is the only way to recover funds. The user backs it up. No account, no email, no password, no server holds it.

Depositing. The user connects a wallet, chooses an amount, and deposits. The app shows the fee and the resulting note value before confirming. The transaction is a normal on-chain transfer; anyone can see the deposit address and amount. The user's secret note is derived locally and never transmitted.

Waiting. Privacy grows with the crowd. The longer a deposit sits among many others, and the more deposits accumulate, the larger the anonymity set a later withdrawal blends into. A withdrawal issued seconds after a deposit, from the same network origin, is easy to correlate by timing alone; a withdrawal from a mature pool is not.

Getting approved. For the withdrawal to succeed, the deposit's label must be in the published association set. Under the deployment's policy, this happens when the association-set service includes it and publishes the updated root. This is the compliance step: the system attests that the deposit is approved.

Withdrawing. The user enters her recovery phrase; the app scans the chain and finds her notes, marking which are already spent. She selects an unspent note, chooses a fresh destination address, and clicks withdraw. The browser generates a zero-knowledge proof locally—the heaviest computation in the flow—and submits it. The funds arrive at the fresh address, unlinked from the original deposit.

Recovering. On any device, the same recovery phrase reconstructs every note. There is no support desk to call and no reset to request, because there is nothing for anyone else to reset. The phrase is the account.

I Economic Model

I.1 Protocol fees

The pool charges a vetting fee on deposit, a small percentage of the deposited amount, expressed in basis points and enforced by the contract. The fee is deducted at deposit time; the note's value is the deposited amount net of the fee, and this net value is what is later withdrawable. Fees accrue to the protocol and fund its operation, including the association-set service and ongoing development. The exact rate is a configured parameter and is published as part of the deployment metadata rather than fixed in this document, so that it can be governed transparently.

I.2 Relaying

Withdrawals may be submitted directly by the user or, for stronger operational privacy, through a relayer that pays gas on the user's behalf so that the receiving address need never hold funds linkable to the user. A relayer may charge a fee for this service, bounded by protocol parameters and bound into the proof's context so that it cannot be altered after the fact. Relaying is optional and does not change the protocol's trust model: a relayer can neither redirect funds nor learn the user's secrets.

I.3 The $RAST token

The protocol's native token is $RAST. Its purpose is to decentralize control over the protocol's few points of human discretion and to coordinate the economic actors who operate its infrastructure. We fix the name and the design intent here; the quantitative parameters are being finalized and are marked as to-be-determined (TBD) below rather than stated prematurely.

Intended utility. $RAST is expected to serve three functions:

- **Governance.** Holders govern the protocol's discretionary parameters—fee rates, association-set policy administration, relayer parameters, and the schedule of progressive decentralization.
- **Infrastructure coordination.** The token aligns the incentives of relayers and the association-set service, the two off-chain roles whose liveness the user experience depends on.
- **Value alignment.** Protocol fees and token economics are intended to be coupled so that those who secure and operate the system share in its success, without inserting any rent between a user and her own funds.

Parameters (TBD). The following are placeholders pending a dedicated tokenomics specification:

Parameter	Value
Token name	$RAST
Total supply	TBD
Initial circulating supply	TBD
Emission / inflation schedule	TBD
Distribution (community / team / treasury)	TBD
Governance activation threshold	TBD
Fee-to-token coupling mechanism	TBD

What does not depend on the token. We state this explicitly because it is a design guarantee, not a marketing point: the core properties of the protocol—non-custody, unlinkability, and provable compliance—hold whether or not a user ever acquires, holds, or interacts with $RAST. The token governs and coordinates; it is never a gate between a user and the ability to deposit, prove, and withdraw her own funds. A user who never touches $RAST receives the full cryptographic guarantees of the protocol.

J Roadmap

The protocol is developed in stages, each of which strengthens either its guarantees or its usability. We describe the direction rather than fixed dates.

1. **Core protocol on Robinhood Chain.** Deposit and withdrawal live, with client-side proving, an ETH pool, and the association-set mechanism. This is the foundation and it is operational.
2. **Association-set automation.** Moving association-root publication from a manual step to a continuously running service, so that new deposits become withdrawable without operator intervention.
3. **Relayer network.** A relayer service (and eventually a decentralized set of relayers) for gasless, origin-unlinked withdrawals, closing the operational privacy gap that direct submission leaves open.

4. **Multi-asset pools.** Extending beyond the native-currency pool to fungible tokens, each with its own scope, state tree, and configuration.

5. **Governance and $RAST.** Progressive decentralization of the protocol's discretionary parameters, coordinated through the token.

6. **Independent audit.** Security review of contracts and circuits, and publication of the trusted-setup ceremony transcript for independent verification, ahead of scaling to significant value.

K Frequently Asked Questions

Is this a mixer? No. A mixer hides everyone indiscriminately, including illicit funds, and offers no way to distinguish clean money. ra:st requires every withdrawal to prove, in zero knowledge, that it belongs to an approved association set. It is a shielded pool with compliance built in, not a mixer.

Can you freeze or seize my funds? No. The protocol is built so that no operator, and no administrative key, can move a specific user's funds. Only the note secret—derived from your recovery phrase—can do that. This is a deliberate limitation on our own power.

What happens if I lose my recovery phrase? Your funds become permanently inaccessible. There is no recovery mechanism, because a recovery mechanism would mean someone other than you could access your funds. Back up your phrase.

Does the deposit reveal my identity? The deposit reveals that a particular address deposited a particular amount—this is public. It does not reveal your note secret, and it cannot be linked to your later withdrawal. Your practical privacy depends on the size of the anonymity set and on avoiding timing and network correlation (see below).

Is it truly private? The cryptographic unlinkability between deposit and withdrawal is strong and provable. But privacy in practice is also bounded by factors outside the cryptography: the number of comparable deposits in the pool, the timing of your withdrawal, and network-level metadata such as IP address. Withdrawing into a large, active pool, after a delay, and ideally through a relayer, materially improves your privacy.

Do I have to trust the association-set provider? You must trust it for one thing only: to include your deposit so that you can withdraw with privacy. It cannot take your funds, cannot deanonymize your withdrawal, and cannot alter a published set. Its influence is confined to privacy, never to ownership.

L Risk Disclosures

We state the material risks plainly.

- **Loss of secret.** Losing your recovery phrase means losing your funds, irreversibly. This is inherent to self-custody.
- **Small anonymity sets.** Early in a pool's life, or for unusual amounts, the anonymity set may be too small to provide meaningful privacy despite the cryptography.
- **Metadata linkage.** Timing and network-level observation can link a deposit and withdrawal outside the protocol's cryptographic guarantees.

- **Trusted setup.** Soundness depends on at least one honest participant in the setup ceremony. The ceremony transcript is public and independently verifiable.
- **Smart-contract and implementation risk.** As with any on-chain system, undiscovered bugs in contracts or circuits could affect funds. Independent audit is on the roadmap and should precede scaling to significant value.
- **Regulatory and policy risk.** The choice of association-set curation policy carries legal significance that varies by jurisdiction and is the responsibility of the deploying operator. Users should understand the applicable rules in their own jurisdiction.

Nothing in this document is financial, legal, or investment advice.

M Operational Security Guidance

The cryptographic guarantees of Section 14 are necessary but not sufficient for privacy in practice. This section translates the analysis of Section 14.5 into concrete guidance for users, since the protocol's mathematical anonymity can be squandered by operational mistakes it cannot prevent.

M.1 Preserving unlinkability

Withdraw into a mature anonymity set. By Proposition 14.10, anonymity is $\log_2 k$ bits where k is the number of compatible, approved, unspent deposits present. A user should prefer to withdraw when the pool is populous and the amount common, and should avoid being the sole recent deposit of her denomination.

Decorrelate timing. A withdrawal issued immediately after a deposit collapses the posterior onto that deposit regardless of pool size. Introduce a delay drawn independently of deposit timing; the longer and less predictable the delay, the closer the observer's posterior returns to uniform.

Break network linkage. The proof hides the on-chain link, but network metadata can re-establish it: a deposit and a withdrawal from the same IP address, or funded by the same gas source, may be correlated off-chain. Use a relayer (Section 14.6) so that the withdrawal transaction and its gas originate independently of the user, and consider network-level anonymization for the submission itself.

Use fresh destinations. Withdrawing to an address with prior on-chain history, or one later linked to the user's identity, reintroduces linkage downstream of the protocol. The destination should be fresh and should not be consolidated with identified funds.

M.2 Protecting the secret

The recovery phrase is the account. There is no server-side copy, no reset, and no recovery. Back up the phrase offline, redundantly, and treat its compromise as equivalent to loss of all associated funds. Anyone who learns the phrase can derive every note and spend it; the protocol cannot distinguish the rightful owner from anyone else holding the same secret.

Client integrity. Because proving is client-side, the integrity of the client matters: a tampered client could exfiltrate the secret at derivation time. Users should obtain the client from a trusted source and, where possible, verify its integrity.

M.3 Understanding the compliance boundary

A withdrawal succeeds only if the deposit's label is in the published association set. A user should understand her deployment's curation policy before depositing, since a deposit whose label is never approved cannot be withdrawn with privacy (though it is never seized; see Section 12.4). The compliance boundary is a property of the deployment, not of the protocol, and users bear responsibility for understanding the rules applicable to them.

N Deployment and Operations

We describe the operational lifecycle of a deployment, complementing the architectural view of Section 16.

N.1 Instantiation

Deploying a pool fixes its immutable parameters—scope, asset, minimum deposit, fee rate—and installs the verifier contracts with the verifying key from the trusted-setup ceremony. Because these parameters are immutable once set, a deployment commits to them for the life of the pool; changing them requires a new pool. This immutability is a feature: it removes the operator's ability to alter the rules governing funds already deposited, which is a precondition of the non-custody guarantee.

N.2 Association-set operation

The association-set service observes deposits, applies the deployment's curation policy to each new label, maintains the association tree, and publishes updated roots through the ASP role. Its liveness determines when new deposits become withdrawable; its policy determines which deposits are admitted. The service is the protocol's principal operational dependency, and its automation—moving from manual publication to a continuously running service—is an explicit roadmap item (Appendix J). The append-only registry ensures that every historical root, and thus the provenance of every accepted withdrawal, remains auditable after the fact.

N.3 Relayer operation

A relayer monitors for withdrawal requests, validates that a submitted proof and descriptor are well-formed and that the offered fee is acceptable, and submits the transaction. Because relaying carries no trust (Proposition 14.12), the relayer set can be permissionless: any party may relay, competition drives fees toward the cost of gas, and a user unable to reach one relayer routes to another. Decentralizing the relayer set removes the last liveness dependency in the withdrawal path.

N.4 Monitoring and auditability

Every protocol action emits an event: deposits announce commitment, label, value, and pre-commitment; withdrawals announce the spent nullifier, withdrawn value, and new commitment; association updates announce the new root and its off-chain pointer. These events are sufficient to reconstruct the entire public state—the state tree, the spent set, and the association history—from the chain alone, so that any party can independently audit the pool's integrity without privileged access. The privacy of users is preserved precisely because these public events, by design, do not link deposits to withdrawals.

O Conclusion

Public blockchains made a foundational trade: total transparency in exchange for verifiability without trust. For a system meant to carry ordinary economic life, that trade extracted too high a price in privacy, and the first attempts to buy it back did so in a way that could not coexist with the rules society places on money.

ra:st is a demonstration that the trade can be renegotiated. Zero-knowledge proofs let a person show that her money is clean without showing the money; they let her exit a shielded pool without anyone tracing where she entered. Privacy for honest users and exclusion of illicit funds turn out not to be opposites, but two consequences of the same primitive. Deposit in the open, withdraw in private, and prove your funds are clean.

The protocol holds nothing of yours. It cannot freeze you, cannot recover for you, and cannot see which note is yours. It asks you to trust a small amount of auditable code and a few well-studied assumptions, and nothing else. That is the whole design, and it is the point.

prove everything. reveal nothing.

P Acknowledgments and Attribution

ra:st is an instantiation of the Privacy Pools framework introduced by Vitalik Buterin, Jacob Illum, Matthias Nadler, Fabian Schär, and Ameen Soleimani [1], and is built directly upon the open-source Privacy Pools reference implementation developed and released by 0xbow. The cryptographic design—the commitment and nullifier structure, the dual-accumulator association mechanism, and the withdrawal relation—originates in that line of work; the contribution of the present system is its instantiation, deployment on Robinhood Chain, and the surrounding product, not the invention of new primitives. We record this attribution not merely as a courtesy but as a matter of accuracy: a specification that presented inherited cryptography as original work would misrepresent both its provenance and its assurance, since the confidence one may place in the design derives in large part from the scrutiny the underlying framework and implementation have received. We are likewise indebted to the authors of the Groth16 argument system, the Poseidon hash function, and the incremental Merkle-tree constructions on which the protocol depends, each cited in the references.

The protocol builds on cryptographic research that is the work of many; any errors in this document's presentation of it are our own.

References

[1] V. Buterin, J. Illum, M. Nadler, F. Fabian, and A. Soleimani. Blockchain Privacy and Regulatory Compliance: Towards a Practical Equilibrium. 2023.

[2] J. Groth. On the Size of Pairing-Based Non-Interactive Arguments. In *EUROCRYPT*, 2016.

[3] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In *USENIX Security*, 2021.

[4] S. Bowe, A. Gabizon, and I. Miers. Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model. 2017.

[5] R. C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In *CRYPTO*, 1987.

[6] P. S. L. M. Barreto and M. Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. In *Selected Areas in Cryptography*, 2005.

Q Document Status and Versioning

This is version 0.1 of the ra:st technical whitepaper. It specifies the protocol as designed and as deployed in its initial form on Robinhood Chain, and is intended to evolve alongside the system. Several elements are explicitly provisional and are marked as such in the text: the $RAST token parameters (Appendix I.3), which await a dedicated tokenomics specification; the automation of the association-set service and the relayer network (Appendix J); and the independent security audit and publication of the trusted-setup ceremony transcript, which should precede the protocol's scaling to significant value. Concrete deployment parameters—contract addresses, scope, fee rate, and minimum deposit—are published separately as deployment metadata so that this document remains valid across redeployments and parameter changes.

Future revisions will incorporate the finalized token specification, the results of security review, and any protocol refinements arising from operation. The cryptographic core—the commitment and nullifier scheme, the dual-accumulator association mechanism, and the withdrawal relation—is considered stable and is not expected to change; revisions are anticipated in the ecosystem and operational layers rather than the protocol's mathematical foundation. Readers should consult the latest published version for the current state of the provisional elements identified above.