

Zoivy Whitepaper

A Privacy-Native, Post-Quantum Layer 1

Sep 2026

Abstract

Public blockchains make correctness broadly verifiable by exposing transactions and state; privacy-preserving systems seek the same verifiability without requiring disclosure of the underlying data. A second design problem arises from cryptographic algorithms whose long-term security is threatened by sufficiently capable quantum computers. Zoivy proposes a research-stage Layer 1 architecture in which private-by-default value transfer, explicit public execution, transparent zero-knowledge verification and post-quantum authentication are considered as one protocol rather than independent additions. Its central candidate mechanism, Zoivy Private Spending (ZPS-1), describes a note commitment, recipient-controlled nullifier secret, hidden ML-DSA spending authorization and a STARK-verifiable state transition. This paper develops the motivation, primitives, algorithms, adversarial games, hybrid execution semantics, consensus requirements and cross-domain migration constraints of that proposal. The proposal distinguishes a minimally checkable transfer profile from a stronger recipient-delivery profile, derives state-machine safety conditions and states counterexamples that would falsify the privacy claim. No working native network or proven cryptographic construction exists; proving circuits, security parameters and independent analysis are future work.

Introduction

The problem of publicly verifiable but publicly observable state

A blockchain can allow strangers to agree on the validity and ordering of transactions without delegating final authority to one operator. In common account-based ledgers, however, this transparency exposes addresses, balances, transfer amounts and durable transaction relationships. Pseudonymous accounts are not equivalent to anonymous users: public flows can be correlated with exchange records, network traffic and application activity. Auditability is valuable, but revealing every payment is not necessary for proving that a payment obeys the rules of a ledger. Privacy and public verifiability should therefore be treated as separate properties.

Zero-knowledge protocols provide a way to prove a statement about a private witness while withholding that witness. In a note-based payment system, a prover may establish that an unspent commitment exists, that the prover is entitled to consume it and that the transaction preserves value, without revealing the input note's amount or spending key. The public network still needs enough information to prevent double spending, price execution, account for fees and finalize a common state. A privacy layer is thus not an opaque ledger: it is a carefully selected **public leakage function** coupled to a privately proven transition [5,7,11].

There is also a practical reason not to make all computation opaque. Transparent contracts can expose market prices, validator sets, public governance outcomes and state needed for open coordination. Zoivy consequently investigates one ledger with public and private execution rather than two incompatible chains. Private value transfer is the intended default where supported, while explicit conversions and public application calls remain possible. The conversion boundary is itself a privacy risk and must be described, not hidden behind the word *hybrid*.

The problem of cryptographic longevity

Widely used signature and key-agreement systems built on RSA or elliptic-curve discrete logarithms are vulnerable in principle to a sufficiently powerful fault-tolerant quantum computer. The arrival date of that capability is uncertain. Nevertheless, long-lived signatures and ciphertexts motivate migration planning, including the possibility that encrypted data collected now could be attacked later. NIST’s ML-KEM, ML-DSA and SLH-DSA standards supply important primitive choices, but none certifies a complete blockchain or a composed private-payment protocol [1–4].

A transparent STARK can reduce dependence on pairing-based proof systems. It does not by itself secure wallet ownership, network authentication, encryption, recovery or consensus. The security goal pursued by Zoicy is **post-quantum security at every mandatory trust boundary**, conditional on explicit algorithms, parameters and implementation evidence. The initial token intended for Base does not inherit this future property.

Proposed contribution and scope

Consider a payer who knows everything they put into a confidential output: its amount, randomness, public recipient descriptor and publication time. If that payer can also predict the *recipient’s future public spend marker*, a perfectly hiding amount and a perfectly zero-knowledge proof will not prevent linking payment to subsequent expenditure. Now suppose the payer cannot predict that marker but can silently replace the ciphertext with one the recipient cannot recover. Transfer privacy and *usable delivery* are distinct properties. The technical problem investigated here is whether a protocol can retain recipient-controlled spend privacy while binding its public proof, encrypted output and subsequent ledger transition to one unambiguous authorization transcript.

The specific proposed artifact is **Zoicy Private Spending (ZPS-1)**. Its distinguishing research hypothesis has four parts. First, a recipient supplies a fresh, authenticated one-use descriptor containing an independent *commitment* to a nullifier secret rather than sharing the secret with the payer. Second, the note commitment also binds the selected descriptor, including the note-encryption public key. Third, a hidden ML-DSA verifier key and a complete transcript digest are checked within one candidate STARK validity relation, without publishing the verifier key in the transaction. Fourth, note and nullifier updates settle atomically with explicitly queued public effects under an independently authenticated BFT consensus. The constituent primitives already exist; the proposed *interface*, *binding rules* and *composition obligations* are the objects of research, not a claimed new signature or proof system.

This is neither a claim of first invention nor evidence of superiority over established systems. It is a falsifiable proposal with potential counterexamples. A transcript that validates after ciphertext substitution refutes its binding rule. A payer who can infer the future nullifier refutes its sender-unlinkability target. A finalized double spend refutes its state-transition invariant. A recipient who cannot reconstruct an accepted output demonstrates that baseline envelope binding is insufficient for delivery. Those cases will be carried through the formal and algorithmic description rather than hidden under general claims about privacy.

The paper proceeds from security goals and encodings to a worked note, candidate algorithms and invariant arguments, then considers hybrid execution, consensus and cross-chain migration. Mathematical assertions are either definitions, conditional state-machine implications or stated proof obligations; unproved quantum-cryptographic claims are not presented as theorems. Neither a production ZPS-1 prover nor a native Zoicy blockchain currently exists.

Preliminaries and Security Requirements

Actors, state and adversaries

Let a *client* control its spending secrets and local private state; a *prover* construct a proof from private witnesses; a *validator* verify public transaction data and proposed state transitions; and a *consensus participant* authenticate a vote over an ordered block. These roles may share a device but possess different authority. A client need not entrust a validator with note secrets. A network observer can

see commitments, nullifiers, ciphertext sizes, submitted proofs, timing and network traffic even when the payment contents are secret.

The adversary model includes adaptive clients, colluding transaction submitters, malicious RPC endpoints, dishonest validators, compromised wallets, timing analysts and a quantum-capable cryptographic attacker. Their powers must be bounded in a final game: key-corruption timing, oracle access, stake control, message scheduling, historic root access and quantum queries cannot simply be assumed harmless. Physical coercion, malware on the client’s computer and universal invisibility to network observers are outside the guarantees of a zero-knowledge proof.

The fundamental goals are: (i) *authorization*—no successful spend without an authorized witness; (ii) *conservation*—no creation of unbacked units; (iii) *uniqueness*—a consumed note cannot be accepted twice; (iv) *conditional confidentiality*—transactions with identical permitted leakage should be computationally difficult to distinguish; (v) *finality*—honest validators do not finalize conflicting states within the specified Byzantine bound; and (vi) *availability*—authorized recipients can discover the encrypted notes they need to spend. Some of these goals are cryptographic, some concern consensus and others depend on network service and user behavior.

From security goals to falsifiable mechanisms

These goals are deliberately separate because a system may satisfy one and fail another. Consider a prover that proves membership and conservation but does not bind the authorization message to its outputs: the proof could be valid while the sender silently changes the recipient. Consider a confidential note that publishes a deterministic owner-derived identifier: it may conserve value but be publicly linkable across uses. Consider a signature-valid transfer whose output ciphertext is never retrievable: consensus may finalize it while the intended owner cannot reconstruct their spendable state. A credible design has to specify and then test all three interfaces.

Three kinds of objects are therefore distinguished in the remainder of the paper: an *unfinalized proposal* made by a client, a *cryptographically verified transition* checked against a candidate state, and a *finalized transition* bound to consensus history. Calling each of them a “transaction” is convenient, but conflating them leads to false claims about double-spend prevention and recipient confirmation. The ZPS-1 verifier accepts a proof relative to specified roots; only the ordered state machine can declare a nullifier spent, and only finalized consensus makes that state canonical.

Encodings and proof interfaces

For a reproducible research profile, define **U64** as an eight-byte unsigned big-endian integer, **U128** as sixteen such bytes, **B32** as an exact 32-byte string and **LP(x)** as a four-byte big-endian length followed by the bytes of **x**. Ordered vectors **MUST** encode their element count and every element length. Amounts are integers, not floating-point values; implicit strings, noncanonical maps, duplicate fields and ambiguous concatenation are rejected. The prior ZPS-1 test profile used the literal transcript prefix **Z0IY-ZPS1x00** with versioned, domain-separated SHAKE256-derived outputs. SHAKE256 in a test fixture is not a final STARK-friendly commitment specification or a security endorsement [18,20].

Write a domain-separated digest as

$$H_d(x_1, \dots, x_k) = \text{XOF}(\text{Enc}(d, k, x_1, \dots, x_k)), \quad (1)$$

where d fixes the purpose and version, and **Enc** is canonical and injective over the permitted input types. Distinct domains are required for commitments, nullifiers, signatures and verification statements. A final implementation must choose output lengths, algebraic hash functions where appropriate, cryptographic parameters and an exact version-upgrade procedure. The equation specifies the interface, not the concrete hash approved for mainnet.

A transaction’s public input is denoted x ; its secret witness is w ; and its proof is π . The abstract prover and verifier are

$$\pi \leftarrow \text{Prove}(\text{vkID}, x, w), \quad \text{Verify}(\text{vkID}, \text{Enc}(x), \pi) \in \{0, 1\}. \quad (2)$$

The verifier identifier vkID is bound to the consensus protocol version. A change in the circuit, hash or verifier must be an explicit protocol upgrade, never a silent reinterpretation of earlier statements.

Design criteria and failure witnesses

The minimum target is a deterministic **public accept/reject predicate** on canonical bytes, a private proving relation for legitimate spends, and a precisely scoped statement of what an observer learns. The candidate is acceptable for further engineering only if independently written validators agree on every valid and malformed transcript, only authorized witnesses can satisfy the spending relation, and finalized state does not accept repeated nullifiers or units minted outside permitted transitions. These objectives are deliberately separated from throughput, which cannot be claimed without an actual prover and network.

One can make this criterion concrete using four hypothetical traces. In trace **A**, an honest receiver generates a one-use descriptor and successfully decrypts a valid output; the private spend reveals only declared leakage. In **B**, the payer changes the ciphertext after the authorization digest is formed; validation must reject. In **C**, a payer publishes a self-consistent but undecryptable ciphertext; a *baseline* protocol can accept this, but a future *strict-delivery* variant must reject or require a separately authenticated recipient acknowledgement. In **D**, two blocks independently include the same nullifier; no valid finalized history may contain both. These are design examples and proposed test cases, **not experimental results**.

Research assumptions and their interaction

The following formalization is intentionally minimal. A protocol whose commitment hash is secure but whose encoding admits two different parses can fail binding at the serialization boundary. A protocol with secure signatures but no canonical chain identifier can admit replay. A proof with an unbounded historic-root policy may allow resource exhaustion or a stale state assumption, even if every Merkle path is valid. These are not flaws in STARK theory; they are composition failures introduced by the surrounding blockchain.

For that reason the research profile assumes canonical serialization, explicit algorithm IDs, separate key purposes and a versioned verifier mapping. The chain must reject unknown versions rather than attempting heuristic compatibility. The cost of this conservatism is explicit upgrades and sometimes disruption. Its benefit is that independent validator implementations can agree on a deterministic accept/reject function.

Network Architecture and State Transition Model

Zoiy separates client-side private execution, public deterministic contract execution, proof/state verification, and consensus finality. The separation is functional, not a promise that every user device will provide strong hardware isolation. The chain is intended to finalize one root of truth for both public and private effects, while limiting the private witness to the originating client's environment.

Let R_t^{note} denote a Merkle commitment to created notes, $\mathcal{N}_t$ the set of spent nullifiers, P_t public contract state and V_t the authenticated validator-set commitment. A complete implementation also maintains a uniquely indexed set $\mathcal{C}_t$ of note commitments and the published encrypted-envelope data or its availability commitment. For compactness the four-component summary state below omits those derived storage objects; it must **not** be read as a complete genesis or block serialization. Define the consensus-visible summary state

$$S_t = (R_t^{\text{note}}, \mathcal{N}_t, P_t, V_t). \quad (3)$$

A submitted transaction is not itself final. Its deterministic validation and execution must produce either an entirely checked state transition or a rejection:

$$\delta(S_t, T) = \begin{cases} S_{t+1}, & \text{if all validity and resource predicates hold,} \\ \perp, & \text{otherwise.} \end{cases} \quad (4)$$

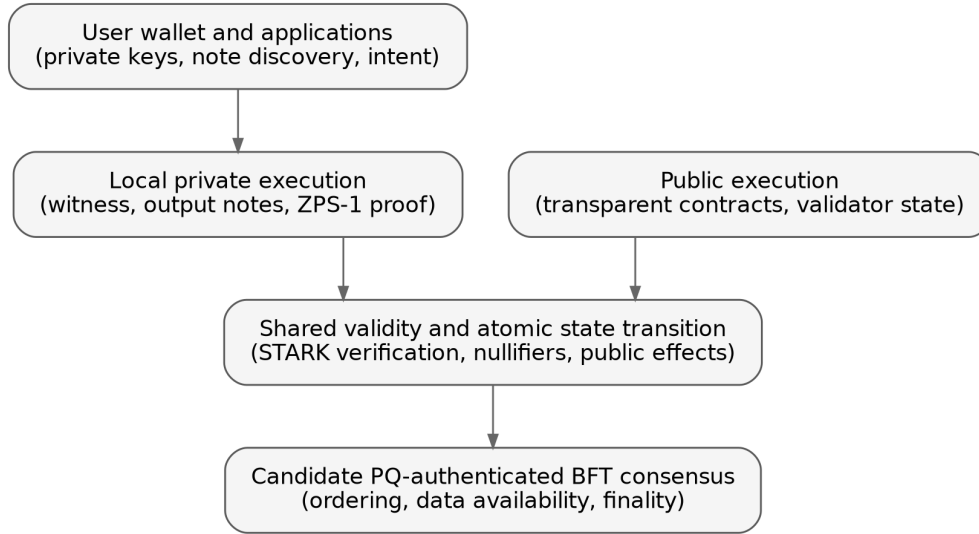


Figure 1: Proposed Zoicy execution architecture. Private witnesses remain client-side; validators verify proofs and public effects before candidate state reaches finality.

Here $\perp$ indicates rejection without committed *application* effects. The treatment of fees consumed during execution is a separately specified policy. A state root certifies a commitment to data, not that each client has retrieved its encrypted notes; availability and proof correctness are different obligations.

The interface between client and validator

A client obtains authenticated note inclusion paths, reconstructs owned notes, selects inputs and prospective outputs, and prepares a proof. The network receives commitments to new notes, spent-nullifier values, versioned proof data, an explicit public-call descriptor where relevant and encrypted recipient envelopes. Validators do not need the secret amounts, signing keys, nullifier secrets or raw private witness. However, they must recheck root validity, double-spend status, transaction age, verifier identity and public authorization against the state in the actual block order. This separation is similar in motivation, but not identical in mechanics, to the local private execution models studied by Aztec and Miden [7,8,13].

Post-Quantum Cryptographic Substrate

Authentication, confidentiality and proof verification

Post-quantum design requires more than replacing ECDSA with a lattice-based signature. Zoicy’s initial candidate suite associates distinct tasks with distinct primitives: ML-DSA for spend authorization and validator authentication, ML-KEM for note-encryption key establishment, a versioned KDF and authenticated encryption for note payloads, and a transparent STARK family for computational integrity. SLH-DSA may be investigated as a hash-based alternative for specific key-management settings. These are **candidate assignments**, not finalized parameters [1–4].

The wallet must derive or store *separate* secrets for spending authorization, nullification, viewing and recovery. Reusing a standard signature secret as an undocumented nullifier PRF key would require a security argument absent from the standard. Similarly, KEM encapsulation does not automatically hide recipient key registration or guarantee sender authentication. System security depends on the entire transcript, key lifecycle and failure behavior.

Consequences of choosing post-quantum primitives

The role of an algorithm matters as much as its name. ML-DSA authenticates a message; it does not encrypt a note. ML-KEM establishes shared key material; it does not authenticate the sender by itself.

A hash commitment can bind a note to an amount while leaking that amount if the committed value has a tiny domain or if note randomness is reused. These familiar separations become decisive when hidden authorization and on-chain validity proofs are composed.

There is a further tension between **standardized native verification** and **efficient proof generation**. A general verifier can check an ML-DSA signature using conventional software, but a zero-knowledge prover must express that same operation in its chosen arithmetic language. If that circuit is costly, simply moving the public signature into a private witness is not an inexpensive privacy upgrade. Zoicy must choose whether the final circuit verifies ML-DSA directly, uses a carefully reviewed alternative proof of authorization, or changes the ownership construction. Until a concrete circuit and reductions exist, none of those options should be described as proven superior.

Why a STARK, and what it does not establish

In a STARK-style argument, execution is represented by an algebraic trace $\mathbf{T}$ over a chosen field. Valid traces satisfy transition, boundary and permutation constraints encoded in an algebraic intermediate representation (AIR), schematically

$$P_j(\mathbf{T}_i, \mathbf{T}_{i+1}, \dots) = 0 \quad \text{for each applicable transition } (i, j). \quad (5)$$

The prover commits to trace-related data; a low-degree test such as FRI supports efficient public checking without revealing every computational step [5,6]. Actual proof size, verifier work and soundness require a concrete field, hash, trace layout, query count, code rate, Fiat–Shamir transform, masking method and recursion design. In particular, **transparent** does not mean **zero-knowledge** without witness-hiding, and hash-based foundations do not automatically provide a specified number of quantum-security bits.

A useful target statement is

$$\Pr[\text{Verify}(\text{vkID}, x, \pi) = 1 \wedge x \notin \mathcal{L}_{\text{ZPS1}}] \leq \varepsilon_{\text{sound}}(\lambda), \quad (6)$$

where the probability is taken in a declared attack experiment and λ is the security parameter. This is a **proof obligation**, not an established result for Zoicy. In-circuit ML-DSA verification is especially important: lattice arithmetic, sampling-related checks and signature encoding may induce a costly execution trace. The proposed design must be tested against a hash-based authorization alternative without assuming in advance that one is faster or safer.

Zoicy Private Spending: The ZPS-1 Candidate

ZPS-1 as a constrained spending language

The spending relation is intentionally narrower than a general-purpose confidential virtual machine. Its first objective is to recognize authorized consumption of immutable note objects and creation of equally well-formed outputs. Each note acts like a concealed unspent output with a committed policy. A valid spend cannot alter an already published note or spend against a fabricated root. Instead, it consumes the right to spend exactly once, signaled by a public nullifier, and creates fresh commitments. This is why note immutability and nullifier uniqueness are complementary: the former anchors what was created, the latter prevents its reuse.

The use of a *one-use descriptor* has a privacy rationale beyond key hygiene. If the same descriptor appears in multiple notes, ciphertext tags, commitments or proofs may accidentally expose a relationship even though amounts stay hidden. The descriptor’s binding to an authenticated recipient identity is also subtle: an unauthenticated network lookup lets an active attacker substitute their own key and redirect funds. A privacy-preserving registry must reconcile authentication with minimizing durable identity linkage. ZPS-1 describes the interface needed from such a service; it does not currently design that entire service.

The ownership problem and its information boundary

Suppose Alice sends a private note to Bob. The chain should verify that Bob, not Alice or a subsequent observer, can spend the note, while neither the note amount nor Bob’s reusable ownership key is revealed publicly. Later, the ledger must notice a double spend without publishing which original commitment was consumed. These requirements introduce two distinct secrets: a *spending key* establishing authority and a *nullifier secret* yielding a public, one-time spend marker.

The critical design choice is that **Bob supplies the nullifier-secret commitment**, while Alice receives only the commitment and Bob’s one-use spending verification key. If Alice learned the underlying nullifier secret, she could calculate Bob’s later public nullifier and correlate his spend to the payment she created. This would defeat an important privacy goal even if the STARK proof were otherwise zero-knowledge. Descriptor authenticity, one-time-use enforcement and anonymous delivery remain independent design problems; the construction cannot assume an unauthenticated descriptor is safe.

One-use descriptor and note construction

Bob generates an ML-DSA spending keypair (sk_{sp}, pk_{sp}) , an independent random nullifier secret s_{nf} and an encryption key/address pair. Let

$$\nu = H_{\text{bind}}(s_{nf}). \quad (7)$$

Bob publishes or communicates an authenticated one-use descriptor $D = (pk_{sp}, \nu, pk_{\text{view}}, \text{id})$ without revealing the secret keys or s_{nf} . Here pk_{view} is an ML-KEM encapsulation public key, separate from the ML-DSA spending key. Its descriptor identity and intended one-use status require an authenticated distribution mechanism; publication alone does not establish either. Alice samples a fresh note identifier ρ and blinding randomness r . Let $d = H_{\text{desc}}(\text{Enc}(D))$ be the canonical digest of **the entire authenticated descriptor**, including the encryption key and policy-relevant identifiers. For asset identifier a , integer value v and policy digest p , she computes

$$C = H_{\text{note}}(\text{chainID}, \text{version}, a, v, \rho, r, pk_{sp}, \nu, d, p). \quad (8)$$

Binding the descriptor digest d matters: without it, a proof about a note and a ciphertext hash can fail to establish **which recipient encryption key was authorized**. This explicit field strengthens the candidate statement; it does not authenticate the initial distribution of D . The hash must commit to all fields through canonical encoding. Hiding requires sufficient entropy and an appropriate commitment construction; binding requires collision or second-preimage assumptions in the relevant quantum model. Neither property follows merely because the expression contains a hash symbol.

The public state stores C , not Bob’s amount or key. A ciphertext envelope delivers the preimage needed by Bob’s wallet to discover and later spend the note. Alice may know some preimage components because she created the output; the independent Bob-generated s_{nf} must never be included in what Alice knows or encrypts on Bob’s behalf.

Encrypted recipient delivery

Alice encapsulates to Bob’s public viewing key, derives a context-bound symmetric key and encrypts an authenticated note payload:

$$(ct_K, k) \leftarrow \text{MLKEM.Encaps}(pk_{\text{view}}), \quad K = \text{KDF}(k, \text{context}), \quad (9)$$

$$ct_N \leftarrow \text{AEAD.Enc}(K, \text{nonce}, \text{notePayload}, ad). \quad (10)$$

The associated data ad binds the ciphertext to the chain, protocol version, note commitment, the recipient’s one-use descriptor identifier and envelope type. Bob decapsulates, decrypts, checks the

recovered commitment against C and stores the note together with his independently maintained nullifier secret. The envelope is given a canonical hash $E = H_{\text{env}}(\text{Enc}(ct_K, \text{nonce}, ct_N, ad))$ which enters the transaction-wide authorization digest. Thus, after authorization, replacing ciphertext changes E and invalidates authorization; this protects *integrity of the published envelope*, not the sender’s honest construction of that envelope.

A malicious sender can still encrypt unusable data or omit the ciphertext. Three distinguishable objectives are therefore needed: *integrity* (published bytes hash to the authorized E), *availability* (these bytes can be retrieved by an authorized recipient), and *recoverability* (decrypting them under the descriptor’s view secret yields an opening consistent with C). The baseline ZPS-1 relation enforces integrity, and block data-availability rules may enforce byte availability; neither proves recoverability. A recipient acknowledgement signed under an authenticated key is an alternative **interactive** policy, but it proves only that an authorized participant attested to receipt, not that their wallet will retain the key or that decryption is cryptographically correct. These are not equivalent safeguards.

A stronger **strict-delivery candidate** would bind d , C , E and a private encryption witness and require the prover to establish the existence of ML-KEM encapsulation coins r_K , shared key k and note-opening payload p_N such that:

$$\begin{aligned} (ct_K, k) &= \text{MLKEM.Encaps}(\text{pk}_{\text{view}}; r_K), \\ ct_N &= \text{AEAD.Enc}(\text{KDF}(k, \text{context}), \text{nonce}, p_N, ad), \\ H_{\text{note}}(p_N, d) &= C, \quad H_{\text{env}}(\text{Enc}(ct_K, \text{nonce}, ct_N, ad)) = E. \end{aligned}$$

Here $H_{\text{note}}(p_N, d)$ is schematic shorthand for **the same canonical note fields and hash** used in (8), not an independently chosen hash format. The witness relation must additionally constrain the authenticated descriptor and encryption-key membership, uniqueness of nonces, all ciphertext envelope fields and the correct randomized encapsulation semantics. Neither the relation’s compiled STARK cost nor a compositional security argument has been established. The current baseline **does not** implement or require this proposed extension. Even if strict encryption correctness were proved, independent byte availability, recipient key retention and network unlinkability would remain separate problems [1,4,21].

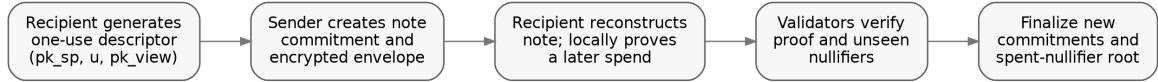


Figure 2: Private-note lifecycle. Recipient-controlled nullifier material is not disclosed to the sender. The published spend marker is checked by validators independently of proof validity.

A worked symbolic payment

Consider a payment where an earlier commitment C_0 encodes a secret amount v_0 and the owner wishes to create two new commitments C_1, C_2 representing v_1, v_2 with fee f . The prover knows the opening of C_0 , a Merkle authentication path to the permitted root, the spending authorization witness and the recipient-held nullifier secret associated with C_0 . The public verifier sees R , one candidate nullifier N_0 , the two new commitments, a fee and a proof. It verifies neither the cleartext amount nor the owner key as separate public values.

What prevents arbitrary creation of value is not the nullifier itself. The proof relation must simultaneously establish the authenticated opening of C_0 , the correct creation of C_1, C_2 and the exact integer equation $v_0 = v_1 + v_2 + f$. What prevents a second consumption of the same input is not the conservation equation. The chain’s state machine must reject any later use of N_0 . Finally, what prevents the sender from changing C_2 after authorization is not either of those checks: the complete transaction transcript must be signed and bound into the proof. These independent predicates explain the compound relation below.

Nullifier and transaction authorization

The recipient’s nullifier secret and the note-specific nonce define the public marker

$$N = H_{\text{null}}(\text{chainID}, \text{version}, s_{\text{nf}}, \rho). \quad (11)$$

The proof must establish both $\nu = H_{\text{bind}}(s_{\text{nf}})$ and the equation above using the **same witness**. This binds the public nullifier to the private note without publishing its originating commitment. The chain additionally checks that the public N is absent from previously spent markers. This relationship depends on domain separation, secret entropy, collision resistance and a reviewed zero-knowledge construction.

A spend authorizes the *complete* transaction, not a partial list of outputs. Let $\mathbf{N}$ be the canonical ordered list of input nullifiers, $\mathbf{C}$ the ordered output commitments, $\mathbf{E}$ the ordered envelope hashes (including KEM ciphertexts and AEAD envelopes), Q the queued public-call descriptor, f the fee and e the expiration. Each output commitment is paired with exactly one envelope hash, and the canonical encoding fixes the pairing. Without $\mathbf{E}$ an attacker could alter delivery data while leaving the signed note commitments unchanged. With accepted anchor root R , the message digest is

$$m = H_{\text{auth}}(\text{chainID}, \text{version}, R, \mathbf{N}, \mathbf{C}, \mathbf{E}, Q, f, e). \quad (12)$$

For each input, its rightful owner produces $\sigma \leftarrow \text{MLDSA}.\text{Sign}(\text{sk}_{\text{sp}}, m)$. The note’s hidden public key verifies the signature inside the ZPS-1 relation:

$$\text{MLDSA}.\text{Verify}(\text{pk}_{\text{sp}}, m, \sigma) = 1. \quad (13)$$

The chain need not learn pk_{sp} or σ as separate public data. A valid proof must nevertheless show that the key used in (13) is exactly the key committed in the spent note. The practical size of this signature-verification subcircuit remains unknown.

The public statement, private witness and proving relation

Define x to contain chain/version and verifier identifiers, anchor root, sorted nullifiers, output commitments, the correspondingly ordered envelope hashes, fee, public-call hash and expiration. The private witness w contains the input note preimages, their Merkle authentication paths, each nullifier secret, signing keys’ corresponding verification material and signatures, plus the output note preimages and any application-specific secret data. The public validator does **not** accept claims about value or ownership merely because a client asserts them; the proof must constrain them.

The ZPS-1 validity relation is the conjunction of seven independently checkable predicates. With every predicate evaluated on the same public statement x and private witness w , abbreviate membership by M , note binding by B , envelope commitment binding by E , post-quantum authorization by A , nullifier correctness by N , bounded integer arithmetic by I , and per-asset conservation by V . The candidate language is then

$$\mathcal{R}_{\text{ZPS1}}(x, w) = M \wedge B \wedge E \wedge A \wedge N \wedge I \wedge V. \quad (14)$$

These letters stand for explicit relations, not interchangeable assumptions. M authenticates existing commitments to an accepted Merkle root; B recomputes input and output note commitments **including the matching descriptor digest** d ; E binds each output to its canonically hashed ciphertext envelope; A checks the ML-DSA signature over the entire authorized transcript; N recomputes each nullifier from the private note opening; I enforces unsigned integer bounds; and V prevents value creation. In particular, E does **not** yet establish that a ciphertext decrypts correctly for its intended receiver. Each predicate must eventually compile to explicit constraints in a pinned STARK verifier. Equation (14) defines a **candidate language**, not a completed AIR implementation or a security theorem.

For a private-only single-asset transaction with k input notes and ℓ outputs, conservation is

$$\sum_{i=1}^k v_i = \sum_{j=1}^{\ell} v'_j + f, \quad 0 \leq v_i, v'_j, f < 2^{128}. \quad (15)$$

The sums **MUST** be checked as integers with appropriately wide accumulators: equality in a finite proof field alone can permit wraparound. For multi-asset or public/private conversions, this equation must be indexed per asset and include the separately validated deposits, withdrawals and fee denomination.

What constitutes a composition claim?

The primary comparison is between three *conceptual* policies, not between deployed benchmarks: an ordinary shielded note with a publicly specified spend relation; the ZPS-1 baseline with recipient-controlled nullifier material, hidden PQ signature authorization and bound ciphertext envelopes; and the proposed strict-delivery extension. The baseline adds transcript and authorization obligations relative to a generic shielded note. The strict candidate would add an **encryption-correctness** witness relation, but at a potentially substantial proving cost. It would be misleading to score these as performance improvements before the circuits and threat games are instantiated.

The sender-unlinkability rationale is similarly conditional. If Alice only knows $\nu = H_{\text{bind}}(s_{\text{nf}})$, the note nonce ρ and public transcript, then computing $N = H_{\text{null}}(\text{chainID}, \text{version}, s_{\text{nf}}, \rho)$ requires recovery of s_{nf} or an unexpected cross-domain relation. This is an *informal reduction intuition*, not a theorem: descriptor reuse, timing, receiver corruption, poor secret generation and quantum oracle access require separately defined experiments. More importantly, it does **not** establish the privacy of a receiver's network address or application activity.

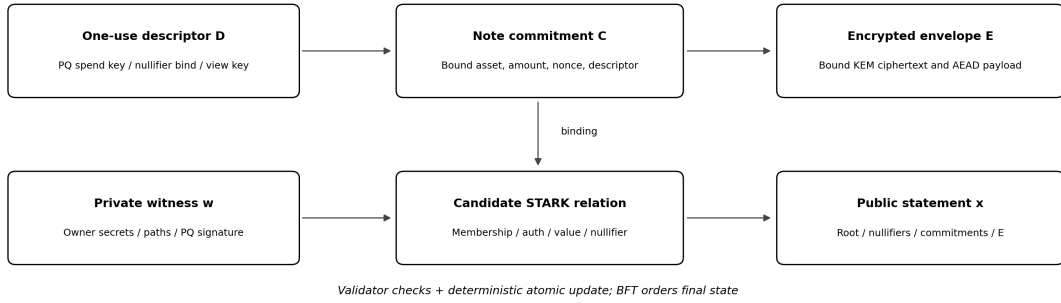


Figure 3: Candidate ZPS-1 transcript dependency diagram. A bound descriptor and ciphertext envelope do not themselves prove recipient decryptability; the verifier checks only the constraints included in its registered proof relation.

Candidate algorithms and rejection behavior

The following pseudocode gives the intended ordering of ZPS-1 operations. It is descriptive specification pseudocode, not executable cryptography.

Algorithm 1 — CreateNote (sender with recipient descriptor)

Input: authenticated one-use D, asset a, amount v, policy p
 Require: valid D; unused descriptor; checked integer v
 Sample: fresh rho and r from a cryptographic RNG
 Compute: d = H_desc(canonical D)
 Compute: C = H_note(chain, version, a, v, rho, r, D.pk_sp, D.nu, d, p)
 Build: ciphertext envelope (ct_K, nonce, ct_N, ad) bound to C and D
 Compute: E = H_env(canonical envelope bytes)
 Output: public (C, E), full envelope bytes, recovery data for recipient
 Caveat: without a delivery-consistency proof this does not certify decryptability

Algorithm 2 — SpendNote (holder's private execution)

Input: owned notes, signed intent, accepted state root, fee, expiry
 Recover and verify input preimages and Merkle paths
 Derive each N_i using recipient-held s_nf_i and rho_i
 Sort N_i in canonical order; reject duplicates
 Create output commitments and encrypted envelopes; compute E_j for each
 Construct x binding (C_j, E_j) pairs, fees, public calls, chain and root
 For each input: ML-DSA-sign the same complete authorization digest
 Check note arithmetic; construct witness w
 Generate pi = STARK.Prove(vkID, x, w)
 Output: transaction (x, pi, ciphertext envelopes)

Algorithm 3 — VerifySpend and ApplyBlock (validator)

For each transaction in the exact proposed block order:
 REJECT malformed bytes, unknown version or wrong chain identifier
 REJECT stale or nonfinal anchor, expired transaction, wrong vkID
 REJECT unsorted or duplicate N values, within transaction or block
 REJECT N already present in the current scratch-state spent set
 REJECT duplicate output commitments in this tx OR existing chain state
 REJECT inconsistent (C_j, E_j) pairs or malformed descriptor binding
 REJECT if supplied envelope bytes do not hash to their bound E_j
 REQUIRE agreed availability of those exact envelope bytes (baseline policy)
 NOTE: baseline verification does not prove receiver decryptability
 REJECT if STARK.Verify(vkID, x, pi) is false
 Simulate private and queued public application effects in scratch state
 REJECT the transaction if any public application predicate reverts
 Only if all effects succeed, stage nullifiers, notes, transfers and fee
 After all transactions, compare computed roots with block header
 COMMIT all staged effects only if every transaction and root is valid
 Otherwise REJECT the block and do not change canonical state

For finalized chain history $\mathcal{T}$, the required one-spend property is

$$\forall T_i \neq T_j \in \mathcal{T} : \mathcal{N}(T_i) \cap \mathcal{N}(T_j) = \emptyset. \quad (16)$$

Mempool admission is not finality. Two conflicting transactions may appear valid against the same previously finalized state; only canonical block ordering determines which, if either, can be committed. The history-wide output-commitment uniqueness check is an additional deterministic state invariant. Test fixtures may choose a 256-root anchor window and capped input/output counts, but these are **test parameters**, not immutable consensus decisions.

Proposition 1 (conditional one-spend safety). Assume a canonical initial nullifier set, deterministic sequential processing of finalized blocks, and a validator rule that tests every proposed nullifier against the current scratch-state spent set and inserts it only upon atomic transaction acceptance. Then no two transactions in one finalized chain history can consume the same nullifier. *Reason:* by

induction on accepted transactions, every previously committed nullifier is present in the next scratch-state set; a repeated value fails the membership rejection rule. This is a state-machine argument. It does not prove that an attacker cannot fabricate a *different* nullifier for the same note; that separate property depends on the nullifier relation, binding, proof soundness and cryptographic assumptions.

Proposition 2 (conditional asset conservation). Assume every private input and output opening is bound to its authentic commitment, the private proof enforces (15) using range-checked integer arithmetic without field wraparound, and public conversions/fees are independently checked under (20). Then each accepted transaction preserves the declared per-asset balance accounting and the composition of accepted transitions preserves that accounting. *Reason:* sum the per-transaction equalities; all internally transferred values cancel and only authenticated external deposits, withdrawals and fees remain. This is an algebraic invariant conditional on correct soundness and state-transition validation, not a proof of the full PQ protocol or of token economic sustainability.

Confidentiality and Adversarial Security

Concrete counterexamples for privacy claims

A same-leakage game rules out easy but irrelevant wins where one challenge transaction includes three outputs and the other includes one. Yet it also exposes genuine privacy failures. If an attacker can create an output and already knows its future nullifier secret, the attacker can link the later nullifier to the output without breaking a hash. If ciphertext sizes vary by asset type or policy, those sizes may reveal information about the private witness. If a wallet obtains Merkle paths from one RPC endpoint only when it receives funds, request timing can fingerprint its note discovery. A correct privacy assessment therefore defines both cryptographic observations and the network trace allowed to the adversary.

The same reasoning applies to *selective disclosure*. Viewing keys are not a universal reversible anonymity switch. They give selected parties the ability to inspect data associated with their scope, potentially for accounting or compliance, but cannot cause observers to unlearn already shared records. A native protocol should avoid a universal plaintext-access credential and make viewing capabilities narrow, auditable at the application level and distinct from spending authority. The exact delegation and revocation format is a future specification, not an implicit consequence of encrypted notes. In particular, neither an ML-KEM ciphertext nor an authenticated data field alone provides a proof that a malicious sender used the right plaintext; this is why recipient recoverability requires a separately specified proof or acknowledgement rule.

Privacy is conditional on what is revealed

A commitment/nullifier scheme can hide the original spent note while still leaking timing, ciphertext lengths, public calls, transaction shape and fees. Thus sender–recipient unlinkability cannot be asserted without describing *which observations the adversary receives*. Let $L(T)$ be a public leakage function that includes all agreed chain-visible fields; additional network leakage must be specified for a network-level experiment. Two challenge transactions are eligible for comparison only when they have equal permitted leakage.

In a private-transfer game, the adversary proposes two eligible worlds T_0, T_1 ; the challenger samples $b \leftarrow \{0, 1\}$ and returns the public view of T_b ; the adversary guesses b' . Its distinguishing advantage is

$$\text{Adv}_{\mathcal{A}}^{\text{priv}}(\lambda) = \left| \Pr[b' = b] - \frac{1}{2} \right|. \quad (17)$$

The goal is a negligible advantage under a complete quantum-aware construction and declared corruption/leakage powers. The adversary must choose two *valid* challenge worlds with exactly the same leakage, may not trivially request the challenge owner’s signing or nullifier secrets during the challenge, and the game’s corruption and post-challenge query rules must be fixed in advance. Ciphertext lengths, envelope hashes, public-call data and confirmation timing all belong to the declared leakage unless an additional construction hides them. No bound has been derived for ZPS-1. Reusable descriptors, observable proof-submission metadata or public/private conversion amounts may make two worlds distinguishable even if the zero-knowledge proof is cryptographically perfect.

A counterexample-driven privacy argument

Case 1: payer-chosen nullifier secret. If Alice generates s_{nf} , she can calculate N later and match a finalized spend to her earlier output without attacking STARKs or recovering a key. This motivates receiver-supplied material rather than claiming a novel hash function.

Case 2: payload substitution. If authorization covers output commitments but excludes envelope hashes, an intermediary may change the ciphertext while preserving the note commitments and proof. ZPS-1’s message digest (12) includes the ordered envelope hash list and the binding predicate checks the same list. This blocks substitution **after** the authorized digest is signed, assuming canonical bytes and unforgeability; it does not prevent an intentionally malformed envelope **before** signing.

Case 3: contract revert after private proof. A private proof may be valid while a queued public swap fails due to intervening public state. If private nullifiers are committed despite that failure, a

user can lose spendability without receiving the expected public effect. ZPS-1 therefore proposes a fail-closed atomic state transition and leaves fee-on-revert as a separate unselected design.

Case 4: same-looking transactions with different public traces. A one-output payment, a three-output payment and a transparent withdrawal are not required to be indistinguishable. A privacy theorem must compare challenge worlds with the same allowed leakage and must specify whether timing and network observations are included. These counterexamples show how to falsify overbroad privacy language, not that the proposed construction has passed an adversarial analysis.

Soundness and unauthorized spending

The corresponding soundness experiment allows an adversary to submit an allegedly valid statement x and proof π . A win requires acceptance even though no authorized, value-conserving witness exists for x . Soundness is the target inequality in (6), but the proof obligation must ultimately account for ML-DSA unforgeability, commitment binding, nullifier secrecy, integer constraints, proof-system security and the validator’s implementation. A signature standard cannot substitute for a theorem about this composition.

A complete authorization game samples honest notes, answers precisely scoped issuance and spend queries, and lets an attacker propose a finalized unauthorized spend. The win condition excludes spends authorized through corrupted or legitimately delegated keys; otherwise *key theft* would be confused with a broken signature system. A meaningful reduction would have to show that a win implies a forgery, broken commitment binding, a failed nullifier rule or a false proof under the **same quantum adversary model**, while accounting for dependencies among these events. This conditional route to a theorem is not itself a theorem or quantitative security bound [2,5,21].

A full unauthorized-spend experiment also distinguishes stolen wallet keys (outside cryptographic authorization guarantees), sender knowledge of outputs, malicious recipient descriptors and adaptive corruption of previously private data. Long-term compromise may expose earlier note information; no retroactive privacy claim is possible without a precisely established forward-secrecy mechanism.

Threats the proof does not remove

A valid proof does not guarantee usable ciphertext delivery or protect against fraudulent wallet interfaces. Consensus finality does not guarantee data availability to a light client. Anonymity sets shrink when users make distinctive public withdrawals. A malicious RPC service can track queries even if it cannot forge a Merkle root. Exposing a viewing key allows disclosure according to that key’s scope, and previously leaked information cannot be withdrawn. These are not incidental operational details: they are part of the security model of a privacy-oriented ledger.

Public and Private Computation on One Ledger

Two execution environments with shared settlement

A private transfer and a transparent contract call differ in where their input data can be seen. Zoicy proposes to evaluate private logic on a client’s device, prove the necessary state effects and then apply publicly verifiable contract logic in a deterministic validator environment. The design follows an important constraint: **a private computation cannot rely on the return value of a public call that has not yet been executed**. Such a dependency would make its witness depend on a future, potentially conflicting state.

Let a candidate transaction have private effects ΔC (new commitments) and ΔN (nullifiers), and a sequence of queued public requests Q . Write

$$\text{PrivateEffects}(T) = (\Delta C, \Delta N, Q, \pi). \quad (18)$$

The client binds Q into the same authorization digest it signs for ZPS-1. At inclusion time, the validator first checks the applicable private proof, root and nullifier predicates; it then executes each public request against the state already modified by preceding valid transactions in the same block. A

private witness prepared for an earlier root is not permission to overwrite a more recent public account balance.

For the initial *fail-closed research profile*, let $S = (P, \mathcal{C}, \mathcal{N}, F)$ collect public balances and contract state P , the commitment tree $\mathcal{C}$, the spent-nullifier set $\mathcal{N}$ and fee accounting F . All transaction checks, private effects, public calls and integer conservation execute against a disposable scratch state. Define the proposed transition as a **partial function**:

$$\delta(S, T) = \begin{cases} S', & \text{if proof, authorization, public calls and fee checks succeed,} \\ \perp, & \text{otherwise (transaction inadmissible).} \end{cases} \quad (19)$$

This choice deliberately rejects an included transaction whose public application reverts; a block containing it is invalid, and **no private nullifier, public transfer or fee is committed** for that transaction. In this candidate profile, transaction fees are paid only by successfully committed transactions from value already shown to be available in the same atomic transition. This avoids a fee-on-revert ambiguity but leaves a separate spam/denial-of-service economic problem that must be addressed before mainnet. If the design later permits charging for a reverted call, it must use an independently authorized, replay-protected fee debit with its own public nonce and accounting, never an implicit partial commit of private nullifiers. No such advanced fee-on-revert rule is specified here.

Block processing is deterministic: each candidate T_i is evaluated against S_i ; if any $\delta(S_i, T_i) = \perp$, the entire proposed block is rejected, otherwise $S_{i+1} = \delta(S_i, T_i)$ and only the final root is committed under consensus. The public debit, private credit and fee debit must be justified by one accepted transition. A valid private proof cannot invent a public deposit or override later intervening public state. This is a **candidate research semantics** rather than the final fee market or contract VM.

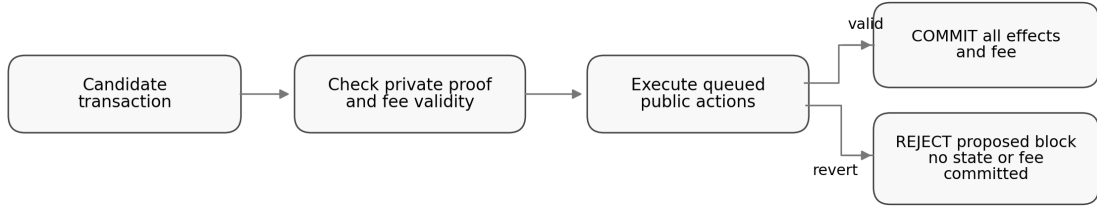


Figure 4: Candidate fail-closed transition: successful transactions commit all effects and fees atomically; a reverting public call rejects the proposed block with no state or fee committed.

Cross-domain conservation and data leakage

Public/private transfer boundaries require per-asset accounting. For asset a , let d_a be an authenticated public-to-private deposit, w_a an authenticated private-to-public withdrawal and f_a an explicit fee charged in that asset. Then a candidate transaction must satisfy

$$\sum_i v_{i,a}^{\text{in}} + d_a = \sum_j v_{j,a}^{\text{out}} + w_a + f_a. \quad (20)$$

Every d_a and w_a must correspond to an actual validated change in public state. For this fail-closed profile, the proof binds declared conversion quantities, and the deterministic public transition checks their corresponding debit or credit before the note/nullifier effects are committed. A public-call revert therefore aborts the complete transaction; there is no private-state-only success path. Otherwise a prover could cite a fictitious public deposit or conceal a public liability. The precise public/private conversion protocol, capability model, contract calls and failure semantics remain open. Their leakage is unavoidable in some designs: if a user deposits a uniquely sized amount and withdraws the same amount immediately, on-chain confidentiality may do little to prevent correlation.

Programmability should therefore proceed in layers. A first research milestone is a well-specified native private transfer with explicit public fee accounting. A second is a constrained private-to-public settlement call with rollback tests. Only then should generalized private contracts and recursive

composability be treated as engineering objectives. This staged approach does not narrow the long-term vision; it allows the privacy and correctness of each abstraction to be tested before complexity compounds.

Consensus, Finality and Quantum-Safe Validator Identity

A conditional safety observation

A useful safety argument begins with a simple observation. Suppose a finalized block requires more than two-thirds of an unchanged voting-weight set, and fewer than one-third of the weight is Byzantine. Any two finalization quorums must overlap in some honest voting power. If an honest validator cannot validly sign conflicting finalization transcripts under the protocol’s locking and round rules, a pair of conflicting finalizations cannot both be justified by honest voting. This is an *argument sketch*, not a consensus proof: the actual result requires exact validator-set transitions, lock-release conditions, evidence handling and a synchrony model.

Liveness has different prerequisites. Byzantine safety might hold during a long partition while no new blocks become final. Restoring progress can require eventual synchrony, controlled proposer rotation, timeout increases and a sufficient fraction of the current set remaining available. Quantum-resistant signatures must not create a signature-verification bottleneck large enough to invalidate the assumed network-delay bounds. Hence cryptographic choice and consensus tuning cannot be treated as independent design tasks.

Why validity proofs do not replace consensus

A STARK proves that a particular transition satisfies a chosen computation. It does not decide which of two individually valid, conflicting spends should happen first, which validator set is authorized to vote or whether all parties have received the necessary data. Consensus remains responsible for ordering and finality. Zoey’s candidate family is a stake-weighted, Byzantine fault-tolerant protocol with post-quantum-authenticated votes. The final protocol and its network synchrony assumptions have not been selected.

For a validator set with total voting weight W and Byzantine weight strictly below $W/3$, many BFT protocols finalize a block using a quorum with more than $2W/3$ of the weight. Two such quorums intersect in more than $W/3$, but intersection alone does not prove safety without the protocol’s locking, voting and validator-set update rules:

$$Q_1, Q_2 > \frac{2W}{3} \implies \text{weight}(Q_1 \cap Q_2) > \frac{W}{3}. \quad (21)$$

The property motivates the threshold but does not constitute a full proof of consensus. A complete specification must define proposal formation, vote rounds, prevote and precommit logic or equivalent phases, timeouts, proposer rotation, reconfiguration, evidence windows and recovery after partial network partitions. CometBFT is a useful reference for the shape of such a specification, not an already selected Zoey implementation [16].

Post-quantum vote transcripts

Validator signatures must authenticate *exactly* the content of a consensus message. A candidate vote digest should commit to the chain ID, epoch, height, round, message type, block or nil identifier and validator-set version. For validator i :

$$M_{i,t} = H_{\text{vote}}(\text{chainID}, e, h, r, \text{type}, B, V_t), \quad \sigma_i = \text{MLDSA}.\text{Sign}(sk_i, M_{i,t}). \quad (22)$$

Votes with invalid signatures, old epochs or incorrectly bound block IDs must be rejected. PQ signatures are generally larger than classical signatures and may significantly alter gossip volume, vote aggregation, light-client certificates and validator-set churn. A naive replacement of a BLS aggregate signature with many individual ML-DSA signatures can yield a materially different protocol cost. Zoey

must compare strategies using a declared validator-set size and adversarial network schedule rather than claiming a specific finality time in advance.

Security funding without inflation

A consensus protocol also needs a sustainable economic environment. Zoicy intends a fixed native monetary supply with no continuing coin issuance. Under that policy, validators cannot be promised ongoing newly minted rewards. A prospective long-run budget is therefore funded by fees or other explicitly authorized revenue, with the economic viability of sufficient decentralization left open. The whitepaper does not infer security from a token's total supply or current market value. Stake concentration, delegation, slashing and fees require separate game-theoretic treatment and cannot be reduced to the cryptographic soundness of a STARK.

Wallets, Encrypted State Availability and Metadata

Upgrade authority and emergency changes

Long-lived security demands the ability to retire a weakened algorithm without granting an administrator unconstrained control over account balances. A future Zoicy chain therefore needs explicit upgrade epochs and verifier IDs, with preannounced rules for when older notes remain spendable and how owners can authorize migration into new notes. An emergency change must not silently create a special spending key or retrospectively expand a viewing capability. Governance of the transition is part of the threat model because a mathematically correct new circuit can still be deployed through an unsafe social or administrative process.

An initial research paper cannot decide these governance details by assertion. It can, however, impose a constraint: *cryptographic agility must preserve the conservation, authorized-spend and one-nullifier invariants across versions*. Interoperability between old and new proofs is therefore a verifiable state-transition problem, not merely a client software update.

Key separation and recovery

A private note has at least four different user-facing functions: authorize spending, derive a nullifier, decrypt incoming data and recover access after device loss. Treating these as one key increases the consequences of compromise and can create undocumented cross-protocol assumptions. Zoicy instead proposes independent signing, nullification, viewing and recovery capabilities; all derived relationships must use purpose-specific key derivation and explicit versioning. Wallets should verify state roots and note openings locally rather than accepting an untrusted RPC server's balance as authoritative.

A recovery feature is not security-neutral. An externally retained key share can create a privacy or seizure risk, whereas purely local secrets can be irrecoverable. Proposed mechanisms such as social recovery or threshold authorization need post-quantum-compatible primitives and a precise account of who can authorize a transfer. The wallet interface must communicate which details a view key grants and which disclosures are irreversible.

Discovery, censorship and data availability

Encrypted note storage is useful only if the recipient can discover its ciphertext. Validators may correctly verify that an output commitment was created while a malicious sender fails to publish or correctly encrypt the associated payload. A robust design must define a bound between accepted output commitments and accessible encrypted envelopes, a scanning/indexing rule and the maximum cost of note discovery. Publishing every ciphertext has bandwidth costs; asking an RPC server only for one's own ciphertexts exposes usage patterns. These trade-offs are protocol-level concerns, not optional wallet polish.

Data availability can be expressed as a required predicate over a finalized block B and an authorized note receiver u :

$$\text{Finalize}(B) = 1 \not\Rightarrow \text{DecryptableFor}(u, B) = 1. \quad (23)$$

The implication deliberately does not hold without additional envelope-validity and retrieval conditions. Finality is a statement about consensus; recipient access requires different evidence. A future ciphertext-binding mechanism might prove correct encryption or require authenticated acknowledgments, but any such design must balance availability, denial of service and receiver privacy.

The network observer remains powerful

A passive observer can correlate packet timing, IP addresses, inclusion heights, failed transactions, distinctive proof sizes, reused relay services and cross-domain deposits. Padding, batching, relayers, private scanning and variable submission delays may reduce selected leakages, at the cost of throughput, latency or availability. Zoicy should evaluate these defenses with an explicit attacker model; the paper makes no blanket claim of anonymity from an internet provider or an adversary controlling the routing infrastructure.

Base Deployment and a Future Native Network

The monetary policy and initial distribution are separate from the research claims above. Zoicy intends an initial Base ERC-20 token with a maximum fixed supply of **21,000,000 ZOICY**, no additional mint function, no reserved team or private-investor allocation and a fair public distribution with a target of 100% circulating supply at launch. These commitments must be verified against the actual deployed token contract, holder distribution and liquidity mechanics; minting all tokens to a single controller is not evidence that all tokens circulate. A standard Base ERC-20 does not acquire private transfers or post-quantum signatures simply because a later network is proposed [9,10,15].

Why bridging is a different trust problem

Migration is often portrayed as a user-interface swap, but it combines at least three independently difficult questions. The source chain must provide evidence that units are irreversibly locked or burned under an agreed finality rule; the destination must verify that evidence without trusting a privileged signer to invent claims; and recipients must securely establish ownership on the new ledger. These conditions apply even if the destination’s own transaction authorization is post-quantum. A historically classical source signature or commitment remains a risk at the boundary.

A snapshot-based airdrop would avoid some live bridge complexity but raises cutoff-height, omitted-holder and replay questions. A live burn-and-redeem bridge provides ongoing flexibility but introduces proof verification, reorganization and custody risks. The locked native genesis-reserve candidate in this paper is not a claim that one can construct a fully trustless Base verifier automatically. Its selection must follow a threat-model comparison and public testing of exceptional cases, including paused migrations, chain reorganizations, lost keys and competing claims.

A conserved migration, not unconstrained new issuance

If Zoicy eventually creates an independent native L1, a migration process must prevent a holder from spending economically equivalent units both on Base and on the native chain. A proposed model creates the fixed native supply in a **locked genesis reserve**, then releases units after validating finalized, one-time source-chain migration claims. This would be initial native genesis allocation rather than later unrestricted monetary minting. It still depends on the security of source-chain evidence and the migration verifier, and should not be presented as a completed bridge.

Let $U_B(t)$ and $U_N(t)$ denote *simultaneously spendable* economic units on Base and the native chain at time t . The required cross-domain bound is

$$U_B(t) + U_N(t) \leq 21,000,000. \quad (24)$$

For a source claim identifier j , native release requires a valid finalized claim, an unused claim marker and a reserve balance sufficient to cover the amount. Abstractly:

$$\text{Release}(j, v) = 1 \implies (\text{FinalizedSource}(j) = 1) \wedge (j \notin \mathcal{J}_{\text{used}}) \wedge (v \leq R_{\text{locked}}). \quad (25)$$

A valid release updates $\mathcal{J}_{\text{used}}$ and reserve accounting atomically. Reorganizations, replay, partial burns, lost historical keys and bridge compromise can violate this invariant unless independently handled. Critically, **a Base-to-native migration inherits a classical cryptographic boundary from Base history**; its existence must not be used to advertise the migration process itself as end-to-end post-quantum secure.

Research Method and Falsifiable Evaluation

A protocol proposal is more informative when its central assumptions are exposed to potential refutation. Zoicy’s research program should therefore distinguish (a) deterministic state-machine conformance, (b) cryptographic proof correctness and security, (c) validator/network performance and (d) privacy against a defined observer. These are separate lines of evidence and should not be collapsed into one ‘tests passed’ statement.

Candidate conformance fixtures

A small Python model associated with the earlier ZPS-1 draft exercised canonical public-input binding and deterministic validation behavior. Its **12 passing tests** covered replay, changed outputs or fees, chain mismatch, duplicate or stale nullifiers, and related rejection cases. The model contains no complete ML-DSA-in-STARK circuit and does not prove witness confidentiality, post-quantum soundness or consensus security. Such a model is useful because it can expose disagreements between two implementations about basic state rules; its scope must remain explicit.

For a future reference prover, each published benchmark should specify circuit hash, verifier ID, field, FRI parameters, input/output counts, hardware, software commit, proof size, prover latency, verifier latency and peak memory. A credible comparison requires at least private note creation, single- and multi-input spending, in-circuit authorization, client scanning and public/private conversion. Before measurements exist, those cells should be reported as **not measured**, not replaced by illustrative throughput figures.

A prospective proof roadmap

The earliest formal arguments need not pretend to resolve the entire quantum cryptographic composition. First, a *state-machine model* can establish the conditional invariants of Propositions 1 and 2 and identify the exact assumptions needed for a proof verifier to justify a witness. Second, a *cryptographic authorization experiment* must fix key generation, corruption timing, chosen-message access and transcript replay, then show how unauthorized spending would relate to ML-DSA unforgeability, note-commitment binding or ZK argument soundness. Third, a *privacy experiment* must state which observers can access the recipient descriptor, ciphertext envelope, note payload, future nullifier and network timing. A game that hides these observations by fiat does not justify operational anonymity.

Fourth, compare the two output-delivery policies by constructing a payer-generated malformed envelope. The baseline should admit that it cannot infer recipient decryptability; strict delivery is acceptable only if an explicit encryption-correctness relation exists and matches the note opening and key in d . Fifth, an isolated consensus proof must specify the validator reconfiguration and locking rules instead of inferring safety from the quorum inequality alone. Combining those separate claims would require a reviewed composition argument; this paper provides a dependency map, **not** that final composition theorem.

Security argument structure

A defensible final security argument should proceed by declaring the exact real-world protocol, a quantum adversary model, admissible corruptions, public leakage and compositional assumptions. It should then address authorization unforgeability, note hiding, nullifier unpredictability, circuit soundness, zero-knowledge simulation, integer accounting and consensus finality, with explicit reductions or

counterexamples. A heuristic expression summing independent failure probabilities is not a security theorem unless the component events, dependencies and games have been defined.

The minimal verification work is concrete: first pin canonical encodings and a reproducible output/envelope binding; then design the missing correct-encryption proof or a privacy-preserving recipient acknowledgement mechanism; then compare the costs of ML-DSA-in-STARK with alternate post-quantum authorization. A sender-controlled, malformed ciphertext should become an explicit negative test case rather than being treated as a rare wallet error. The immediate open questions are concrete. Can in-circuit ML-DSA validation be made practical? Does the commitment/nullifier pairing remain sender-unlinkable under reusable or compromised descriptors? Can public/private execution preserve atomicity while permitting efficient fee collection? How much information is revealed by ciphertext discovery and private-to-public transitions? How will PQ-signed BFT voting behave as stake and validator count change? These questions define experimental milestones rather than shortcomings to conceal in marketing language.

Relation to Earlier Designs

Zoiy draws from established technical ideas without treating those projects as interchangeable implementations. STARKs motivate transparent integrity proofs [5,6]. The Zerocash paper formalizes the goals of decentralized anonymous payments, illustrating why correctness and payment privacy require distinct security definitions [21]; Zcash demonstrates deployed shielded-transaction experience [11]. Aztec provides concrete motivations for separating private and public execution, and Miden illustrates a local-execution, proof-submission design space [7,8,13,17,19]. NIST standards are sources for post-quantum primitive syntax, not a proof of protocol composition [1–4]. CometBFT provides a well-developed starting reference for stake-weighted BFT vote and state-machine design [16], while HotStuff supplies an influential partially synchronous BFT research model [22]. Neither is adopted wholesale by Zoiy.

The intended contribution is best compared against explicit alternatives. Ordinary public-account ledgers already have a transparent ordering and accounting model but do not hide their ordinary transaction graph. Shielded-note systems already address confidential payment semantics; Zoiy does **not** claim invention of commitments, nullifiers, or zero-knowledge transfers. Aztec and Miden study public/private programmability and proof execution. Standards for ML-DSA and ML-KEM provide tested primitive definitions, but not a low-cost hidden-signature proving circuit. The proposed differentiator is the **specific research composition** of recipient-controlled future-nullifier secrecy, descriptor-bound encryption keys, private PQ authorization, transcript-wide ciphertext binding and a unified transition interface for transparent calls. This composition has not yet been demonstrated to outperform any named system; its potential novelty is an open research question rather than an established priority claim.

Limitations, Research Boundary and Conclusion

Zoiy is presently a paper-level protocol proposal, not a functioning Layer 1. The proposed ZPS-1 relation has no completed prover/verifier, formal quantum-security proof or independent audit. Mandatory cryptographic parameters, full note scanning, general-purpose private execution, actual BFT consensus, wallet recovery, fee accounting and migration verification remain design and engineering tasks. Some may prove prohibitively expensive or require replacing the candidate architecture. A sound technical document should acknowledge that possibility rather than retrofit unverifiable promises.

The design nevertheless yields a clear, falsifiable research thesis. Its most persuasive claim is narrow: a collection of explicit predicates and transactional rules can be checked for internal consistency before promising a general-purpose PQ privacy chain. Stronger claims about speed, sender anonymity, receiver recoverability and validator decentralization require independent evidence. A user should be able to commit a private note under a recipient-generated authorization descriptor, transfer spending authority without leaking the receiver’s future nullifier secret, prove a later post-quantum-authorized spend, and settle the transition in an atomic, publicly verified and finally ordered state. Transparent applications must operate without being granted hidden witness data; consensus must authenticate

its own history with appropriately selected post-quantum algorithms. A fixed-supply Base token, if launched, is an initial ecosystem distribution, not a substitute for these properties.

Whether this thesis can be made practical depends on concrete proof costs, rigorous composition arguments, availability and leakage behavior, and protocol testing. Those are the next research obligations. Until then, the strongest accurate description of Zoicy is a **privacy-native, post-quantum-oriented Layer 1 research design with an explicit candidate private-spending mechanism**.

References

- [1] National Institute of Standards and Technology. **FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard**. 2024. <https://csrc.nist.gov/pubs/fips/203/final>
- [2] National Institute of Standards and Technology. **FIPS 204: Module-Lattice-Based Digital Signature Standard**. 2024. <https://csrc.nist.gov/pubs/fips/204/final>
- [3] National Institute of Standards and Technology. **FIPS 205: Stateless Hash-Based Digital Signature Standard**. 2024. <https://csrc.nist.gov/pubs/fips/205/final>
- [4] National Institute of Standards and Technology. **SP 800-227: Recommendations for Key-Encapsulation Mechanisms**. 2025. <https://csrc.nist.gov/pubs/sp/800/227/final>
- [5] E. Ben-Sasson, I. Bentov, Y. Horesh and M. Riabzev. **Scalable, Transparent, and Post-Quantum Secure Computational Integrity**. IACR ePrint 2018/046. <https://eprint.iacr.org/2018/046>
- [6] StarkWare. **STARK Technology: Transparent Proofs and Quantum-Security Motivation**. <https://starkware.co/stark/>
- [7] Aztec. **Foundational Topics: Public and Private Execution**. <https://docs.aztec.network/developers/docs/foundational-topics>
- [8] Miden. **Protocol Repository and Architecture**. <https://github.com/0xMiden/protocol>
- [9] Base. **Base Transaction Model and Documentation**. <https://docs.base.org/get-started/make-a-transaction>
- [10] OpenZeppelin. **ERC-20 Supply and Constructor Minting**. <https://docs.openzeppelin.com/contracts/5.x/erc20-supply>
- [11] Zcash. **Shielded Transactions and Privacy Concepts**. <https://z.cash/learn/>
- [12] Ethereum.org. **Proof-of-Stake Rewards and Penalties**. <https://ethereum.org/developers/docs/consensus-mechanisms/pos/rewards-and-penalties/>
- [13] Aztec. **State Management**. https://docs.aztec.network/developers/docs/foundational-topics/state_management
- [14] StarkWare Quantum Hub. **Quantum-Security Foundations**. <https://quantum.starkware.co/foundations>
- [15] Ethereum.org. **ERC-20 Token Standard**. <https://ethereum.org/developers/docs/standards/tokens/erc-20/>
- [16] CometBFT. **Consensus Specification**. <https://github.com/cometbft/cometbft/blob/main/spec/consensus/consensus.md>
- [17] Aztec. **Private Note and Nullifier State Variables**. https://docs.aztec.network/developers/docs/aztec-nr/framework-description/state_variables
- [18] NIST. **FIPS 202: SHA-3 Standard**. 2015. <https://csrc.nist.gov/pubs/fips/202/final>
- [19] Aztec. **Private/Public Call Types and Revert Behavior**. https://docs.aztec.network/developers/testnet/docs/foundational-topics/call_types
- [20] IETF. **RFC 8949: Concise Binary Object Representation (CBOR)**. 2020. <https://www.rfc-editor.org/rfc/rfc8949.html>

- [21] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer and M. Virza. **Zerocash: Decentralized Anonymous Payments from Bitcoin.** IEEE Symposium on Security and Privacy, 2014. <https://eprint.iacr.org/2014/349>
- [22] M. Yin, D. Malkhi, M. K. Reiter, G. Golan Gueta and I. Abraham. **HotStuff: BFT Consensus in the Lens of Blockchain.** PODC 2019; arXiv:1803.05069. <https://arxiv.org/abs/1803.05069>