



Fuyu: Programmable Privacy for DeFi from zkSNARKs

September 2026

The Fuyu Team

Abstract

On a public blockchain, anyone can read every balance, payment, and trade, and can often link them to their owners. Privacy pools hide who owns which funds, but most of them rely on elliptic-curve cryptography, which a quantum computer would break. Some depend on an operator to reach DeFi, and disclosure tools reveal either an account's entire history or a single fact about where its funds came from. We present Fuyu, a privacy pool for EVM chains that addresses these problems. Fuyu records the owner of each payment as a hash instead of an elliptic-curve public key, and it encrypts each payment with X-Wing, which combines the post-quantum ML-KEM-768 with elliptic-curve X25519 and stays secure while either one is. An account's recorded history therefore stays private even against a future quantum computer, as long as ML-KEM-768 is secure. For DeFi, Fuyu's Pool contract itself measures what each action returned and records the result in a receipt that anyone can finalize, so no operator is needed. No governance role, relayer, or finalizer can pause the Pool or move, freeze, or reveal a user's funds. For compliance, an auditor asks Yes/No questions in the Fuyu Policy Language (FPL), and for lower bounds on balances and activity, the holder answers on its own device with proofs that reveal only the answers. FPL's type system decides which answers the holder's records can certify. A note can also require a second approval, such as an institution's Safe quorum. Among the pools that support both private payments and DeFi, Fuyu offers the strongest privacy: it is the only one whose payment encryption resists quantum computers, and the only one whose audits prove facts about balances and activity while revealing only the answers. Its DeFi actions also cost 15 to 40% less gas than RAILGUN's. We argue that Fuyu meets its integrity, privacy, and disclosure goals under standard cryptographic assumptions, given a publisher trusted to screen deposit sources and admit venues.

1 Introduction

On Ethereum, transactions are “fully transparent by default” [23]: anyone can see every balance, payment, and trade, and the record is permanent. A bank account, in contrast, is visible only to its owner, the bank, and the authorities. Blockchain addresses carry no names, but they rarely stay anonymous. Users reveal their identities through public names, an exchange's passport check, or by accident [23], and analysts can then link a user's other addresses into one profile [24]. This exposure is costly. A fund's positions reveal its strategy, a company's payments reveal its suppliers, and a large visible balance invites extortion and physical attacks [27]. As regulated stablecoins and tokenized securities move onchain [1], the institutions that hold them cannot accept it.

The problem. Such users need *programmable privacy*: their transactions are private by default, and they disclose information only when they choose to [23]. Concretely, a holder should be able (i) to

keep its balances, payments, and trades private permanently, even after quantum computers exist; (ii) to pay, earn yield in DeFi, and withdraw without depending on any operator; and (iii) to prove to each auditor exactly the facts that this auditor needs, such as “the balance exceeds \$1M” or “no deposit came from a sanctioned address”, and nothing more. Throughout, only the holder should be able to spend its funds or disclose its private data.

Existing approaches. Privacy pools such as Tornado Cash, RAILGUN, and zkBob hide which deposit funded a withdrawal [8, 29, 31, 40], but they fall short in three ways. First, their privacy may not last. Most rely on elliptic-curve cryptography, which a quantum computer would break, and L2BEAT rates Tornado Cash and RAILGUN at risk against a quantum adversary and Privacy Boost exposed [22]. Privacy Pools records only hashes, but it offers neither private transfers nor DeFi. Second, some pools depend on an operator. Aztec Connect sent its users' DeFi transactions through its own operator, and when the operator shut down, users could only withdraw [4, 5]. Third, disclosure is all or nothing. A Zcash viewing key reveals an account's entire history [18], a hosted audit service decrypts every transaction inside a trusted enclave [30], and screening proofs show only where funds came from [11, 32]. Tornado Cash can reveal only which deposit funded a withdrawal [22], and it was sanctioned after being used to launder over \$7 billion [37].

Challenges. Meeting these needs raises three challenges.

C1: Post-quantum privacy. Privacy matters most, and on a blockchain it has to last. The record is public and permanent, so an adversary can store it today and decrypt it once quantum computers break elliptic-curve cryptography. A pool must therefore hide who owns each note, what each payment contains, and how each recipient finds its payments, even from such an adversary.

C2: Decentralization. The protocol must have no central point of failure: transfers, trades, and withdrawals must keep working when an operator, a relayer, or the protocol's governors stop. DeFi makes this hard. A holder must create its proof before its trade runs, so the proof cannot state what the trade will return, and some party has to record the result. In Aztec Connect, that party was the operator.

C3: Compliance. In a compliance review, a holder should not have to hand over all its records, yet the auditor needs answers. The middle ground is for the auditor to ask rich Yes/No questions and for the holder to answer each one with a proof that reveals only the answer. Records that a holder presents can prove a balance above \$1M, but they can never prove that no deposit came from a sanctioned address, because the holder could leave some deposits out. A disclosure system must know which answers need the complete history. Institutional control is closely related: an institution's funds must follow its own approval rule, such as a multisig quorum.

Our approach. We present Fuyu (Figure 1), a privacy pool for EVM chains that meets each challenge in turn.

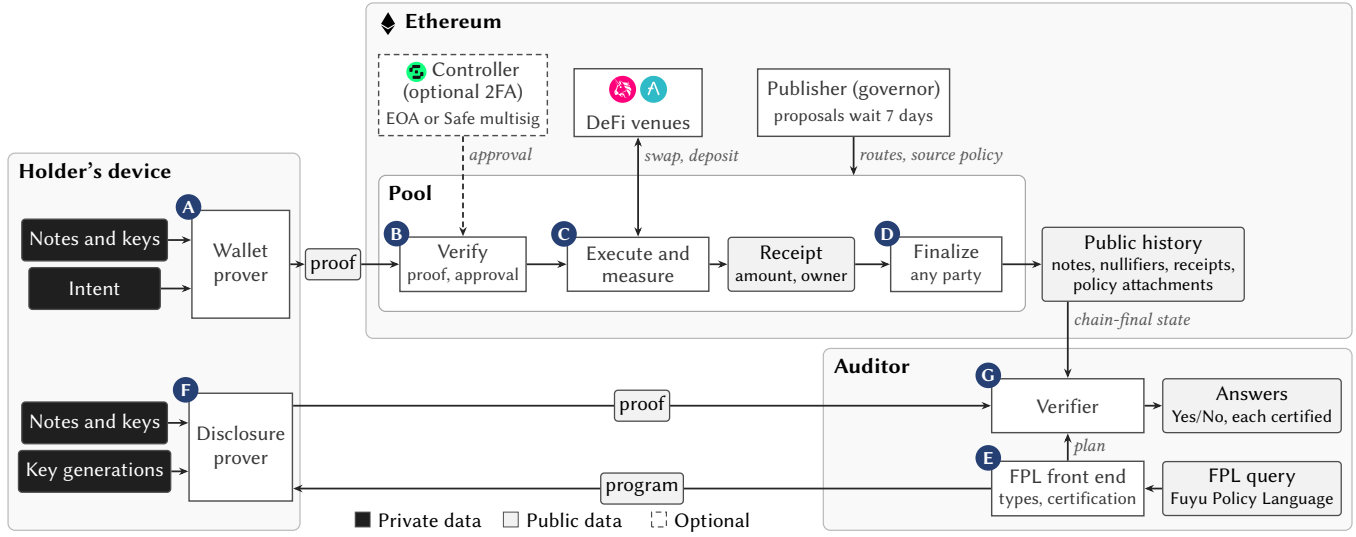


Figure 1: Fuyu’s architecture. The holder’s wallet proves a spend (A). The Pool verifies the proof and the approval of the optional controller (dashed) (B), runs the action and measures its output (C), and any party finalizes the receipt (D). The auditor’s front end checks a program of Yes/No questions in the Fuyu Policy Language (FPL) and compiles it to proof plans (E). The holder answers with proofs from its own device (F), which the auditor’s verifier checks against public history (G).

Post-quantum records (C1). Like Zerocash [8], Fuyu holds funds in *notes*: each note is an amount of one asset with an owner, and the Pool stores only a commitment to it. A Fuyu note records its owner as a one-time hash of the owner’s key, and the tag that marks a note as spent is also a hash, so neither contains an elliptic-curve public key. When Alice pays Bob, her wallet encrypts Bob’s new note with X-Wing [6], which combines the post-quantum ML-KEM-768 with the elliptic-curve X25519 and stays secure while either one is. Bob’s wallet finds his payments by trying to decrypt each new one, so no elliptic-curve hint in the record points to him. The spend proof also binds each ciphertext to its note. Recorded payments therefore stay private against a quantum computer as long as ML-KEM-768 is secure.

Measured settlement (C2). For a DeFi action, the holder’s proof fixes what the holder controls: the route, the exact input, the minimum return, and the owner of the result. The Pool runs the action itself, measures what the venue returned, and records the result in a receipt. Any party can then finalize the receipt, which makes the new note spendable, so no operator is needed. A holder can submit its transactions through a relay to hide its address, or submit them itself. No governance role can pause the Pool or move, freeze, or reveal a note, and no governance action can stop a note from leaving the Pool. Once funds are private, no operator’s, relay’s, or governor’s absence can stop their transfers, trades, or withdrawals. New deposits still depend on the publisher, since a deposit moves privately only after it passes the source policy.

Selective disclosure (C3). An auditor writes its questions in our Fuyu Policy Language (FPL). FPL’s type system decides, for each question, whether the holder’s records can certify the answer or only the account’s complete history can. For lower bounds on

balances and activity, the holder answers on its own device with proofs that reveal only the answers, and the auditor checks them against its own Ethereum node. Deposits must pass a published source policy before they move privately, and, under a deny-list policy, Fuyu’s governance can block an illicit source at once. For institutional control, a note can require a second wallet, such as a treasury’s Safe multisig, to approve every spend, like a second factor (2FA). We call this wallet the note’s *controller*, and controlled notes share one Pool with other notes.

Evaluation highlights.

- **Privacy.** Of the pools we compare, three support both private payments and DeFi: RAILGUN, Privacy Boost, and Fuyu. Fuyu offers the strongest privacy of the three, and it matches or beats the other two on every row of Table 3. Only Fuyu keeps links private against a future quantum adversary, as long as ML-KEM-768 is secure, because only its payment encryption resists quantum computers; L2BEAT rates RAILGUN at risk and Privacy Boost exposed against such an adversary. Only its audits prove facts about balances and activity while revealing just the answers. An auditor learns one Yes/No answer per question, but not which payments count, how many there are, or their amounts, whereas a viewing key reveals an account’s entire history.
- **Cost.** A DeFi action costs 31 to 40% less gas than in RAILGUN, and still 15 to 23% less when the Pool holds a million notes. A deposit costs 54% less than in Tornado Cash, and still 14% less at a million notes.
- **Compliance.** FPL expresses all sixteen questions of our benchmark, and its types show that the holder’s records can certify a Yes for twelve of them. Holders can prove five without revealing

any record, while other systems answer at most two without disclosing a history. Proving a balance over up to 64 notes takes 298 s with 16 threads, and verifying the proof takes 1.4 s.

Contributions. This paper makes five contributions.

- **A language for compliance questions.** We introduce FPL, in which auditors write Yes/No questions about an account's history. Its type system decides whether the holder's records can certify each answer or only the complete history can, and its compiler plans proofs only for the former (Section 6).
- **Audits that reveal only the answer.** A holder proves on its own device that its balance, deposits, trades, or withdrawals reach an amount such as 100,000 USDC, without revealing which payments count, how many, or their amounts (Section 5.2).
- **Payment records that stay private against a quantum computer.** Fuyu records owners as hashes instead of elliptic-curve public keys and encrypts each payment with X-Wing. The spend proof binds each ciphertext to its note, and recipients find payments by trying to decrypt them, without elliptic-curve hints. Records stay private while ML-KEM-768 is secure (Section 4.1).
- **DeFi with no central point of failure.** The Pool measures each action's output and fixes it in a receipt that any party can finalize, and no role can pause the Pool (Sections 3 and 4.4).
- **Implementation and evaluation.** We implemented Fuyu, and all 23 funded end-to-end suites passed on a copy of Ethereum mainnet. The highlights above summarize our comparison with Tornado Cash, Privacy Pools, RAILGUN, and Privacy Boost (Sections 8 and 9).

2 Overview

2.1 Background

DeFi venues. Automated market makers [2] and tokenized vaults [34] exchange one ERC-20 token for another. The rate is fixed only when the call executes, since other transactions can move it first, so a caller states the minimum output that it accepts.

Privacy pools. A privacy pool is one contract that holds the funds of many users, so that a withdrawal cannot be linked to the deposit that funded it [23]. Fuyu follows the note design of Zerocash [8]. A *note* is an amount of one asset that only the holder of a secret can spend. The pool stores only a *commitment* to each note, as a leaf of an append-only Merkle tree. To spend a note, its holder proves in zero knowledge that it knows the opening of some leaf, without saying which, and publishes the note's *nullifier*, a tag that blocks any second spend of the same note. A pool's privacy is its *anonymity set*, the users who could have made a given transaction; address reuse, distinctive amounts, and timing shrink it [23].

Zero-knowledge proofs. Fuyu uses Groth16 [17], a proof system with short proofs that are cheap to verify onchain, over circuits written in Circom. We write $\mathcal{R}(X; w)$ for a relation between a public statement X and a private witness w .

Safe accounts. A Safe holds assets under a set of owners and a threshold, which its owners can change. An ordinary Safe transaction needs signatures from at least threshold-many current owners, which we call a *quorum*, while enabled modules can act without

them [33]. An owner is either an externally owned account (EOA) or a contract, such as a passkey signer or another Safe.

Post-quantum encryption. A key encapsulation mechanism (KEM) lets a sender derive a shared secret from a recipient's public key, and only the holder of the matching private key can recover it. X-Wing combines post-quantum ML-KEM-768 with elliptic-curve X25519 and stays secure while either does [6, 26].

2.2 Motivating Example

Consider a fund that keeps its treasury onchain in stablecoins and tokenized shares, governed by a Safe multisig. The fund wants its payments hidden for years, its idle stablecoins earning yield in a lending vault without an operator, and its quorum rule to hold for every payment. Each quarter its lender asks five questions. Is the balance above \$1M? Did any deposit come from a sanctioned address? Was the vault position held on September 30? Did trading stay within its daily limit? Is the owner a verified entity? A public ledger answers the first four and reveals everything else; a privacy pool reveals nothing and answers at most the second. Keeping its payments private for years is challenge C1, earning yield without an operator is C2, and enforcing its quorum rule and answering the lender's questions is C3. Figure 3 writes the questions in FPL.

2.3 Walkthrough

Figure 2 follows Alice's deposit through four transactions and an offchain query. Notes work like Bitcoin's unspent outputs: a spend consumes whole notes and creates new ones for payment and change, so any amount can be deposited, paid, or withdrawn.

- (1) *Deposit.* Alice deposits an amount a of a token X from her own address. The Pool itself computes the commitment of a *Pending* note, so a deposit needs no proof, and adds it to the note tree. Everyone sees that Alice deposited a of X .
- (2) *Transfer.* Alice pays Bob an amount b of X privately. Her wallet spends her whole note and creates two *Active* notes: b of X for Bob and $a - b$ of change for her. It proves that she owns a note in the tree and publishes that note's nullifier, which the Pool checks is unused. Each new note is a commitment and an encryption of its opening under the recipient's public key, so Bob need not be online when Alice pays. Because Alice's note is Pending, this first private spend is *screened*: the proof shows that the deposit's source passes the *source policy*, a list of allowed or denied deposit sources that the *publisher* maintains. Alice's wallet sends the proof to a *relayer*, which submits it from the relayer's own address, so Alice's address does not appear. Anyone can act as a relayer: the role needs no secret, and the proof binds the whole request, so a relayer cannot alter it.
- (3) *Execute.* Bob swaps his b of X for a token Y on Uniswap. His proof authorizes an *intent*: exactly b of X in, through one admitted *route*, at least m of Y back, and a fresh *owner tag* of Bob's, a one-time name for the result's owner that links to no address. The minimum m protects Bob against a price change before execution. A route names an *adapter*, a contract that runs one operation at one venue, and its asset pair. The relayer submits Bob's proof as in step 2. The Pool calls the adapter, measures its own balance change, which is the amount q of Y that Uniswap

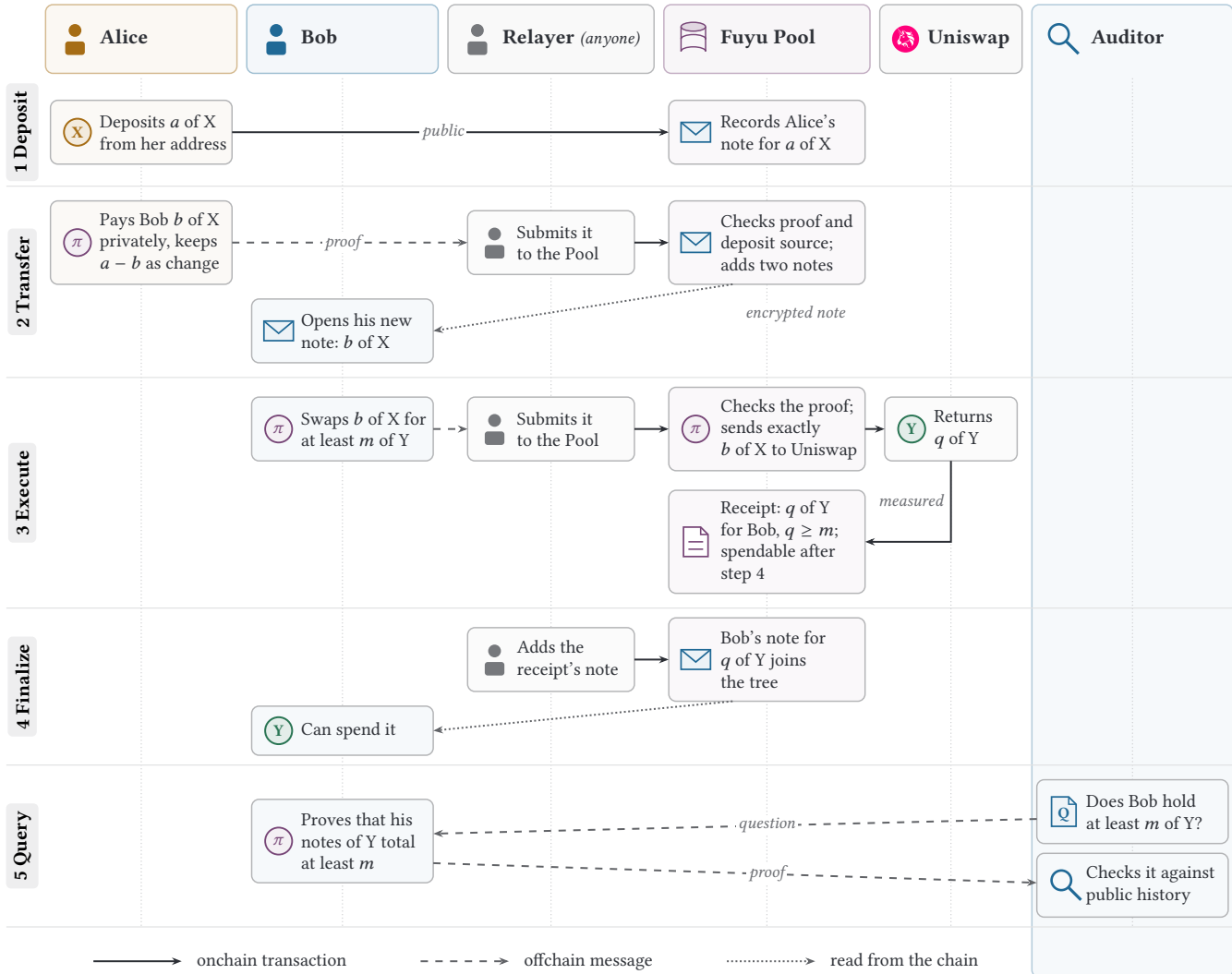


Figure 2: One deposit's path through Fuyu. (1) Alice deposits an amount a of token X. (2) She pays Bob b of X privately and keeps $a - b$ as change. A relayer submits her proof; relaying is permissionless, so anyone can be the relayer. (3) Through the relayer, Bob swaps his b of X on Uniswap for at least m of token Y. The Pool measures the q of Y that Uniswap returns and fixes Bob's note for it in a receipt. (4) The relayer adds this note to the Pool's note tree, which makes it spendable. (5) An auditor asks whether Bob holds at least m of Y, and Bob answers with a proof. Layout after Thunderbolt [38].

returned, and requires $q \geq m$. It then records a *Settled* note for exactly q of Y, owned under Bob's tag, in an action *receipt* stored under the spent note's nullifier. The receipt fixes the note, but Bob cannot spend it until it is in the tree. Had Bob's note named a *controller*, such as the fund's Safe, it would also have to approve this spend (Section 4.3).

- (4) Finalize.** The relayer now acts as the *finalizer*: it adds the receipt's note to the note tree, which we call *appending* it. Every spend proves membership in the tree, so only an appended note is spendable. The receipt fixes the amount and the owner, so the finalizer can change neither. Appending is a separate step so that any other party could complete the receipt instead; by

default, a router runs Execute and Finalize in one transaction (Section 4.2).

- (5) Query.** Later, an auditor asks, in FPL, whether Bob holds at least m of Y. Bob runs the query on his own device and proves that his unspent notes of Y total at least m , without revealing which notes, how many, or their nullifiers. Answering is Bob's choice, and no Fuyu service takes part: the auditor's own verifier checks the proof against the Pool's public history.

What observers learn. Table 1 lists what each operation reveals. Observers see Alice's deposit of a in step 1, the relayer as the sender of steps 2 to 4, and the venue, the assets, and the amounts b , m , and

Table 1: What each operation reveals to all observers.

Operation	Publicly revealed
Every transaction	Submitter (such as a relay) and timing
Every spend	Nullifiers, the tree root that it proves against, the fee, relay, and deadline, its controller with its approval, and its Pending bit; a screened spend also names the source policy that it satisfies
Deposit	Depositor, asset, amount, the new note’s commitment, and its policy attachment
Transfer	New commitments, their encrypted openings, and their policy attachments; the asset only when a fee is paid in it
Execute	Route, exact input, minimum return, measured output, Settled owner tag, and change commitment
Finalize	Nothing new: the receipt was fixed at Execute
Withdraw	Recipient, asset, amount, and the change commitment
Exit	Deposit source, asset, payout recipient and amount, and fee

q of Bob’s action in step 3; trades against public venues reveal their quantities anyway [3]. From step 2’s public *Pending bit*, they also see that it spent one of the unspent deposits that the source policy covers. They do not see which one, that step 2 paid Bob, or that Bob acted in step 3: a spend reveals nullifiers and new commitments, not the notes that it consumed, and the Settled note’s owner tag is fresh. The remaining handles for linking Bob’s action to Alice’s deposit are the amounts and the timing, so common amounts and delays still matter [23]. A holder leaves the Pool with a *withdrawal* to a public recipient or, for a deposit that fails screening, a public *exit* that reveals its source.

3 Threat Model and Goals

Parties and authority. Table 2 lists what each party may do. Auditors include external auditors, counterparties, and regulated intermediaries, each running its own verifier.

Governance. The publisher is a governor contract, not a key. A council contract, typically a threshold multisig, proposes admissions, source policies, and role changes; each proposal waits at least seven days. For at most 30 days after deployment, a *genesis operator* can also make any such change at once; nothing reopens this *genesis* window. A *guardian* and the council can act at once, but only to reduce risk. Either can veto a proposal (the guardian cannot veto its own replacement), revoke an unprotected route, or, when the source policy is a deny list, deny a hacked or sanctioned deposit source; together they can deny at most 64 sources a day. A proposal can protect an exit route, and then neither can revoke it at once (Appendix C.1).

Exits and upgrades. No role can pause the Pool or move, freeze, or reveal a note, and a withdrawal needs no route, so a position’s token can always be withdrawn. Upgrades ship as new routes and periphery that users choose per action, or as a successor Pool that holders may move to by spending; they never change the notes in the Pool.

Table 2: What each party in Fuyu may and cannot do.

Party	May	Cannot
Holder	Authorize one exact intent	Create value; bypass a controller
Controller	Co-approve each spend of a controlled note	Spend without the note’s opening
Adapter	Execute one admitted operation	Choose the owner; misreport the output
Publisher (governor)	Admit assets and routes and publish source policies, each at least seven days after its proposal (at once in genesis)	Spend, freeze, or reveal notes; alter receipts
Guardian or council	At once: veto a proposal; revoke an unprotected route; deny 64 sources a day	Admit anything or lift a denial at once
Genesis operator	Any publisher change at once, for up to 30 days	Reopen genesis; spend, freeze, or reveal notes
Relayer	Submit, delay, or refuse a request	Alter a request
Finalizer	Append a fixed receipt (any party)	Change its amount or owner
Auditor	Give the holder a program; verify the answer	Compel an answer

Trust and assumptions. We trust the publisher to classify deposit sources correctly and to admit only adapters that behave as reviewed and only *exact-transfer* assets, whose every transfer moves exactly the stated amount. We trust each client’s RPC endpoint to report *chain-final* state. For FPL answers, we also trust the auditor’s registry of price feeds, list snapshots, and credential issuers. We assume that no adversary knows the trapdoor of any Groth16 setup, that Groth16 is knowledge-sound [17], that note commitments are binding and hiding, and that the KEM is IND-CCA secure. Spend soundness rests on Groth16 over the BN254 curve, which a quantum adversary could break, but the privacy of an account’s history rests on no elliptic-curve assumption (Section 4.1). Appendix C.1 states them all.

Adversary. The adversary controls any holders other than the victim; any relayers, finalizers, and auditors; but not the key of a victim’s controlling wallet or, for a Safe, a quorum of its owners or any of its enabled modules. It may submit malformed proofs, race or replay transactions, and observe all public activity, including pending transactions; an approval covers one exact statement, so a captured approval authorizes no other spend. It may move venue prices, for example by trading ahead of an action [12]; the holder’s minimum return bounds the loss. For privacy, it may later obtain a quantum computer and apply it to all recorded history.

Goals. Fuyu provides five integrity properties: ownership, single consumption, conservation, fixed settlement and delivery, and solvency (Section 7). For privacy, an operation reveals only its declared fields (Table 1, in full Table 5), plus timing. For disclosure, an accepted answer implies its statement, a rejected or missing answer implies nothing, and an FPL answer is accepted only with

the evidence that its type requires. For control, a controlled note moves only with its controller’s approval of that exact spend.

4 The Fuyu Protocol

This section defines the notes and the ledger, the post-quantum delivery (C1), the operations, controlled notes, and the measured settlement (C2). Appendix A gives the encodings, the action algorithm, note recovery, and payments to unregistered addresses.

4.1 Notes and Ledger State

A Fuyu note works like a Zerocash note (Section 2.1) with three additions. First, it commits to a *controller*, which is either zero or a wallet that must co-approve each spend (Section 4.3). Second, it has one of three types, which share one spend interface. A *Pending* note records a public deposit, an *Active* note holds an encrypted payment or change, and a *Settled* note holds a quantity that an action measured. Third, it carries a 32-byte *policy attachment* (Section 6), which the Pool records under the note’s commitment but never uses to restrict spending.

Ledger state. Each deployment has a domain D bound to its chain and Pool address. The ledger consists of the note tree T , the spent-nullifier set N , a map J of action receipts, accounted asset reserves C , and the policy attachments. A spend proves membership under an *accepted root*: any nonzero root the Pool recorded after an append has been accepted, and the spent-nullifier set prevents replay.

Commitments. The circuits hash with the Poseidon permutation [16]. A holder’s spend secret determines a public *owner root* and a *nullifier key*, which recognizes the holder’s spends but cannot authorize them. Each note has a fresh secret tweak and an *owner tag*, which hashes the owner root with the tweak and therefore links to no receiving address. A note’s commitment hashes its domain, asset, value, owner tag, serial, randomness, and controller; its nullifier hashes the commitment under the nullifier key; and the tree stores a typed leaf that marks the note as Pending, Active, or Settled (Appendix A.1 gives each formula).

Private delivery. A recipient registers by publishing a receiving descriptor (its owner root, an X-Wing public key, and a controller) in Fuyu’s onchain *receive directory*. The sender encapsulates to the X-Wing key. The shared secret determines the new note’s tweak, and hence its owner tag, and also the key under which a Poseidon cipher encrypts the note’s opening. The recipient finds the note by decapsulating. A wallet derives all its keys from one random *seed*, and with the seed alone it rebuilds its whole account from public events: it decapsulates each output’s KEM ciphertext, and it re-derives the tweak of each deposit and settlement from a public *recovery salt* (Appendix A.3). The Pool also lets any address publish an encrypted profile and replace it only by compare-and-swap, so a wallet can keep its encrypted seed onchain. Directory keys and KEM ciphertexts contain X25519 values, but breaking X25519 yields only half of each X-Wing shared secret, so a quantum adversary still faces ML-KEM-768.

4.2 Operations

Value enters the Pool with a *DEPOSIT*, moves through any number of *TRANSFER* and *EXECUTE* operations, and leaves with a *WITHDRAW*

or, for a Pending note that cannot pass screening, a source-revealing exit: a public withdrawal of exactly that note, which reveals the deposit’s source and creates no private output. Each *EXECUTE* (Stage 1) is completed by a *FINALIZE* (Stage 2).

Deposit. The Pool collects exactly the stated amount of an admitted token or of the chain’s native asset and appends a Pending commitment that it computes from the amount, the source, the block, and an owner commitment that binds the owner tag and the controller that the depositor chose (Section 4.4).

Spends. Every other operation spends one or two whole notes with one proof and creates up to two notes, such as a payment and its change. Let v_0, v_1 be the inputs, y_0, y_1 the private outputs, v_{pay} a public payout, v_{fee} a fee, d an external input, and $\alpha \in \{0, 1\}$ the action flag. The circuit range-checks every amount and enforces

$$v_0 + v_1 = y_0 + (1 - \alpha)y_1 + v_{\text{pay}} + v_{\text{fee}} + d. \quad (1)$$

The relation $\mathcal{R}_{\text{spend}}(X; w)$ holds when the witness w opens the inputs under the accepted root in X , binds them all to one public controller, derives their nullifiers, satisfies the source policy for a Pending input (Section 5.1), and builds output commitments and ciphertexts that obey Equation (1). The statement X binds the domain, the root, the nullifiers, the outputs, and a digest of the complete request: the recipient, fee, relayer, deadline, the outputs’ policy attachments, and both outputs’ KEM ciphertexts, and for an action also its route, minimum return, and settlement owner. The Pool verifies the proof, rejects unknown roots, spent nullifiers, and expired requests, and applies the transition atomically.

Transfer and withdraw. A transfer appends its Active outputs in the same step as its spend, and a withdrawal pays v_{pay} to a public recipient and appends the private change. A stateless router can chain up to eight spends in one transaction, each proved against the root that the earlier ones create, so that a payment to several recipients, or a merge, lands completely or not at all.

Execute and finalize. For an action, the proof fixes the route, the exact input d , the minimum return q_{min} , and the settlement owner O (Section 4.4). In Stage 1, the Pool consumes the inputs, calls the adapter, requires an exact input debit and a measured output $q \geq q_{\text{min}}$, and stores the receipt $J[n_0] = (c_{\text{change}}, c_{\text{settled}})$ under the first input nullifier n_0 ; any failure reverts the whole transaction. In Stage 2, any caller appends the receipt’s commitments, and the Pool deletes the receipt. By default, a router runs both stages in one transaction (Algorithm 1 in Appendix A.2).

4.3 Controlled Notes

A note with a controller can be spent only with two independent authorizations: a valid spend proof, which needs the note’s opening, and the controller’s approval. The Pool checks an EIP-712 approval [9], bound to the chain and the Pool, over hashes of the exact proof, statement, request, and verification key. The controller is any wallet: an EOA signs the approval with its key, a contract wallet confirms it through ERC-1271, and a Safe checks it against its current owners and threshold [33], even with one owner; because the Pool passes the Safe the approval’s preimage, passkey and nested-Safe owners count toward its quorum. The gate covers every

spend, withdrawals and Execute included, and no call through a Safe or its modules bypasses it.

A Safe’s quorum is *current-state* authority: when the Safe changes its owners or threshold, later spends need the new quorum. Controlled and ordinary notes share one Pool and one note tree, so no Safe needs its own pool, but every spend reveals its controller, so a controller’s spends are linkable (Table 1). Each other owner of a Safe can check a proposed spend before signing it (Appendix A.2).

4.4 Measured Settlement and the Note Tree

The settlement owner O is an owner commitment that binds a fresh owner tag and the input’s controller. From the measured output q , its asset, a serial derived from n_0 , and O , the Pool computes the Settled note’s commitment c_{settled} (Equation (8) in Appendix A.1). The receipt thus binds the measured output to the authorized owner before any append, so no adapter or finalizer can change them.

Note tree. The note tree is a lean incremental Merkle tree of at most 2^{32} typed leaves, so an append hashes about $\frac{1}{2} \log_2 n$ times with n notes instead of at all 32 levels (Appendix A.1).

5 Accountability: Screening and Selective Disclosure

Source screening checks a deposit’s source when the deposit first moves privately, and selective disclosure answers an auditor’s questions on demand.

5.1 Source-Screened Deposits

The publisher posts source policies, each with a scheduled activation and expiry. A *source policy* consists of the root of an indexed interval tree over source addresses, for membership proofs in ALLOW mode or non-membership proofs in DENY mode, and a *coverage block*, the last deposit block that the review covers. A *screened* spend is any spend of a Pending note other than the source-revealing exit. Its proof shows that the deposit’s source satisfies the root and that the deposit block plus the deployment’s minimum delay is at most the coverage block, and the Pool checks that this policy is in force. Each republication of the policy, including a denial, advances its *epoch*, a sequence number that the proof binds. A screened proof made before a republication must therefore be made again and, for a controlled note, approved again. Screening checks a spend’s direct Pending inputs only, and only once; later notes carry no proof of ancestry. It resembles Privacy Pools’ association sets [11], but the publisher, not the user, picks the set.

5.2 Holder-Run Audits

An audit is a *program*, an FPL query (Section 6) that the auditor gives the holder. The holder proves the answers on its own device, and the auditor checks them with its own verifier against its own node, so no server receives a witness or a viewing key.

Two relations answer programs. The *notes relation* proves that the subject’s notes of a requested kind reach a threshold. For holdings, these are the notes held unspent at the question’s block; for receipts and deposits, the notes received or deposited. The *spend-records relation* does the same for the subject’s trades and withdrawals. Each proof has four public inputs: a hash of its plan, which

fixes its program, question, snapshot, and statement; an account tag, salted per program, that binds the proof to every key that the subject has registered (its *key generations*); and two blinded running totals that chain a fixed number of steps, so the number of notes stays hidden. No commitment, nullifier, amount, or position is public. A held note is proved unspent by non-membership in the set of spent nullifiers, which the verifier rebuilds from the Pool’s history, so the note’s later spend stays unlinked.

6 The Fuyu Policy Language

The lender of Section 2.2 does not want the treasury’s transactions; it wants answers to five questions. FPL is a query language for such questions, and it is how Fuyu meets challenge C3. Fixed disclosure tools cannot serve, because each auditor asks different questions and some answers are negative. An auditor writes a *query*, a set of named Yes/No questions about an account’s complete history, and gives it to the holder as a program identified by its digest; the holder answers each question with evidence. A *policy* can state consent in advance: it names the auditors who may ask, the classes of questions that each may ask, and the public facts that an answer may reveal. The Pool reserves one opaque attachment per note for a policy’s commitment and never interprets it (Section 4.1). Attaching a policy is optional, and a holder-run program consults no attachment, so for such a program consent is the holder’s choice to run it. No answer is given without the holder’s participation, and FPL never restricts what an account may do, so it can gain new question forms without changing the Pool. FPL gives three guarantees.

- *Certification typing.* The type system uses each question’s monotonicity to decide which answers the holder’s records certify and which need a certificate that the history is complete, so no omitted record can change a certified answer, and every answer states the evidence that it needs (Section 6.3).
- *Effect coverage.* Under a policy, every public fact an answer reveals is in the one grant that admits the question (Section 6.3).
- *Fail-closed lowering.* A verifier backend refuses a whole program unless its verifier certifies every question, and names each question that it cannot certify with the reason; the query plan names the evidence that every answer needs (Section 6.4).

Example. Figure 3 writes the treasury’s policy and the lender’s quarterly review in FPL’s surface syntax. The review asks the five questions, and a sixth, *cleared*, that combines four of them, about the treasury’s history through block 24,000,100 of the example chain. The first grant admits holdings questions such as the one that Bob answers in Figure 2, whether he holds at least m of token Y . The second grant admits all six questions, because naming the treasury adds its identity to what every answer reveals. On the example history, the reference front end answers Yes, No, Yes, Yes, Yes, and Yes, and the answers rest on different evidence (Section 9.3). Notes worth at least \$1M witness the first Yes, a vault-share note held on September 30 witnesses the third, and a credential witnesses the fifth. The No to the second question and the Yes to the fourth concern all deposits and trades, so only a complete history certifies them. Asked in a program of its own, the third question is proved with the notes relation (Section 5.2). For a Safe treasury, replaying

```

policy TreasuryDisclosure {
  permit auditor ask holdings expose manifest(CurrentView);
  permit auditor ask holdings, provenance, activity,
    identity expose manifest(AuditReview);
}

query QuarterlyReview for auditor about treasury
  at finalized(24000100) expires 1893456000 {
  over_1m: value(USD) >= 1_000_000 USD;
  sanctioned_funds: exists deposit where source in OFAC@17;
  held_sep_30:
    balance(stataUSDC) at 2026-09-30 >= 25_000 stataUSDC;
  within_limits: every day in 2026-07-01..2026-09-30:
    sum(trade.amount_in in USD) <= 50_000 USD;
  verified_entity: attested(KycIssuer, verified_entity);
  cleared: over_1m and not sanctioned_funds
    and within_limits and verified_entity;
}

```

Figure 3: The treasury’s disclosure policy and the lender’s quarterly review in FPL. Each question is answered Yes or No over the treasury’s complete history through the snapshot.

$p ::= \text{policy}(g)$	<i>policy</i>
$g ::= \text{grant}(u, C, E) \mid g \oplus g$	<i>grants</i>
$C \subseteq \{\text{hold, prov, act, id}\}$	<i>capabilities</i>
$Q ::= \text{query}(u, s, h, \eta, \delta, v, \vec{q})$	<i>query</i>
$q ::= x : \varphi$	<i>question</i>
$\varphi ::= \tau \triangleright c \mid \exists R.F \mid \neg R.F \mid \forall R.F \Rightarrow G$ $\mid \text{attested}(is, cl, t) \mid x$ $\mid \text{every}_P I. \varphi \mid \text{some}_P I. \varphi$ $\mid \neg \varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$	<i>windows</i>
$\tau ::= \text{bal}(a, F, t) \mid \text{val}(N, F, t)$ $\mid \text{count}(R, F) \mid \text{sum}(R.f, N, F)$	<i>holdings</i> <i>aggregates</i>
$F, G ::= f \triangleright v \mid f \in \vec{L} \mid f \notin \vec{L} \mid f \in I$ $\mid \neg F \mid F \wedge F \mid F \vee F$	<i>field tests</i>
$R ::= \text{deposit} \mid \text{trade} \mid \text{transfer} \mid \text{withdrawal} \mid \text{receipt}$	
$P ::= \text{hour} \mid \text{day} \mid \text{week} \mid \text{month}$	<i>periods</i>

Figure 4: Abstract syntax of FPL.

the Pool’s events by controller would list all of its trades and withdrawals (Appendix B.3). With this evidence, the lender would learn six Yes/No answers and the public facts that its grant names, and nothing about the treasury’s other transactions.

6.1 Syntax

Figure 4 gives the abstract syntax. A stored policy is fixed when a note is issued; a query arrives later, from an auditor. A grant names an auditor u , a set of capabilities C , and an effect set E , but no threshold, subject, or snapshot of any particular query. Questions are closed formulas without recursion, unbounded iteration, or access to any state outside the account’s history and the registered names below, so evaluating a question always terminates. Implication $\varphi \Rightarrow \psi$ abbreviates $\neg \varphi \vee \psi$. In this section, C, E, N, q, x , and w denote capabilities, effects, numeraires, questions, question names, and windows, not the ledger symbols of Section 4.

Names and bindings. Names resolve in a trusted registry: an admitted asset a ; a numeraire N such as USD, with one reviewed price feed per priced asset and a staleness bound; a union $\vec{L}$ of content-addressed address-list snapshots; a credential issuer is and a claim cl that it issues; an authenticated auditor u ; and a set E of named public effects, which a policy writes as a reviewed manifest such as `manifest(CurrentView)`. A query names its auditor u , an optional subject s (the account asked about), an optional policy digest h , a chain-final snapshot η , an expiry δ , and an optional nonce v . Each question $x : \varphi$ may refer to the answers of earlier questions.

6.2 History and Semantics

History. A query ranges over the history M of its subject’s account through the snapshot: every record since the Pool’s deployment. A subject’s account is every note owned under a receiving key that it registered in the receive directory through the snapshot, revoked keys included, or, for a Safe subject, every note that the Safe controls. M consists of five relations, one note set, and the subject’s issuer credentials; prices and list snapshots are shared registry inputs. Records carry their block and time. The relation deposit holds the source address, asset, and amount of each deposit credited to the account. The relation trade has one record per Stage-1 action that spends the account’s notes, with its venue, asset pair, input, and measured output (Stage 2 adds none). The relation transfer holds the private transfers received or sent, and withdrawal holds the withdrawals and exits with their recipients. The relation receipt holds every Active note that the account’s keys can spend, payments and change alike (a received note, unrelated to the action receipts of Section 4.2). The note set $\mathcal{H}$ records every note that the account ever held, with its asset, amount, origin, creation time, and spend time, and with $\text{src}(o)$, the public deposit sources that directly funded the creating event of note o . Funds that reached a note through earlier Active or Settled notes contribute no direct source, because the Pool screened them when they left Pending (Section 5.1). Times are Unix seconds of chain-final blocks, and a date names a whole UTC day.

Denotations. Let t be the evaluation point, initially the snapshot time, and w an optional window. Write $R(w)$ for R ’s records through the snapshot, limited to w when a window is present, and $\mathcal{H}(t)$ for the notes held at t : created at or before t and unspent at t . With t' an explicit time point or else t ,

$$\begin{aligned}
[[\exists R.F]] &= |\{r \in R(w) : F(r)\}| > 0, \\
[[\forall R.F \Rightarrow G]] &= \bigwedge_{r \in R(w), F(r)} G(r), \\
[[\text{count}(R, F)]] &= |\{r \in R(w) : F(r)\}|, \\
[[\text{sum}(R.f, N, F)]] &= \sum_{r \in R(w), F(r)} \text{px}_N(a_r, r.f, r.\text{time}), \\
[[\text{bal}(a, F, t')]] &= \sum_{o \in \mathcal{H}(t'), o.a=a, F(o)} o.\text{amount}, \\
[[\text{val}(N, F, t')]] &= \sum_{o \in \mathcal{H}(t'), F(o)} \text{px}_N(o.a, o.\text{amount}, t'),
\end{aligned} \tag{2}$$

where $\neg R.F$ is the negation of $\exists R.F$, a_r is the unit of field f in record r , and a sum without a numeraire adds amounts of one asset. The conversion $\text{px}_N(a, x, t) = \lfloor x \cdot pr \cdot 10^{d_N} / 10^{d_a + d_{pr}} \rfloor$, where d_a and d_N are the decimals of a and N , uses the latest answer pr , with d_{pr} decimals, of a ’s registered feed at or before t . Evaluation fails if that answer is older than N ’s staleness bound or if a converted asset

has no feed. $\text{attested}(is, cl, t')$ holds when the issuer's credential for claim cl names the subject and is valid and unrevoked at t' . A window $\text{every}_P I. \varphi$ is the conjunction of φ over the calendar-aligned P -periods of I , each clipped to I and evaluated at its last second; some_P is the disjunction. A question reference x denotes that question's answer at the snapshot. The connectives are strict, so every operand is evaluated and no failure is masked by another operand's value, and arithmetic is exact over unbounded integers.

6.3 Static Semantics

Units. Every term has a sort, such as an amount of one asset, a value in a numeraire, or a count, and a comparison's bound must have the term's sort, so a bound of 1 sAAPL on a USDC balance is a type error (Appendix B.3).

Capabilities and effects. Each question φ has the capabilities $C(\varphi)$ that its terms use: hold for balances and values, prov for deposits and sources, act for trades, transfers, withdrawals, and received notes, and id for credentials; a reference inherits those of the question that it names. Its effects $E(\varphi)$ collect the auditor and the snapshot, the deployment's floor effects of each capability (the public facts that every answer in that capability reveals), and one effect for each feature that it uses: a list digest, a price source, window bounds, an issuer, a named subject, and a history scope when an answer needs a complete history. The floor of hold is the public field set of a holdings proof (Section 5.2). A grant $g = \text{grant}(u', C, E)$ admits a question when it names the query's auditor and covers both sets:

$$\text{Admit}(g, \varphi) = [u' = u] \wedge [C(\varphi) \subseteq C] \wedge [E(\varphi) \subseteq E]. \quad (3)$$

Grants are alternatives: $g_1 \oplus g_2$ admits φ if g_1 or g_2 does, and no question combines the auditor of one grant with the capabilities of another. A resolved grant must also declare the floor effects of its capabilities, so no policy can declare an answer to be less revealing than the proof that carries it. Each answer reveals one bit and the public facts that $E(\varphi)$ names; a verifier checks every other part of a witness inside a proof or against data that $E(\varphi)$ already names. A grant bounds which questions an auditor may ask, not how many. Adaptively chosen thresholds reveal one bit each, so about $\log_2 B$ answers fix a balance of B atomic units; the holder, who takes part in every answer, decides how many to give.

Certification. The central design decision is to type each question by the evidence that certifies its answers. Let $\hat{M} \subseteq M$ be the records that the holder presents. Each is authenticated on its own, from public data or a proof, as a record of the subject's account, with every field of its record in M (a note's spend time and complete direct-source set, a credential's revocation status), and distinct presented records are distinct records of M . We call φ *upward* if $[[\varphi]](\hat{M}) \Rightarrow [[\varphi]](M)$ for every such $\hat{M}$ and every M on which $[[\varphi]]$ is defined, and *downward* if $\neg[[\varphi]](\hat{M}) \Rightarrow \neg[[\varphi]](M)$. A Yes to an upward question is certified by a *witness*, the presented records that make it true, and a No to a downward question by a counter-witness. Every other answer needs a certificate $\mathcal{R}_{\text{hist}}$ that $\hat{M} = M$ for the named subject's account. The judgment $\Gamma \vdash \varphi : \text{Ans}[\kappa^+, \kappa^-]$, where Γ is the resolved registry, records the method for the Yes (κ^+) and for the No (κ^-), with $\kappa \in \{\text{wit}, \text{cmp}\}$ and $\text{wit} \sqsubseteq \text{cmp}$; we call the pair the question's *profile*. Lower bounds, existence, and

credentials are upward:

$$\frac{\bowtie \in \{\geq, >\}}{\Gamma \vdash \tau \bowtie c : \text{Ans}[\text{wit}, \text{cmp}]} \quad \frac{}{\Gamma \vdash \exists R.F : \text{Ans}[\text{wit}, \text{cmp}]} \quad (4)$$

$$\frac{s \neq \perp}{\Gamma \vdash \text{attested}(\dots) : \text{Ans}[\text{wit}, \text{cmp}]}$$

The other atoms are negations of these: $\tau \leq c$ is $\neg(\tau > c)$, $\tau < c$ is $\neg(\tau \geq c)$, $\nexists R.F$ is $\neg\exists R.F$, and $\forall R.F \Rightarrow G$ is $\neg\exists R.(F \wedge \neg G)$. Negation exchanges the two methods, and a conjunction or disjunction takes, in each direction, the join of its operands' methods:

$$\frac{\Gamma \vdash \varphi : \text{Ans}[\kappa^+, \kappa^-]}{\Gamma \vdash \neg\varphi : \text{Ans}[\kappa^-, \kappa^+]} \quad (5)$$

$$\frac{\Gamma \vdash \varphi_i : \text{Ans}[\kappa_i^+, \kappa_i^-]}{\Gamma \vdash \varphi_1 \circ \varphi_2 : \text{Ans}[\kappa_1^+ \sqcup \kappa_2^+, \kappa_1^- \sqcup \kappa_2^-]}$$

where $\circ \in \{\wedge, \vee\}$. Comparisons with $=$ or $\neq$ have $\text{Ans}[\text{cmp}, \text{cmp}]$, windows take the profile of their body, and a reference takes the profile of the question that it names.

PROPOSITION 6.1 (WITNESS SOUNDNESS). *If $\Gamma \vdash \varphi : \text{Ans}[\text{wit}, \kappa^-]$, then φ is upward; if $\Gamma \vdash \varphi : \text{Ans}[\kappa^+, \text{wit}]$, then φ is downward.*

Appendix B.3 sketches the proof, by induction on the derivation.

For example, a balance threshold has the profile $\text{Ans}[\text{wit}, \text{cmp}]$, and an aggregate equality such as `count(trade) == 3` has the profile $\text{Ans}[\text{cmp}, \text{cmp}]$. In Figure 3, a Yes to `over_1m`, to `sanctioned_funds`, to `held_sep_30`, or to `verified_entity` is witnessed, and so is a No to `within_limits`, by a single day over the limit. The combination cleared mixes both directions, so only a complete history certifies its answers. Without a subject, $\mathcal{R}_{\text{hist}}$ has no referent, so the checker admits only witnessed answers and rejects a question that no witness can certify either way. When the holder presents no witness, the question stays unanswered.

6.4 Verification and Compilation

Verifiers. Each kind of evidence that an answer needs can be checked against published data: Pool events for notes, deposits, and spends, and registered feeds and issuers for prices and credentials (Table 6 in Appendix B.3). Of these, the notes and spend-records relations of Section 5.2 have verifiers, and each proves the Yes of a lower bound: an existence, a count or sum of at least a threshold, or a balance of at least an amount.

Holder-run acceptance. A question Q of a program lowers to parts, each a fixed chain of proofs of one relation. Write σ_β for the chain-final Pool state at the program's snapshot β ; for $\text{bal}(a, F, t) \geq m$, β is the last block at or before t . Let $\Phi_Q(\sigma_\beta)$ state that notes (or spend records) of the subject's registered keys reach Q 's thresholds: each is in σ_β 's note tree (or spend records), passes Q 's filters, and, for holdings, is unspent in σ_β , and no note counts twice. The verifier compiles the program itself, rebuilds the note tree, spent nullifiers, and spend records at β from its own node, reads the subject's key generations from the receive directory, and accepts π when

$$V_{\text{audit}}(Q, \pi, \sigma_\beta) = 1 : \quad (6)$$

every proof verifies under the pinned key against the plan hash that the verifier recomputes, every proof carries the subject's account

tag for this program, and each chain starts at zero, continues its blinded totals, and ends at the thresholds. Acceptance implies Φ_Q .

The answer certifies this Yes for the subject’s registered account: each counted note is owned under one of its keys, and each spend record matches both a nullifier of such a key and the account’s controller, so a Safe’s answer never counts personal spends of the same key. As with any key-based proof, parties who share a spend secret can pool notes in one answer (Appendix C.2).

Compilation. The FPL front end resolves names; checks units, capabilities, effects, grants, and profiles; and emits one of two artifacts. A *query plan* states, for each question, its profile, its admitting grant, the records, lists, price sources, issuers, time points, and complete-history scope that its answers need, and the record sets that no verifier proves. The *holder-run planner* lowers each supported lower bound to parts of the notes or spend-records relation and binds each plan to the program’s digest. It refuses a program with any other question, naming it and the reason (Appendix B.2), so it never emits a plan that its verifier cannot certify.

Verifier soundness. Every FPL verifier must be one-way sound: it accepts a complete answer only if it equals $[[\varphi]](M)$ for the complete history at η , and a witnessed answer only in the direction that the question’s profile certifies. The holder-run path meets this for the subject’s registered account, by Claim 7 and the proposition above.

7 Security Analysis

The protocol meets the goals of Section 3 under its assumptions. Each claim states why it holds; Appendix C gives the arguments.

Integrity. *Claim 1 (ownership):* an accepted spend consumes only notes whose openings the prover knows, and a controlled note only with its controller’s approval over the exact statement, for a genuine Safe its current quorum, by knowledge soundness and the Pool’s approval check. *Claim 2 (single consumption):* each opening determines one nullifier, which the Pool records atomically and never accepts twice. *Claim 3 (conservation):* Equation (1) accounts for every input, and an action’s output enters only as the Pool’s measured balance change. *Claim 4 (fixed settlement and delivery):* FINALIZE appends only the stored receipt, whose c_{settled} binds the measured q and the authorized owner (Equation (8)), and the spend relation ties each ciphertext to its commitment. *Claim 5 (solvency):* for each asset a , every transition preserves $Bal_a \geq U_a + Res_a$, custody against unspent notes and outputs reserved for finalization.

Privacy. *Claim 6 (declared leakage):* an operation reveals only the fields that Table 5 declares for it and its timing, because proofs are zero knowledge, commitments hiding, ciphertexts semantically secure, nullifiers pseudorandom, and settlement owner tags fresh. The claim also holds against a later quantum adversary if ML-KEM-768 is IND-CCA secure against it: owners and nullifiers are hashes, an X25519 value yields only half of an X-Wing secret, and signatures protect no note secret.

Disclosure. *Claim 7 (disclosure soundness):* an accepted answer to a holder-run program implies the statement Φ_Q of Section 6.4 for the subject’s registered account, and a rejected or missing answer implies nothing. *Claim 8 (certified answers):* an accepted witnessed

answer to φ equals $[[\varphi]](M)$ for the account that the witness authenticates, by the witness-soundness proposition of Section 6.3, and an answer whose type requires a complete history is accepted only with a certificate for the named subject.

8 Implementation

We implemented Fuyu as EVM contracts, Circom circuits with Groth16 keys for the spend relation and the two audit relations, a web application with a browser prover, a reference front end for FPL, and an audit library that proves and verifies the answers to holder-run programs. Admitted routes cover Uniswap V3 and several lending vaults, and adding an asset or a pair needs no change to the spend relation.

9 Evaluation

We compare Fuyu with the systems of Table 3 along three questions. *RQ1 (control):* How finely can holders and institutions control who may spend, complete, and read? *RQ2 (cost and security):* What do operations cost, and what remains of funds and privacy in the worst case by L2BEAT’s criteria? *RQ3 (expressiveness):* Which questions can an auditor ask in FPL, and which answers can a holder prove?

Setup. We ran every experiment on a 16-core Linux host (AMD EPYC 7B12). Fuyu’s 23 funded suites each deploy a fresh Pool on a copy of Ethereum mainnet at block 26,038,058 and run against the deployed Uniswap V3, Aave, sDAI, and Safe 1.4.1 contracts; all passed. The suites and their gas receipts predate EOA and ERC-1271 controllers and native-ETH action legs, which the measured operations do not use. Other systems’ facts come from their documentation, L2BEAT’s reviews [22], and mainnet transactions (Appendix D).

9.1 RQ1: Control

Only Fuyu pairs an optional second approval with a history that no one else reads (Table 3). Like Privacy Pools and Privacy Boost, it can bar an illicit deposit from private use at once (Section 3). Any party can complete an action by finalizing its receipt, whereas Aztec Connect’s actions depended on its operator [4]. In Tornado Cash, Privacy Pools, RAILGUN, and zkBob, one secret authorizes every spend, and Aztec Connect let any one of several keys spend [4, 40]; a shared viewing key shows an account’s whole history. Penumbra and Zcash split a key among owners on their own chains [18, 28], and Privacy Boost lets a smart account approve spends, while its operator’s enclave reads every note. A Fuyu spend reveals its controller, so a Safe’s spends are linkable (Table 1), which threshold keys hide.

9.2 RQ2: Cost and Security

L2BEAT rates link privacy against five adversaries: a public observer, a chain analyst who matches amounts and timing, a network observer, a privileged insider such as an operator or a governing body, and a future quantum adversary. It also records the *exit window*, the time users have to leave before an upgrade takes effect, and the trusted setup [22]. Table 3 gives its ratings and ours for Fuyu, which keeps links private against all but a privileged insider.

Table 3: Fuyu against privacy systems on Ethereum and OP Mainnet. ●: yes, or the link stays private; ○: partly, or at risk; ○: no, or exposed. Link privacy follows L2BEAT’s ratings [22] and, for Fuyu, our assessment; all five keep the link private against a public and a network observer.

	Tornado Cash [29]	Privacy Pools [11]	RAILGUN [31]	Privacy Boost [30]	Fuyu
<i>Control (RQ1)</i>					
Private transfers	○	○	●	●	●
Optional second approval	○	○	○	○	●
No one else reads history	●	●	●	○	●
Selective disclosure	○	○	○	○	●
Blocks illicit deposits at once	○	●	○	●	●
<i>Security (RQ2): link privacy against</i>					
A chain analyst	●	●	●	○	●
A privileged insider	●	○	○	○	○
A future quantum adversary	○	●	○	○	●
<i>Security (RQ2): the worst case</i>					
Infinite exit window	●	●	○	○	●
DeFi without an operator	○	○	●	○	●

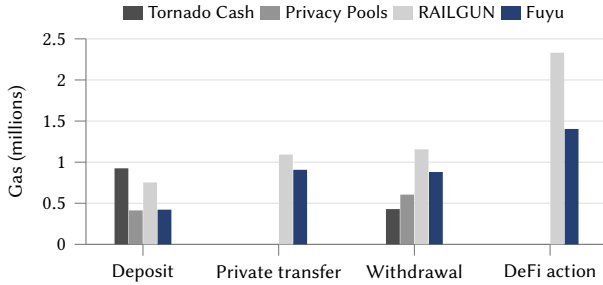


Figure 5: Gas per operation on Ethereum: each system’s median over recent mainnet transactions, and Fuyu’s funded-suite receipts on a fresh Pool with few notes; the text gives costs at a million notes. No bar: no such operation.

No role can pause, freeze, move, or reveal a note, and no upgrade changes a note in the Pool (Section 3), so the exit window is infinite. We rate Fuyu private against a future quantum adversary: while ML-KEM-768 holds, it learns no link from the hashes and X-Wing ciphertexts that Fuyu records. Separately, spend soundness rests on Groth16 over BN254 (Section 3). We rate Fuyu at risk against a privileged insider: its guardian or council can deny a deposit’s source at once, leaving the deposit only the public exit, and after seven public days the council can publish a source policy that narrows the deposits among which a first private spend hides; the genesis operator can do both at once. Denials are capped at 64 a day, which limits how many sources can be denied quickly but not

Table 4: Sixteen auditor questions (Appendix D): holdings (H), provenance (P), activity (A), identity (I). Yes, No: ● the holder’s records certify it, ○ only a complete history does. Holder-run: ● lowered to proofs of the notes or spend-records relation, ○ refused, naming its reason. Others: ● an association set answers it for one deposit, ○ only the full history does.

Question	Class	Yes	No	Holder-run	Others
funds_100k	H	●	○	●	○
held_sep_30	H	●	○	●	○
over_1m	H	●	○	○	○
kept_50k	H	●	○	○	○
under_cap	H	○	●	○	○
sanctioned_funds	P	●	○	○	○
clean_funds	H, P	●	○	○	○
deposited_1m	P	●	○	●	○
within_limits	A	○	●	○	○
withdrew_q3	A	●	○	●	○
active_trader	A	●	○	●	○
no_large_exit	A	○	●	○	○
paid_out	A	●	○	○	○
verified_entity	I	●	○	○	○
accredited	H, I	●	○	○	○
cleared	all	○	○	○	○

which; every change is public, and no policy reaches an already screened note. Fuyu’s Groth16 keys need a multi-party ceremony, whose participant count is TBD; L2BEAT counts 23 to 80 for the other four.

Fuyu’s DeFi action costs less than RAILGUN’s relayed venue calls (Figure 5): 1.40M gas for an Aave supply in two transactions, and 1.60M for a swap and supply in one, against RAILGUN’s median of 2.33M, mostly for aggregator swaps. Deposits and transfers cost less than in RAILGUN, and a deposit about as much as in Privacy Pools. Withdrawals cost more than in Tornado Cash and Privacy Pools: one spend relation with 39 public inputs serves every operation, and a withdrawal records a post-quantum ciphertext for its change. Appending costs more as the tree grows: at a million notes, an operation’s new notes add about 0.38M gas (Appendix D). A deposit would then cost 0.80M, still below Tornado Cash’s. A transfer would cost 1.28M and a withdrawal 1.26M, 17% and 9% above RAILGUN’s, and the two actions would cost 1.79M and 1.98M, still below RAILGUN’s.

9.3 RQ3: Expressiveness

FPL expresses all sixteen benchmark questions, and the holder-run planner lowers five to proofs; without disclosing a history, other systems answer at most the two source questions, with an association-set proof for one deposit [11, 32]. We modeled the questions on those that lenders ask and that FATF guidance asks virtual-asset service providers to check [13], including the review of Figure 3, and ran FPL’s checker and planner on them (Table 4).

Twelve are upward, so the holder’s records certify a Yes; three are downward, so only a complete history certifies their Yes; and the combined review needs a complete history for both answers. The planner lowers five Yes answers to proofs of the notes or spend-

records relation, which hide which records answer, how many, and their amounts. It refuses the other eleven by name: seven upward ones because no holder-run proof covers their evidence (a price, the spans of an **every** window, a source set, a transfer's opening, or a credential), and four that need a complete history.

With 16 threads, a holdings question over up to 64 notes proved in 298 s and a withdrawals question in 143 s, and the auditor's verifier checked each in 1.4 s. In the web application, capped at four CPUs, a holder proved both after a real deposit and withdrawal in 483 s, fetching the keys included.

10 Related Work

Shielded payments. Zerocash introduced notes, commitments, and nullifiers [8], Zexe generalized them to private computation over records [10], and Tornado Cash brought fixed-denomination mixing to Ethereum [29]. Deployed pools leak through timing and amounts [20], which is why Fuyu declares its leakage per operation. Most of them also record elliptic-curve keys or shares that a quantum adversary could use (Table 3), whereas Fuyu records owners and nullifiers as hashes, and its X-Wing delivery leaves such an adversary facing ML-KEM-768 [6].

Private DeFi. Hawk, Zapper, and zkay make contract data and state private [21, 35, 36]. Fuyu instead leaves public venues unmodified and keeps only ownership private. Baum et al. survey privacy-enhancing technologies in finance more broadly [7]. Aztec Connect batched deposits per bridge call [4], zkBob's operators submit its tree updates [40], and Penumbrabatches public swaps on its own chain [28]. RAILGUN's relay contract comes closest: it runs public calls against a private balance and re-shields its whole token balance in the same transaction [31]. Fuyu instead credits the measured output of one admitted action to the owner that the proof named, so a controller can co-approve the action, and asynchronous venues settle through the same receipts.

Accountable privacy. Garman, Green, and Miers add policy enforcement and selective tracing to Zerocash-style payments [15]. Association sets prove where funds came from [11, 32], and Fuyu's source screening applies the same idea when a spend consumes a deposit. zkLedger's aggregate answers are complete because its ledger holds a commitment for every bank in every transaction [25]. Fuyu's note tree offers no such completeness, which is why FPL types each answer by the evidence that certifies it.

Privacy languages. zkay checks privacy annotations in contracts, Zapper compiles private contracts for zero-knowledge execution, and ZQL compiles privacy-preserving queries [14, 35, 36]. vSQL proves SQL answers over outsourced data correct and complete [39], and work on incomplete databases finds the answers that hold in every completion [19]. FPL restricts no transaction: its types assign each answer, by monotonicity, the evidence that certifies it.

11 Conclusion

Fuyu shows that a privacy pool can keep payments private against a quantum computer, let no operator's or governor's absence stop private transfers, trades, or withdrawals, and let auditors ask rich questions while holders prove lower bounds on balances and activity without revealing records. Fuyu records each owner as a hash

and encrypts each payment with X-Wing, so recorded payments stay private while ML-KEM-768 holds. The Pool fixes each action's measured outcome in a receipt that anyone can finalize. The holder answers auditors with proofs, and FPL's types state the evidence that each answer needs. Of the systems that we compared, only Fuyu proves facts about balances and activity without revealing the records behind them, and its DeFi actions cost less than RAILGUN's, the only other pool that reaches venues without an operator. An optional controller, such as a Safe quorum, can co-approve each spend, and every upgrade ships as periphery or a successor Pool that holders choose.

References

- [1] 119th United States Congress. 2025. Guiding and Establishing National Innovation for U.S. Stablecoins Act (GENIUS Act). Public Law 119-27, 139 Stat. 419 (S. 1582). <https://www.govinfo.gov/app/details/PLAW-119publ27> Codified at 12 U.S.C. 5901 et seq.; approved July 18, 2025.
- [2] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. 2021. Uniswap v3 Core. Whitepaper. <https://app.uniswap.org/whitepaper-v3.pdf>
- [3] Guillermo Angeris, Alex Evans, and Tarun Chitra. 2021. A Note on Privacy in Constant Function Market Makers. arXiv:2103.01193. <https://arxiv.org/abs/2103.01193>
- [4] Aztec. 2023. Aztec Connect Specification: Claim Circuit and Rollup Contract. GitHub repository AztecProtocol/aztec-connect, https://github.com/AztecProtocol/aztec-connect/blob/9374aae687ec5ea01adeb651e7b9ab0d69a1b33b/markdown/specs/aztec-connect/src/claim_circuit.md Files claim_circuit.md and rollup_contract.md, last modified 2023-02-01; cited at commit 9374aae; see also the bridge README, <https://github.com/AztecProtocol/aztec-connect-bridges>.
- [5] Aztec Labs. 2023. Sunsetting Aztec Connect. The Aztec Labs Blog (Medium). <https://medium.com/aztec-protocol/sunsetting-aztec-connect-a786edce5cae> Published 13 March 2023 (UTC); dates as later stated in Aztec's documentation page "Aztec Connect Sunset", https://docs.aztec.network/aztec_connect_sunset.
- [6] Manuel Barbosa, Deirdre Connolly, João Diogo Duarte, Aaron Kaiser, Peter Schwabe, Karolin Varner, and Bas Westerbaan. 2024. X-Wing: The Hybrid KEM You've Been Looking For. *IACR Communications in Cryptology* 1, 1 (2024). doi:10.62056/a3qj89n4e
- [7] Carsten Baum, James Hsin-yu Chiang, Bernardo David, and Tore Kasper Frederiksen. 2023. SoK: Privacy-Enhancing Technologies in Finance. In *5th Conference on Advances in Financial Technologies (AFT 2023) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 282)*, Joseph Bonneau and S. Matthew Weinberg (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 12:1–12:30. doi:10.4230/LIPIcs.AFT.2023.12
- [8] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy (SP)*. IEEE, 459–474. doi:10.1109/SP.2014.36
- [9] Remco Bloemen, Leonid Logvinov, and Jacob Evans. 2017. EIP-712: Typed Structured Data Hashing and Signing. Ethereum Improvement Proposals, no. 712. <https://eips.ethereum.org/EIPS/eip-712>
- [10] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. 2020. ZEXE: Enabling Decentralized Private Computation. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 947–964. doi:10.1109/SP40000.2020.00050
- [11] Vitalik Buterin, Jacob Illium, Matthias Nadler, Fabian Schär, and Ameen Soleimani. 2024. Blockchain Privacy and Regulatory Compliance: Towards a Practical Equilibrium. *Blockchain: Research and Applications* 5, 1 (2024), 100176. doi:10.1016/j.bcr.2023.100176
- [12] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2020. Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 910–927. doi:10.1109/SP40000.2020.00040
- [13] Financial Action Task Force. 2021. Updated Guidance for a Risk-Based Approach to Virtual Assets and Virtual Asset Service Providers. FATF, Paris. <https://www.fatf-gafi.org/en/publications/Fatfrecommendations/Guidance-rba-virtual-assets-2021.html> Published 28 October 2021.
- [14] Cédric Fournet, Markulf Kohlweiss, George Danezis, and Zhengqin Luo. 2013. ZQL: A Compiler for Privacy-Preserving Data Processing. In *22nd USENIX Security Symposium (USENIX Security 13)*. USENIX Association, 163–178. <https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/fournet>

- [15] Christina Garman, Matthew Green, and Ian Miers. 2017. Accountable Privacy for Decentralized Anonymous Payments. In *Financial Cryptography and Data Security – 20th International Conference, FC 2016 (Lecture Notes in Computer Science, Vol. 9603)*, Jens Grossklags and Bart Preneel (Eds.). Springer, Berlin, Heidelberg, 81–98. doi:10.1007/978-3-662-54970-4_5
- [16] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schafneggger. 2021. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 519–535. <https://www.usenix.org/conference/usenixsecurity21/presentation/grassi>
- [17] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *Advances in Cryptology – EUROCRYPT 2016, Part II (Lecture Notes in Computer Science, Vol. 9666)*, Marc Fischlin and Jean-Sébastien Coron (Eds.). Springer, Berlin, Heidelberg, 305–326. doi:10.1007/978-3-662-49896-5_11
- [18] Daira-Emma Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. 2026. Zcash Protocol Specification. Version v2026.7.0-202-gafa086 [NU6.2]. <https://zips.z.cash/protocol/protocol.pdf> Dated September 22, 2026; build of the latest tagged release 2026.7.0 (2026-07-05); see Sec. 1.2 (High-level Overview) and Sec. 3.1 (Payment Addresses and Keys).
- [19] Tomasz Imieliński and Witold Lipski. 1984. Incomplete Information in Relational Databases. *J. ACM* 31, 4 (1984), 761–791. doi:10.1145/1634.1886
- [20] George Kappos, Haaron Yousaf, Mary Maller, and Sarah Meiklejohn. 2018. An Empirical Analysis of Anonymity in Zcash. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 463–477. <https://www.usenix.org/conference/usenixsecurity18/presentation/kappos>
- [21] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 839–858. doi:10.1109/SP.2016.55
- [22] L2BEAT. 2026. Privacy. <https://l2beat.com/privacy/summary>
- [23] L2BEAT Research. 2026. Onchain Privacy Best Practice. <https://l2beat.com/publications/privacy-best-practices>
- [24] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. 2013. A Fistful of Bitcoins: Characterizing Payments Among Men with No Names. In *Proceedings of the 2013 Internet Measurement Conference (IMC)*. ACM, New York, NY, USA, 127–140. doi:10.1145/2504730.2504747
- [25] Neha Narula, Willy Vasquez, and Madars Virza. 2018. zkLedger: Privacy-Preserving Auditing for Distributed Ledgers. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, 65–80. <https://www.usenix.org/conference/nsdi18/presentation/narula>
- [26] National Institute of Standards and Technology. 2024. Module-Lattice-Based Key-Encapsulation Mechanism Standard. Federal Information Processing Standards Publication 203. doi:10.6028/NIST.FIPS.203
- [27] Marilyne Ordekian, Gilberto Atondo-Siu, Alice Hutchings, and Marie Vasek. 2024. Investigating Wrench Attacks: Physical Attacks Targeting Cryptocurrency Users. In *6th Conference on Advances in Financial Technologies (AFT 2024) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 316)*, Rainer Böhm and Luciana Kiffer (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 24:1–24:24. doi:10.4230/LIPIcs.AFT.2024.24
- [28] Penumbra Team. 2025. Decentralized Exchange (ZSwap). The Penumbra Protocol (protocol specification). <https://protocol.penumbra.zone/main/dex.html> See also the Batch Swaps page, <https://protocol.penumbra.zone/main/dex/swap.html>.
- [29] Alexey Pertsev, Roman Semenov, and Roman Storm. 2019. Tornado Cash Privacy Solution, Version 1.4. Whitepaper. https://web.archive.org/web/20201105064118/https://tornado.cash/Tornado.cash_whitepaper_v1.4.pdf Dated December 17, 2019; originally published at tornado.cash (now offline); archived copy.
- [30] Privacy Boost. 2026. Auditability. Privacy Boost Documentation. <https://docs.privacyboost.io/product/auditability>
- [31] RAILGUN. 2025. Cross-Contract Calls. RAILGUN Developer Guide. <https://docs.railgun.org/developer-guide/wallet/transactions/cross-contract-calls>
- [32] RAILGUN. 2026. Private Proofs of Innocence. RAILGUN Wiki. <https://docs.railgun.org/wiki/assurance/private-proofs-of-innocence>
- [33] Safe Ecosystem Foundation. 2023. Safe Smart Account v1.4.1 (contracts/Safe.sol). GitHub repository [safe-fndn/safe-smart-account](https://github.com/safe-fndn/safe-smart-account), tag v1.4.1. <https://github.com/safe-fndn/safe-smart-account/blob/v1.4.1/contracts/Safe.sol#L257-L334> Tag v1.4.1 (commit bf943f8); checkSignatures and checkNSignatures at lines 257–334, approveHash at lines 342–346.
- [34] Joey Santoro, t11s, Jet Jadeja, Alberto Cuesta Cañada, and Señor Doggo. 2021. ERC-4626: Tokenized Vaults. Ethereum Improvement Proposals, no. 4626. <https://eips.ethereum.org/EIPs/eip-4626>
- [35] Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin Vechev. 2019. zkay: Specifying and Enforcing Data Privacy in Smart Contracts. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, New York, NY, USA, 1759–1776. doi:10.1145/3319535.3363222
- [36] Samuel Steffen, Benjamin Bichsel, and Martin Vechev. 2022. Zapper: Smart Contracts with Data and Identity Privacy. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, New York, NY, USA, 2735–2749. doi:10.1145/3548606.3560622
- [37] U.S. Department of the Treasury. 2022. U.S. Treasury Sanctions Notorious Virtual Currency Mixer Tornado Cash. Press release JY0916. <https://home.treasury.gov/news/press-releases/jy0916> Released August 8, 2022.
- [38] Hongbo Wen, Hanzhi Liu, Yanju Chen, Jingyu Ke, Dahlia Malkhi, and Yu Feng. 2025. Thunderbolt: Fast Asynchronous Off-Chain Bitcoin Transfers. Cryptology ePrint Archive, Paper 2025/709. <https://eprint.iacr.org/2025/709>
- [39] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. vSQL: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 863–880. doi:10.1109/SP.2017.43
- [40] zkBob. 2024. ZkBobPool Contract (src/zkbob/ZkBobPool.sol). GitHub repository [zkBob/zkbob-contracts](https://github.com/zkBob/zkbob-contracts), commit 1677f1e. <https://github.com/zkBob/zkbob-contracts/blob/1677f1ef45eed83fb4397fae96c87ad869bcc131/src/zkBob/ZkBobPool.sol> Commit 1677f1ef45eed83fb4397fae96c87ad869bcc131 (2024-09-10); pending-commitment queue and operator-only proveTreeUpdate().

A Protocol Details

A.1 Encodings

The circuits hash with Poseidon: $H_k(z_1, \dots, z_k)$ absorbs k field elements into a zero initial state and returns the first output element, and integer tags separate what is hashed. A holder’s spend secret k determines the owner root $R = H_3(218, k, 0)$ and the nullifier key $nk = H_3(219, k, 0)$, and a note with secret tweak t has the owner tag $o = H_3(220, R, t)$. An Active note with asset a , value v , serial s , randomness ρ , and controller ctl , a 160-bit address, in domain D has commitment c and nullifier n :

$$\begin{aligned} \bar{c} &= H_3(202, D, a, v, o, s, \rho, 0), & c &= H_3(216, \bar{c}, ctl), \\ n &= H_4(224, D, nk, c). \end{aligned} \quad (7)$$

A Pending commitment binds $(D, a, v, \text{source}, \text{block}, s, O)$ under tag 201 for the owner commitment $O = H_3(215, H_3(217, o, \rho), ctl)$, so the controller is bound into every note type, including ordinary notes whose controller is zero. For a Settled note, the Pool computes the serial $s' = H_3(214, D, n_0)$ from the first input nullifier n_0 and, for a measured output q of asset b , evaluates from receipt data

$$\begin{aligned} \zeta_0 &= H_3(213, D, b), & \zeta_1 &= H_3(\zeta_0, q, s'), \\ c_{\text{settled}} &= H_3(\zeta_1, O, 0). \end{aligned} \quad (8)$$

The tree stores the typed leaf $\ell = H_3(203, ty, c)$, with $ty \in \{0, 1, 2\}$ for Pending, Active, and Settled notes. Its nodes are $H_3(204, l, r)$; a node without a right sibling rises unchanged, so an append hashes once per set bit of its position, and the empty tree’s root is 0, which the Pool never accepts. A membership path marks the levels that have a sibling, and leaves and nodes hash under distinct tags, so a path that reaches the root starts at a leaf.

A.2 Operations

Statements. The request digest of Section 4.2 also covers the Pool domain, the asset, the payout, the nullifiers, the output commitments, the exit fields, and, for a screened spend, the source policy’s epoch. An action’s digest also fixes its change and the policy attachments of the change and the settlement, and its request carries one parameter word for the adapter, which lets a venue take a user parameter without a new Pool; every reviewed adapter requires it

Algorithm 1 Executing and finalizing one admitted action.

```

1: procedure EXECUTE( $\pi_{\text{spend}}, X, A$ )
2:   Decode  $(a, b, d, q_{\min}, O, v_{\text{fee}}, n_0, n_1, c_{\text{change}})$  from  $X, A$ 
3:   require SpendOK( $\pi_{\text{spend}}, X, A, T, N$ )
4:   require ControllerOK( $X, A$ )
5:   require  $a \neq b$ , RouteOK( $A, a, b$ )
6:   require  $C[a] \geq d + v_{\text{fee}}$ ,  $J[n_0] = \emptyset$ 
7:   require capacity for  $1 + [c_{\text{change}} \neq 0]$  leaves
8:    $N \leftarrow N \cup \{n_0, n_1\}$ ;  $C[a] \leftarrow C[a] - d - v_{\text{fee}}$   $\triangleright$  before the call
9:    $(\hat{b}, \hat{q}) \leftarrow \text{CallAdapter}(A, d, q_{\min})$ 
10:  Measure input debit  $\Delta_a$  and output credit  $q$ 
11:  require  $\Delta_a = d$ ,  $\hat{b} = b$ ,  $\hat{q} = q$ 
12:  require  $0 < q \leq 2^{128} - 1$ ,  $q \geq q_{\min}$ 
13:   $C[b] \leftarrow C[b] + q$ 
14:   $c_{\text{settled}} \leftarrow \text{SettledCM}(D, b, q, n_0, O)$ 
15:   $J[n_0] \leftarrow (c_{\text{change}}, c_{\text{settled}})$ ; reserve leaf slots
16:  Pay fee  $v_{\text{fee}}$  to the relayer
17:  return  $J[n_0]$   $\triangleright$  neither output is spendable
18: end procedure
19: procedure FINALIZE( $j$ )
20:  require  $J[j] = (c_{\text{change}}, c_{\text{settled}})$  exists
21:  Delete  $J[j]$ ; release its reserved slots
22:  Append  $(1, c_{\text{change}})$  if nonzero, then  $(2, c_{\text{settled}})$  to  $T$ 
23:  Record the policy attachments fixed in Stage 1 for those leaves
24:  Remember the new root; leave  $N, C$  unchanged
25: end procedure
26: procedure SETTLE( $\pi_{\text{spend}}, X, A$ )  $\triangleright$  the router's default
27:   EXECUTE( $\pi_{\text{spend}}, X, A$ )
28:   FINALIZE( $n_0$ )  $\triangleright$  in the same transaction
29: end procedure

```

to be zero. The Pool computes the settlement owner O from the request and the input's public controller, so the settlement and the change keep that controller, while each payment output carries its recipient's declared controller. Because the proof fixes the owner tag without proving knowledge of its opening, the wallet derives the tweak from its seed and the public recovery salt.

Deposits. An address's first deposit publishes its encrypted profile (Section A.3), and the Pool records a pointer to it, the profile's block number and ciphertext hash. Each later deposit names the current pointer, so that a wallet funds only the profile that it opened, and the next deposit nonce, so that a retry deposits at most once.

The action algorithm. In Algorithm 1, SpendOK covers the spend proof, the accepted root, unused and distinct nullifiers, the source policy, the action digest, and the deadline. ControllerOK checks a nonzero controller's approval, for a Safe its current quorum. RouteOK requires admitted exact-transfer assets, the published route and runtime code hash, and an adapter bound to this Pool. The Pool marks the nullifiers and debits the input reserve before the external call (line 8), under a reentrancy guard, and any later failure reverts the whole transaction. FINALIZE re-checks neither admission nor approval, because it spends nothing, and it cannot change the owner or the amount.

Balance. All terms of Equation (1) are in the input asset. An ordinary operation has $\alpha = d = 0$, and an action has $\alpha = 1$ and $v_{\text{pay}} = y_1 = 0$, so y_0 is its change. The second input may be con-

strained padding with a distinct nullifier, which the Pool also marks, and zero output commitments are never appended.

Delivery binding. The circuit takes each output's KEM shared secret as a private input, derives the tweak and the symmetric key from it, and requires the public Poseidon ciphertext to decrypt to the output's asset, value, randomness, and controller and to the request or action digest. The digest in turn binds the KEM ciphertexts, one for each nonzero output, so no relayer can replace or reorder them. A valid commitment therefore cannot be paired with an unrelated delivery payload, and a sender that encapsulates to a wrong key can only lose its own payment.

Validity. A request's deadline may be the largest 256-bit value, so that its proof stays valid until it executes or one of its inputs is spent, since a failed execution spends nothing. A screened proof lapses when the source policy is republished, even only to change its expiry (Section 5.1), and a Safe approval, which the Pool checks against the live Safe, fails once a signing owner is removed or the threshold rises above its signatures, but not when an owner joins or the threshold falls.

Controller approval. The signed EIP-712 data holds the Pool domain, the operation, the controller, and hashes of the complete encoded proof, statement, request, sender capsule, and verification key, so even a proof regenerated for a newer root needs new approvals. The Pool verifies an EOA's signature itself, asks a contract wallet through ERC-1271 only when the approval says so, and otherwise hands this preimage to a Safe-compatible threshold-signature interface in a static call; a rejected route never falls back to another. A *co-signer*, any other owner of the Safe, can check a proposed spend from a restricted opening of each private output, without the spend secret or the seed: it re-encapsulates, recomputes the commitment and both ciphertexts, and checks the proof.

A.3 Recovering Notes from Public History

A wallet reconstructs its account from authenticated Pool events with its seed, which determines the spend secret, owner root, nullifier key, X-Wing key pair, and viewing material. It credits a Pending note only if the owner commitment, note commitment, and typed leaf that it re-derives from the deposit's recovery salt agree, and an Active output only if the commitment and typed leaf that it recomputes from the decrypted opening, with the serial derived from the spend's first nullifier and the output's index, match the Pool's. A sender capsule carries both outputs' shared secrets for the sender's own recovery, and a Settled note is rebuilt from the recorded recovery salt and owner tag once Stage 2 is chain-final. Public data cannot restore an opening, so the seed is the holder's only path back to its notes. A key rotation replaces the address's profile with one that seals up to 32 earlier seeds under a key derived from the new seed, so the newest seed reopens every key generation, while an older seed reveals no newer one.

A.4 Payments to Unregistered Addresses

A *claim link* pays a wallet without a receiving descriptor through a private transfer to a one-time descriptor derived from a secret in the link. Its controller, a claim contract that only a claim or a refund deploys, names the wallet, a one-time refund address, and

the deadline, and spending needs the link’s secret and the wallet’s signature before the deadline or the refund signer’s from then on. A deposit *Portal* gives an account a fixed ERC-20 address that names no owner; a permissionless sweep turns plain transfers to it into Pending notes against invoices published in advance, and their screening rests on the publisher’s classification of the Portal.

B Leakage and Disclosure Details

B.1 Declared Leakage

Table 5 extends Table 1 to every field of the onchain record; network metadata, such as IP addresses, lies outside it. The *source-policy context* of a screened spend consists of the root, mode, and coverage block of its policy and the deployment’s minimum delay, and it narrows one of the spend’s inputs to the unspent Pending notes that the policy covers. A controlled action’s change and settlement inherit its public controller, so its venue actions and amounts are attributable to it, whereas a recipient’s controller stays hidden until the output is spent, unless the receive directory lists the recipient. A policy attachment reveals only whether it is zero.

B.2 Holder-Run Audits

The compiler lowers each question to one part per conjunct and turns a strict bound $> m$ into $\geq m + 1$. Each part’s plan names the program’s SHA-256 digest, the question and part, the relation and its key’s digest, the snapshot, and the statement. A step proves four notes or eight spend records, so 16 or 8 steps reach the recommended cap of 64, and strictly increasing positions keep any record from counting twice. The account tag is salted with the program’s digest and nonce, so answers to two programs cannot be linked.

The compiler refuses a program without a nonce or a subject, and every question that needs a complete history: a **no** or **all** quantifier, a negation, or a comparison other than a lower bound. It also refuses disjunctions and implications, since proving one side would reveal which side holds; windows, since a proof for one span would reveal that span and an **every** window would need a proof chain per span; and priced values and sums, source and list filters, $\neq$ in a filter, transfers, attestations, and count bounds above 65,535.

B.3 FPL Details

Sorts and field tests. A balance of asset a has sort $\text{Amount}[a]$, a value or a sum in a numeraire N has sort $\text{Value}[N]$, and a count has sort Count ; decimals must convert exactly to atomic units. List tests on direct sources are affirmative: $\text{src} \in \tilde{L}$ holds when the note has at least one direct source and all of them lie in $\tilde{L}$, and $\text{src} \notin \tilde{L}$ when it has at least one and none of them does, so a note without a direct source satisfies neither.

Witness soundness. The proposition of Section 6.3 follows by induction on the derivation, and on question order for references. A field test reads one record, and a presented record carries every field of its record in M , so each filtered set $\{r \in R(w) : F(r)\}$ and each held set $\mathcal{H}(t)$ can only grow from $\hat{M}$ to M , and so can existence, counts, and valid credentials. Amounts and prices are non-negative, so sums, balances, and values can only grow as well; a lower bound that holds on $\hat{M}$ therefore holds on M , and an upper bound that fails on $\hat{M}$ fails on M . Negation exchanges the two properties. Con-

Table 5: Every field that each operation makes public.

Operation	Publicly revealed
Every transaction	Submitting account and timing
Every spend	Nullifiers, accepted root, fee, relayer, deadline, sender capsule, spending controller and its approval (a Safe’s owner signatures or a claim link’s signature), and the Pending bit with, when it is set outside an exit, the source-policy context
Deposit	Depositor, asset, amount, owner commitment, recovery salt, policy attachment, and either the encrypted profile or its pointer and deposit nonce
Transfer	Output commitments, KEM and Poseidon ciphertexts, and policy attachments; with a relayer fee, also the asset and the fee
Router batch	Its spends share one transaction, and each later spend proves against a root that only the earlier spends’ outputs produce, which links them
Execute	Adapter and route, asset pair, exact input, minimum return, measured output, change commitment and ciphertexts, settlement owner tag, parameter word, recovery fields, and the change and settlement attachments
Finalize	No new note data; the receipt was fixed at Execute
Withdraw	Recipient, asset, amount, change commitment, ciphertext, and policy attachment
Exit	Deposit source, asset, payout recipient and amount, fee
Claim or refund	The claim controller’s deployment (named wallet, refund address, deadline) and the approving signature, whose time tells a claim from a refund
Portal payment	Payer, asset, and amount; the sweep is a deposit whose depositor is the Portal, and a published invoice reveals its asset, amount range, owner commitment, and salt
Profile replacement	Owner, the new encrypted profile, and the number of earlier key generations that it carries
Receive directory	On registration: the address, its owner root, X-Wing public key, and controller, now publicly linked; a revocation reveals the address

junction, disjunction, and windows, which are finite conjunctions or disjunctions, preserve each property of their operands.

Permissions. Under a policy, a record may back an answer only if the attachment of every note that it creates or spends opens to a policy that admits the question for this auditor, and a zero attachment admits nothing. A complete answer is refused unless every record that it reads is permitted, since the permitted remainder could omit the record that would change it.

Verifier soundness. One-way soundness (Section 6.4) bounds what a verifier accepts. *Coverage*, meaning that every true answer has an accepted certificate, is a separate property that depends on the evidence paths of Table 6, and a question whose evidence no verifier checks stays unanswered.

Complete histories. Any verifier can list every public record for a complete-history certificate by replaying the Pool’s events up to a chain-final block, without a root stored by the Pool. For a Safe

Table 6: Verification paths for the evidence of FPL answers. The first three rows have verifiers; the others list the data a verifier would read.

Evidence	Check	Published data
Notes held at t	Notes relation, Equation (6)	Note root, spend events, key directory
Receipts, deposits	Notes relation	Note root, key directory
Trades, withdrawals	Spend-records relation	Spend events, key directory
Deposit sources	Deposit record and opening	Deposit events
Complete Safe spends	Replay by controller	Spend events
Prices, credentials	Feed answer; issuer signature	Feed and issuer registries
Other histories	History relation	All Pool events

subject, replay alone completes the trade and withdrawal relations, because every spend publishes its controller, and actions and withdrawals publish their amounts (Table 1). A history relation would classify every replayed record under every key that the subject registered, with no change to the Pool.

Attachments. A note’s policy attachment is

$$C_p = \text{Keccak256}(\text{abi.encode}(\text{tag}, \text{chain}, \text{Pool}, h_p, \text{salt}))$$

for a fresh nonzero salt and the SHA-256 digest h_p of the resolved policy p , which names its language version. A query’s digest excludes the holder’s policy, so both sides agree on the questions before the policy is revealed.

C Security Details

C.1 Assumptions

We assume that each relation’s Groth16 setup was run so that no adversary knows its trapdoor, for example through a multi-party ceremony, and that Groth16 is then knowledge-sound, as Groth proves in the generic group model [17]. We also assume binding and hiding note commitments, an IND-CCA-secure KEM and a secure symmetric layer for delivery, and a host chain that provides an authenticated final order. Privacy against a later quantum adversary also assumes that ML-KEM-768 stays IND-CCA secure against it. Holders keep their seeds secret, also from a later quantum adversary; how a wallet stores its seed lies outside the protocol. Admitted assets are exact-transfer tokens, with no transfer fee or rebasing, and balances are accounted faithfully; no contract can prove an issuer’s offchain backing. Nullifiers are pseudorandom in the nullifier key nk , and Poseidon is collision-resistant, so each tree root binds its leaves.

Clients do not trust relays or indexers; they replay Pool history from an RPC endpoint that they trust to report chain-final state, and they credit only chain-final outputs. FPL answers also trust the auditor’s registry: its price feeds and staleness bounds, list snapshots, credential issuers and their status services, and the bindings of auditor and subject names. Services that the adversary controls can delay requests but gain no spending authority. Adapters and

assets that violate their admission assumptions are outside the model. The integrity and disclosure properties assume that commitments, ciphertexts, and recovery salts stay available onchain and that verifiers read the current nullifier set.

Governance. The publisher is trusted to classify sources and to admit only adapters that behave as reviewed. After a proposal’s delay, anyone may execute it within 14 days, and the genesis operator can end the genesis window early. The 64 daily denials count per UTC day, so up to 128 can fall within 24 hours across midnight. Only a proposal or the genesis operator lifts a denial or revokes a protected exit route, such as a vault redemption. A reviewer can extend the policy’s coverage and set its expiry within fixed bounds. Governance can neither pause nor reorder transfers, withdrawals, exits, or finalization, nor change a verifier or the note format.

Liveness. A relayer can refuse a request, an asset issuer can restrict transfers out of the Pool, as regulated stablecoin issuers must be able to [1], and a stalled source policy leaves Pending deposits only the exit; each delays progress without violating safety. No service is needed to leave: a holder with its seed and any RPC endpoint can recover its notes, finalize its actions, and withdraw.

C.2 Arguments

Claim 1 (ownership). By knowledge soundness, an accepted proof implies an opening of each real input under an accepted root (Equation (7)). Every real input carries the public spending controller. For a nonzero controller, the Pool checks the controller’s approval (for a Safe, its quorum) over hashes of the exact proof, statement, and request before it applies the transition.

Claim 2 (single consumption). Each opening determines one canonical nullifier. The Pool rejects any nullifier already in N and records the new ones in the same atomic step as the rest of the transition. A later proof against an older accepted root therefore still fails, because the nullifier is already marked.

Claim 3 (conservation). A deposit credits exactly the quantity that the Pool collects. Equation (1), with every amount range-checked, accounts for the input debit, fee, payout, and private change in the input asset. For an action, the Pool admits the output asset only through its own measured balance change q . With exact-transfer tokens, measured changes equal the amounts that actually moved. An adapter or token that violates these assumptions falls outside the argument.

Claim 4 (fixed settlement and delivery). Stage 1 computes c_{settled} from the measured q and the proof-authorized owner (Equation (8)) and stores it with the change commitment in $J[n_0]$. FINALIZE checks those commitments against the stored receipt, appends them, and deletes the receipt, so a finalizer can choose neither another quantity nor another owner, and no receipt can be appended twice. Delivery follows from the delivery binding of the spend relation (Section A.2), which ties each ciphertext to its commitment.

Claim 5 (solvency). The ledger records accounted reserves $C[a]$, whereas actual token balances may also include unallocated donations. A deposit raises custody and Pending liabilities by the same amount. An ordinary spend redistributes liabilities or pays them

out. An action lowers the input asset’s custody and liabilities by the input debit and the fee, and raises the output asset’s custody and reservations by the measured q . Finalization moves the reserved change and output into U_a and U_b without counting them twice, so every transition preserves $Bal_a \geq U_a + Res_a$ for each asset a .

Claim 6 (declared leakage). Groth16 proofs are zero knowledge, so the spend and audit proofs reveal nothing beyond their public statements. Note commitments are hiding, recipient ciphertexts are semantically secure, nullifiers are pseudorandom in the nullifier key, and settlement owner tags are fresh, since wallets derive a fresh tweak for each action. Every other value that an operation makes public, in its calldata, its public statement, or its events, is either a field that Table 5 lists for that operation or a function of such fields and the public order of transactions. The spend verifier’s input layout and the Pool’s event definitions fix these fields. An external action publishes the fields from which its settlement is built, so its privacy rests on the secrecy of the tweak, which keeps the owner tag unlinked to the holder’s root and to its other activity. Anonymity therefore depends on how the Pool is used, not only on the size of its tree.

The claim also holds against a later quantum adversary if ML-KEM-768 is IND-CCA secure against it. Owner roots, owner tags, commitments, and nullifiers are Poseidon images; each payment’s opening is a Poseidon ciphertext under a key derived from an X-Wing shared secret, which stays secret while either of X-Wing’s components is secure [6]; and sender capsules use AES-256-GCM. An encrypted profile is opaque to the Pool, and the seeds that it holds stay secret by the assumption of Section C.1. Directory keys and KEM ciphertexts hold X25519 values, but breaking X25519 yields only half of each X-Wing shared secret, which leaves the adversary facing ML-KEM-768, and controller approvals and transaction signatures authorize actions but protect no note secret. Soundness is not claimed against such an adversary.

Claim 7 (disclosure soundness). By knowledge soundness of the notes and spend-records relations, an accepted chain implies openings of distinct notes, or spend records, that pass the statement’s filters and reach its thresholds, each owned under a key whose owner root is one of the generations hashed into the account tag and, for holdings, whose nullifier lies outside the spent set. The verifier’s checks (V_{audit} in Equation (6)) supply the rest of Φ_Q : the plan hash that it recomputes binds the program, question, part, snapshot, and statement; the roots that it rebuilds from its own node make the trees and the spent set the Pool’s at the snapshot; and the generations that it reads from the receive directory make the tag the subject’s. A rejected or missing answer carries no information: a holder may decline, and any failed check rejects.

Claim 8 (certified answers). The witness-soundness proposition of Section 6.3 shows that an upward question that holds on authenticated records holds on the complete history, and that a downward question that fails on them fails on it. The checker assigns every other answer the method `cmp`, and each backend lowers only questions that its relation certifies (Section 6.4). Under $\mathcal{R}_{\text{hist}}$ the presented records are all of M , so the answer computed from them is $[[\varphi]](M)$. For balance thresholds, Claim 7 supplies the authenticated record.

```
query Benchmark for auditor about treasury
  at finalized(1020) expires 1893456000 nonce 0x7777 {
    funds_100k: balance(USDC) >= 100_000 USDC;
    held_sep_30: balance(USDC) at 2026-09-30 >= 25_000 USDC;
    over_1m: value(USD) >= 1_000_000 USD;
    kept_50k: every day in 2026-07-01..2026-09-30:
      balance(USDC) >= 50_000 USDC;
    under_cap: balance(USDC) <= 5_000_000 USDC;
    sanctioned_funds: exists deposit where source in OFAC@17;
    clean_funds: balance(USDC where
      source not in OFAC@17) >= 10_000 USDC;
    deposited_1m: sum(deposit.amount
      where asset == USDC) >= 1_000_000 USDC;
    within_limits: every day in 2026-07-01..2026-09-30:
      sum(trade.amount_in in USD) <= 50_000 USD;
    withdrew_q3: sum(withdrawal.amount where asset == USDC
      and time in 2026-07-01..2026-09-30) >= 10_000 USDC;
    active_trader: count(trade) >= 10;
    no_large_exit:
      no withdrawal where amount >= 1_000_000 USDC;
    paid_out: sum(transfer.amount where direction == out
      and asset == USDC) >= 1_000 USDC;
    verified_entity: attested(KycIssuer, verified_entity);
    accredited: attested(KycIssuer, accredited_investor)
      and over_1m;
    cleared: over_1m and not sanctioned_funds
      and within_limits and verified_entity;
  }
```

Figure 6: The sixteen benchmark questions of Table 4 in FPL. The nonce, 32 bytes in the program, is shortened.

Scope of a disclosure. An accepted answer binds keys, not a person: parties who share a spend secret can pool their notes in one answer, and for an account absent from the receive directory, the holder declares its key generations, so the answer shows only that the holder controls them. It says nothing about the account’s other notes and does not establish Safe spending authority.

D Evaluation Details

Gas. For the other systems, Figure 5 shows medians over their latest successful mainnet transactions of each kind up to block 26,067,900: 100 deposits and 100 withdrawals through Tornado Cash’s router, 52 ETH deposits and 105 relayed withdrawals through Privacy Pools’ entry point, and 64 RAILGUN shields, 88 private transfers, and 74 transactions with an unshield. RAILGUN’s DeFi median uses the 38 of the 6,429 transactions of its relay contract in blocks 25,481,717 to 26,067,921 whose calls reached another contract, mostly swap aggregators. Fuyu’s bars are a funded suite’s second deposit, a relayed transfer, a relayed withdrawal with change that pays a relay fee, and a 200 USDC Aave supply in two transactions (1.20M and 0.20M gas). On cold storage at 128 random tree sizes per point, one new note costs 0.14M gas with four to eight notes and 0.51M with one to two million. Inserting two notes together grows by about as much, so the million-note figures of Section 9.2 add this growth, about 0.38M gas, once per operation.

Benchmark questions. Figure 6 lists the sixteen questions of Section 9.3. The planner refused the whole program, naming the eleven that it cannot prove, and lowered a program of the other five to proof plans.