



Fuyu: Programmable Post-Quantum Privacy for DeFi

Abstract

Public blockchains such as Ethereum now host decentralized finance (DeFi) and a growing share of tokenized assets. While accounts are pseudonymous, anyone can deduce from the public ledger who paid whom, how much, and which positions they hold. Even worse, the record is permanent. Shielded pools hide each payment among many others. Yet most protect their records with elliptic-curve cryptography that future quantum computers can break to reveal past payments. Several support only atomic DeFi trades, and their disclosure mechanisms answer only limited compliance questions.

In this paper, we present Fuyu, a privacy protocol for DeFi that leverages recent advances in zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs) and in post-quantum key encapsulation. First, like other shielded pools, Fuyu holds money in private unspent transaction outputs (UTXOs), called notes, but it goes further and introduces the *deferred note*, whose owner is determined in advance but whose amount depends on external state, such as the reserves of a Uniswap exchange. Fuyu fills in the amount when the trade executes, and anyone can then finalize the deferred note into a normal note. A deferred note can also hold a claim on a pending position, such as a queued withdrawal, so that even non-atomic DeFi operations stay private. Second, Fuyu replaces the elliptic-curve encryption of earlier pools with ML-KEM, a standard quantum-resistant key encapsulation mechanism whose efficient partial decryption also lets recipients scan the ledger faster for the notes addressed to them. Third, we design the Fuyu Policy Language (FPL), a domain-specific language for Yes/No questions about an account's history, such as whether the account holds at least \$1 million. FPL is sound, so any answer that the account's holder proves with a zkSNARK holds on the complete history where the question is defined, while the proof reveals none of the account's records.

We implement and evaluate Fuyu on Ethereum, where it achieves privacy against quantum computers and lower DeFi gas costs than RAILGUN. Fuyu's delivery-tag filter finds a user's outputs on the ledger 6.2× faster than in RAILGUN, the current state of the art for private DeFi on Ethereum. Fuyu keeps its notes in a lean incremental Merkle tree (LeanIMT), which needs fewer hashes to add a note than a standard Merkle tree. Common DeFi operations such as swaps and lending supplies cost around 10% less gas with Fuyu than with RAILGUN under the same conditions. Fuyu is also compliance-friendly, since FPL can express a wide range of common compliance questions, and a user can produce a zero-knowledge proof of funds in a browser within 3.4 minutes.

1 Introduction

Public blockchains such as Ethereum are the first ledgers to run open financial applications, known as decentralized finance (DeFi), at scale. They can do so in part because, unlike banks, they require

no trusted intermediary. Instead of a central operator, every node keeps a copy of the ledger and checks every transaction, so the ledger is public. Financial institutions have now begun to issue stocks, bonds, and fund shares on these ledgers as tokens. The world's stock markets are worth about \$170 trillion [61], and more than \$3 billion of stocks and funds are already tokenized [56].

While users may hold many addresses, or *pseudonyms*, a growing body of research shows that anyone can *de-anonymize* them from public data [29, 35]. An exchange account, a public profile, or a reused address often suffices to tie an address to a person or a company [32], and from then on anyone can search that party's past payments and positions. Worse, the record is permanent. As a result, a public blockchain fails to offer even the confidentiality that institutions have in traditional finance. A fund that moves its treasury onchain exposes every payment, counterparty, and position, and large visible balances make their owners targets [42].

Yet institutions have legitimate reasons to protect their activity. A fund wishes (1) to keep its payments and positions confidential, today and after quantum computers arrive, (2) to keep using DeFi even if a service it relies on stops working, and (3) to show its lender and its auditor only the minimum required information. To move onchain, institutions thus need a privacy protocol that hides their activity from any future attacker, works with the financial contracts they already use, and still lets them prove what their counterparties need to know, such as a minimum balance for a lender or clean sources of funds for an auditor.

Existing approaches and limitations. Privacy protocols that use zero-knowledge proofs, most notably Tornado Cash, RAILGUN, and Privacy Pools v1 [11, 47, 52], follow the design of Zerocash [9]. A zero-knowledge succinct non-interactive argument of knowledge (zkSNARK) lets a prover convince any verifier, including a smart contract, that a statement about secret data, the *witness*, is true without revealing the witness. These *shielded pools* keep money in *notes*, record only hashes of them on the chain, and let a user spend a note by proving that it owns one of the recorded notes, without revealing which. They remain private payment systems rather than a private platform for DeFi, for three reasons.

First, they protect their records with elliptic-curve cryptography. Anyone can save these records now and, once quantum computers exist, link deposits to withdrawals or read payments. This *retrospective quantum attack*, also called *harvest-now-decrypt-later*, can expose past payments even after funds have moved. L2BEAT, the leading third-party platform that rates the security and privacy risks of Ethereum projects, rates Tornado Cash and RAILGUN as at risk [31]. Privacy Pools v1 uses only hashes but supports only deposits and withdrawals.

Second, a DeFi trade's result depends on contract state when it executes, but in Tornado Cash, the proof determines everything a transaction does before it is sent. Many DeFi operations are also *non-atomic*, meaning that they settle only in a later, separate transaction.

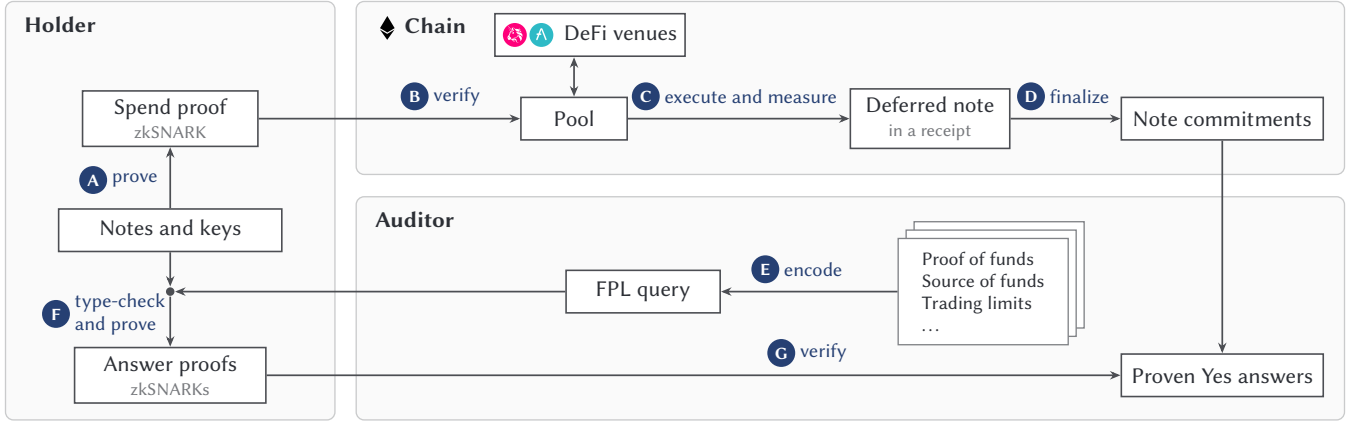


Figure 1: Overview of Fuyu. Boxes are data or components, and the blue labels A–G are steps. A–B prove and verify a spend, C–D execute a trade and finalize its measured result, and E–G let an auditor encode its compliance questions as a query in the Fuyu Policy Language (FPL), send it to the holder, and verify the answers.

In lending, for example, a lender deposits funds into a vault and, to take them back, requests a withdrawal, which some vaults pay out only days later [41]. RAILGUN allows only atomic trades, whose outputs return to its pool within the same transaction [52], so such a withdrawal requires an adapter that represents the pending claim as a shieldable token.

Third, their disclosure mechanisms do not answer arbitrary compliance questions. A Zcash viewing key shows every payment that an account has received [26], and the deposit proofs of Privacy Pools v1 and RAILGUN prove nothing about how much a fund holds [11, 53]. Different questions also need different evidence. A fund can show that it holds at least \$1 million with a few notes that add up to that amount, but to show that none of its money came from a sanctioned address, it must account for every deposit, since any deposit left out could be the sanctioned one.

Challenges. These three limitations pose three challenges for any protocol that serves the fund.

C1: Post-quantum payment privacy. Blockchain data is public and permanent, so an attacker can copy it today and decrypt it years later with a quantum computer. A protocol must therefore make sure that nothing it records on the chain can be read or linked once elliptic curves are broken, including the records of who owns what, the encrypted payments, and any hint that helps a recipient find its payments.

C2: Censorship-resistant transactions. The *holder* of private funds should be able to pay, trade, and withdraw even if the app it uses or the service that submits its transactions disappears. Trades make this difficult. A trade’s output depends on the chain’s state when the trade executes, and a non-atomic DeFi operation delivers it only in a later transaction, so the holder cannot create the resulting note in advance. Some party must create the note once the output is known, and if only one operator can do so, that operator can withhold the holder’s funds.

C3: Privacy-preserving compliance. A lender or an auditor needs answers to specific questions, not the fund’s full history.

A protocol must therefore determine sufficient evidence for each answer and prove the answer in zero knowledge, so that the auditor learns only Yes or No and nothing else about the holder’s notes.

Our approach. In this paper, we propose Fuyu, a programmable privacy protocol for DeFi from zkSNARKs that meets all three challenges. Fuyu follows Bitcoin’s unspent transaction output (UTXO) model. It keeps money in notes, each holding an asset, an amount, and an owner, and each note is spent in full, with any change returned as a new note. All notes belong to one shielded pool, the *Pool*, which records on the ledger only a hash of each note, its *commitment*. To spend a note, its holder sends the Pool a zkSNARK showing that it owns the note, together with the note’s *nullifier*, a one-time hash that prevents double-spending attacks.

As shown in Figure 1, the holder generates at **A** a spend proof, a zkSNARK showing that it owns the notes it spends. At **B**, the Pool verifies the proof and uses the notes’ nullifiers to check that they were not spent before. At **C**, the Pool runs the holder’s trade at a DeFi venue, measures the amount that comes back, and records the trade’s output as a deferred note in a receipt. At **D**, finalizing turns the deferred note into a normal note and adds its commitment to the Pool’s note commitments. At **E**, an auditor encodes its compliance questions, such as proof of funds or source of funds, as a query in the Fuyu Policy Language (FPL) and sends the query to the holder. At **F**, the holder type-checks the query and proves supported answers with zkSNARKs on its own device. At **G**, the auditor verifies these proofs against the note commitments and learns the proven Yes answers, but nothing about the private witness.

To address C1, Fuyu protects note owners and payment openings with hashes, symmetric encryption, and post-quantum key encapsulation, so these records stay private after quantum computers arrive under our cryptographic assumptions. Fuyu uses ML-KEM inside X-Wing [7], which pairs it with the classical elliptic-curve key exchange X25519, so the encryption stays secure as long as either one does. Beyond quantum resistance, ML-KEM also helps a recipient find its notes. Because notes are private, a recipient must

check every new note on the ledger to find those addressed to it. Each delivered private output carries a short *delivery tag*, a hash of its ML-KEM shared secret, so a partial decryption of only a few polynomial multiplications identifies candidate notes for the recipient, without the costly re-encryption check of a full decryption. This check costs far less than the elliptic-curve key exchange that a recipient would otherwise compute for every note.

For C2, we introduce the *deferred note*, a new type of UTXO whose owner is determined in advance but whose amount depends on the state of a DeFi contract. A trade creates a deferred note in two steps. First, the holder’s zkSNARK determines the note’s owner, and in the same transaction the Pool sends the holder’s funds to the contract, measures the amount that the contract returns, and records this amount with the owner in a *receipt*. Second, *finalizing* turns the receipt into a normal note that its owner can spend. Finalizing is permissionless, because the receipt fully constrains the note’s owner and amount, so no centralized operator is needed. A non-atomic DeFi operation, such as a queued withdrawal, becomes two atomic trades. In the first, the venue’s *adapter*, a contract that holds the pending position for all its ticket holders, returns a fungible *ticket*, which the Pool measures into a deferred note. Once the position settles, a second trade exchanges the ticket for the assets, and the holder’s ownership stays private between the two trades.

To meet C3, we design FPL, a domain-specific language for Yes/No questions about an account’s history. We give FPL a formal semantics and a type system that infers sufficient evidence for each answer, either a few of the account’s records or its complete history. We prove that FPL is sound, so any answer that the holder proves with a zkSNARK on its own device holds on the complete history where the question is defined, while the auditor learns only Yes or No.

Evaluation. We implement Fuyu on Ethereum and compare it with Tornado Cash, Privacy Pools v1, RAILGUN, and Privacy Boost, four privacy protocols deployed on Ethereum and OP Mainnet.

- **Privacy.** Of the compared privacy protocols that support private payments and DeFi, only Fuyu keeps payments unlinkable against a future quantum computer, because it protects note owners and payment openings with hashes, symmetric encryption, and post-quantum key encapsulation.
- **Speed.** Fuyu’s delivery-tag filter finds a user’s outputs on a ledger with one million notes in 2.4 min, 6.2× faster than in RAILGUN on the same machine, because a delivery tag needs only a partial decryption.
- **Cost.** Common DeFi operations such as swaps and lending supplies cost around 10% less gas with Fuyu than with RAILGUN under the same conditions. Fuyu’s lean incremental Merkle tree (LeanIMT) needs fewer hashes to add a note.
- **Compliance.** FPL can express a wide range of common compliance questions. A zero-knowledge proof of funds over up to 64 notes takes 15 s natively and 3.4 minutes in a browser, and the auditor verifies it in 0.34 s, whereas the compared systems offer no such zero-knowledge proof of funds.

Our contributions. Our contributions are as follows.

- (1) **Post-quantum payment privacy.** A note format that protects owners and payment openings with hashes, symmetric encryption,

and post-quantum key encapsulation, with delivery tags that let a recipient identify candidate notes after the first, cheap step of ML-KEM decryption.

- (2) **Deferred notes.** A new type of UTXO whose owner is determined in advance but whose amount depends on the state of a DeFi contract. Permissionless finalization turns it into a normal note, in the same or a later transaction, which supports non-atomic DeFi operations without a centralized finalization operator.
- (3) **A domain-specific language for privacy-preserving compliance.** The Fuyu Policy Language (FPL), a domain-specific language for Yes/No questions about an account’s history, with a formal semantics, a type system that infers sufficient evidence for each answer, and a proof of soundness, so that any answer that a holder proves with a zkSNARK holds on the account’s complete history where the question is defined.

We believe that Fuyu opens up private DeFi to institutions and is a key step toward moving institutional finance onchain.

2 Overview

2.1 Background

zkSNARKs. A zkSNARK is a non-interactive zero-knowledge proof of knowledge that is *succinct*, meaning that its proofs are short and quick to verify. More precisely, a rule $\mathcal{R}(X; w)$ relates a public *statement* X to a private *witness* w . To prove the rule, one writes it as an *arithmetic circuit*, a static list of additions and multiplications over a finite field whose inputs are X and w , and which accepts exactly when $\mathcal{R}(X; w)$ holds. A prover then produces a proof π that it knows some w with $\mathcal{R}(X; w)$, and anyone can check π . The proof is *zero knowledge*, so it reveals nothing beyond this fact, and it is a *proof of knowledge*, so a prover can make it only if it actually knows such a w . Knowing matters here, because a witness exists for every note, but only its owner knows one.

Shielded pools. Like Bitcoin [36], shielded pools follow the UTXO model, in which money is held in unspent transaction outputs that are each spent in full, and any change is returned as a new output. In a shielded pool, these outputs are called *notes*. A note records an asset, an amount, and an owner, together with secret randomness, and these contents form the note’s *opening*. The pool never stores the opening. It stores only the note’s *commitment*, a hash of the opening. Because the opening includes secret randomness, the commitment reveals nothing about the note, and because finding two openings with the same commitment is infeasible, the commitment can be opened only to the note it was made from. A deposit hands the pool some tokens together with a commitment to a new note for that amount. To spend a note, its *holder* posts the note’s *nullifier*, a hash of the commitment and a secret key of the holder, together with a zkSNARK proof of the statement “I know the opening of a commitment in the pool’s list whose nullifier is this one.” The pool rejects a nullifier that it has seen before, so no note can be spent twice. Since the proof reveals nothing more and the nullifier looks random to anyone without the key, no one can tell which note was spent. All the notes that a spend could have come from form its *anonymity set*, and unusual amounts, timing, and reused addresses can shrink that set [29].

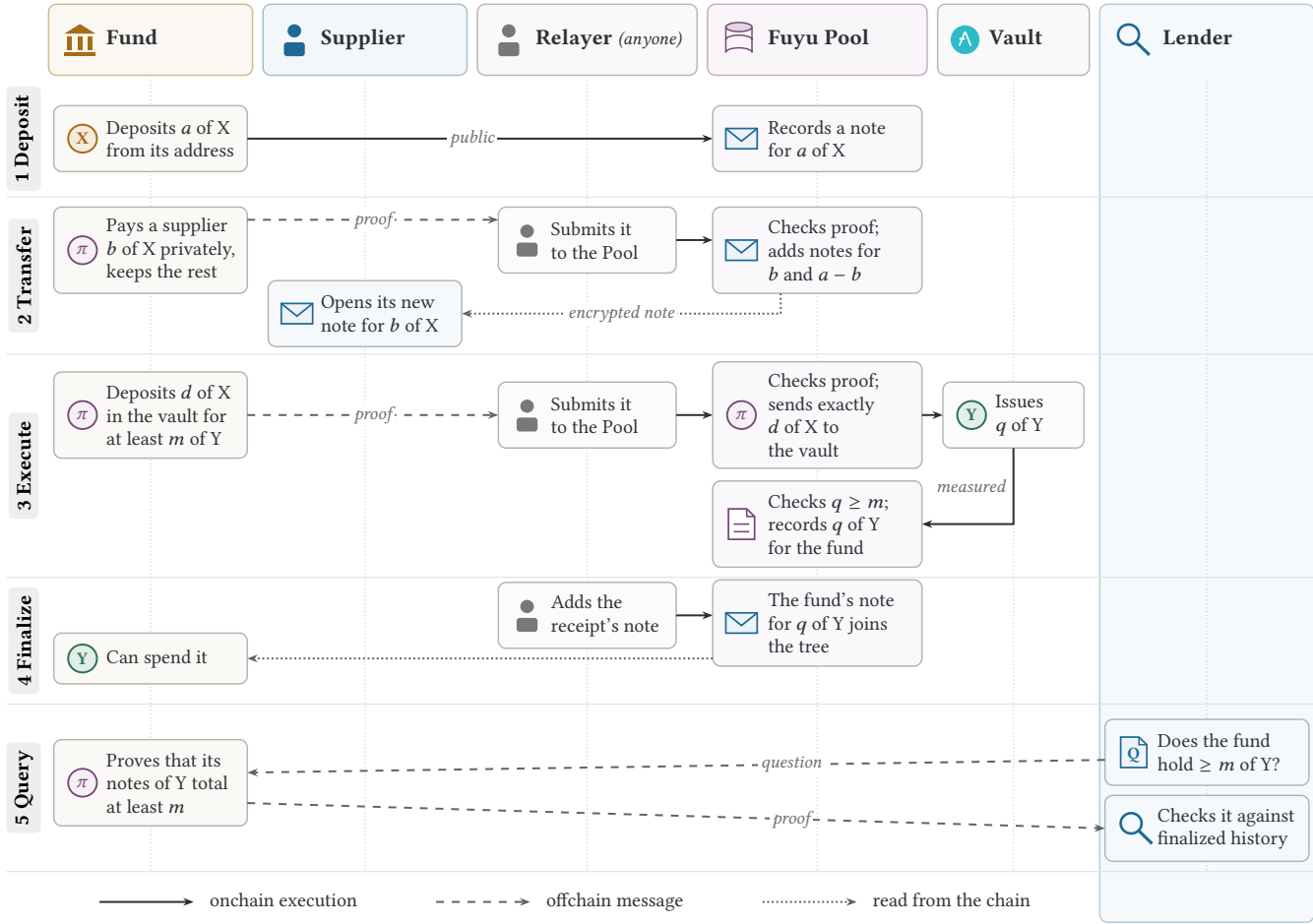


Figure 2: One deposit of the fund’s treasury through Fuyu. (1) The fund deposits a of a stablecoin X. (2) The fund pays a supplier b of X privately and keeps the rest as change, through a relay that anyone can run. (3) It deposits d of X in a vault for at least m of the share token Y, and the Pool records the measured q of Y in a receipt. (4) The relay adds this note to the note tree. (5) The lender asks whether the fund holds at least m of Y, and the fund answers with a proof.

The note tree. The pool keeps the commitments as the leaves of a *Merkle tree*, in which each inner node hashes its two children, so that the *root* rt summarizes every leaf. A spend then proves that it knows the opening of a leaf under rt , using only the leaf’s path to the root, whose length grows with the logarithm of the number of notes.

Private payments. A spend consumes notes and creates new ones, one for the recipient and one for any change, and its proof shows that the values add up. A spend may also pay part of its value to a public address, which is how funds leave the pool. The sender encrypts each new note’s opening for its recipient, so that the recipient can later spend it. In Zcash and RAILGUN, this encryption uses a key that the sender agrees with the recipient through an elliptic-curve multiplication, and the recipient finds its payments by trying to decrypt every new note [26, 54]. These attempts are the main cost of running a wallet, because they grow with the number of notes on the chain.

Post-quantum key encapsulation. A *key encapsulation mechanism* (KEM) lets a sender use the recipient’s public key to create a fresh shared secret and a ciphertext, from which the recipient recovers the same secret with its private key. ML-KEM [38] is a KEM designed to resist cryptanalytic attacks by future powerful quantum computers, and it was standardized in 2024. Its decapsulation first decrypts the ciphertext with a few polynomial multiplications and then re-encrypts the result to check it, which costs most of the time. X-Wing [7] is a KEM that runs ML-KEM-768 and the elliptic-curve key exchange X25519 side by side and hashes both secrets into one, so it stays secure as long as either one does, X25519 against classical attackers and ML-KEM-768 against quantum ones.

DeFi venues. A *DeFi venue* is a contract that takes one token and gives back another, for example an exchange such as Uniswap [3] or a vault that takes deposits [57]. Because other trades can move the price before it runs, the amount it returns is known only afterwards, so the caller states the least it will accept. A lending vault, for

instance, takes deposits and issues shares that earn interest. Some venues, such as a withdrawal queue or a prediction market, settle only after a later event.

2.2 Motivating Example

Figure 2 follows one deposit of the fund’s treasury through Fuyu. Each column of the figure is a party, from the fund and its supplier to the relayer, the Pool, a lending vault, and the lender, and each row is one step. The fund pays suppliers from its stablecoin treasury and puts idle stablecoins in a lending vault, and each quarter its lender asks about its holdings. In the figure, X is a stablecoin such as USDC, Y is the vault’s share token, π marks a proof, and Q marks the lender’s question. Because a note is spent whole, like a Bitcoin output, every payment creates one note for the recipient and another for any change.

- (1) *Step 1: Deposit.* The fund deposits a of X from its public address, which everyone can see. The Pool records a note for the fund. The deposit needs no proof, because the Pool sees what it received, and the note names its owner only through a hash.
- (2) *Step 2: Transfer.* The fund then pays a supplier b of X . Its wallet spends the whole note and creates two new notes, with b for the supplier and $a - b$ as change. The proof shows that the fund owns a note in the Pool, and the spend publishes the note’s nullifier. Each new note travels encrypted to its recipient, so the supplier need not be online. The wallet then hands the proof to a *relayer*, which submits it from its own address. Anyone can be a relayer, because the proof covers the whole request, so a relayer can delay or withhold it but not change it.
- (3) *Step 3: Execute.* Next, the fund deposits d of X from its change in the vault. Its proof states the input d , the minimum return m of Y , the *route*, which names the contract that performs the deposit, and the owner of the result, given as a hash that names no address. Once the relayer submits the request, the Pool calls the route, measures the q of Y that the vault issued, and checks that $q \geq m$. It then records q and the owner in a *receipt*, together with the fund’s remaining change.
- (4) *Step 4: Finalize.* The relayer, now acting as *finalizer*, turns the receipt into notes in the Pool, and only then can they be spent. Finalizing is a separate step so that anyone could finish it if the relayer stopped here. Normally, both steps run in one transaction.
- (5) *Step 5: Query.* At quarter end, the lender asks in FPL whether the fund holds at least m of Y . The fund proves on its own device that its unspent notes of Y total at least m at the query’s snapshot, without revealing which notes it counted or how many, up to a preset cap. Answering is the fund’s choice, and no Fuyu service is involved. The lender checks the proof against *finalized* history, the blocks the chain has accepted as final.

What observers learn. Observers see the fund’s deposit of a and the relayer as the sender in steps 2 to 4. They cannot, however, see who made steps 2 and 3, whom the fund paid in step 2, or how much. Step 3 calls a public venue, so its assets and the amounts d , m , and q are visible, as for any trade [4]. The rest of the change stays private, so even someone who knows a and d cannot tell how it was split. Amounts, timing, and the choice of relayer and fee are the

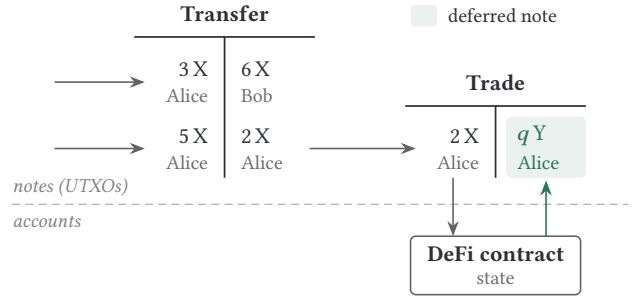


Figure 3: A deferred note. Each transaction lists its inputs on the left and its outputs on the right. X , Y are assets, and q is the output amount determined by the DeFi contract.

main clues that could link a spend to a deposit, so common amounts and waiting still help [32]. A *withdrawal* to a public address shows its amount.

3 Problem Formalization

3.1 Deferred Notes

Blockchains keep money in one of two ways. In Bitcoin’s UTXO model, the ledger holds unspent transaction outputs (UTXOs), each with an owner, an asset, and an amount, and a transaction consumes some UTXOs in full and creates new ones. In Ethereum’s account model, contracts keep state, such as an exchange’s reserves, and what a transaction does depends on that state when it executes. Shielded pools use the UTXO model, because a note can be spent without revealing which one it is, whereas DeFi contracts keep their state in accounts (Figure 3).

In the shielded pools of Section 2.1, the prover decides every new note, its owner and its amount, before the transaction is sent, and the contract only checks the proof. In Figure 3, the transfer is such a transaction. It merges two of Alice’s notes and splits their value into a payment to Bob and Alice’s change. Such a transaction cannot use an account’s state, because what a trade returns depends on that state when it executes (Section 2.1).

A *deferred note* connects the two. It is a new type of UTXO whose owner is determined in advance but whose amount is determined by an account’s state when the transaction runs. To create one, a transaction spends notes into the account, and its proof commits to the operation on the account, called a *route*, with its input, and to the deferred note’s owner through the *owner commitment*, a hash that hides who the owner is. When the transaction executes, the ledger performs the route, measures the amount q that the account returns, and records a *receipt* that binds q to the committed owner. *Finalizing* the receipt then adds the deferred note to the ledger as an ordinary note that its owner can spend. In Figure 3, the trade sends Alice’s 2 X to a DeFi contract, such as an exchange or a lending vault, whose state at that moment sets the amount q of Y that she receives. A non-atomic DeFi operation, such as a queued withdrawal or a bet on a prediction market, settles only later, so it becomes two atomic trades. The first returns a fungible *ticket* for the pending position, and once the position settles, the second exchanges the ticket for the assets.

A deferred note thus splits a new note between two parties. The prover decides, privately and in advance, who owns it, and an account's state decides how much it holds. The DeFi contract does not need to know about notes, because the ledger measures what the contract returns instead of asking it. Any admitted route that pays out such a token can therefore produce a deferred note, and none can misreport the amount. Finalizing a receipt is permissionless and happens exactly once, so no party can withhold a finished trade from its owner. Ordinary payments are the special case in which the prover decides every note.

3.2 Threat Model

Parties and authority. Several parties take part in the example, but only the holder can authorize a spend. No relayer, finalizer, or auditor can move or reveal its notes. Private use of deposit notes also depends on source screening; the holder can otherwise exit by revealing the deposit's source.

- **Holder.** The holder knows its spend secret and the note's opening, so it can spend its notes, but it can neither double-spend a note nor counterfeit funds.
- **Relayer, adapter, and finalizer.** A relayer may submit, delay, or refuse a request, but cannot change it. An adapter performs one operation at one or more DeFi venues. The finalizer, who may be anyone, adds a receipt's notes to the tree but cannot change them.
- **Auditor.** An auditor, such as the fund's lender, may send the holder questions and check its answers. It cannot force an answer or read the holder's private history.

Adversary. The adversary may own notes, run relayers, finalize trades, receive audit queries, and see every public or pending transaction. It may race or replay a request, submit a false proof, or trade ahead of a trade [15]. It does not know the victim's spend secret, although it may know the opening of a note it paid to the victim. A trade's minimum return limits the victim's loss from a price movement. Later, the adversary may gain a quantum computer and revisit all recorded chain data.

Retrospective quantum attacks. A retrospective attack targets the privacy of past, honestly executed transactions. An adversary may archive today's note ciphertexts and later decrypt them with a quantum computer, the *harvest-now-decrypt-later* threat. Recovering an opening through recorded ECDH ciphertexts can expose its note's ownership even if the hash commitment itself remains hiding. Moving funds or replacing a verifier cannot erase these archived records. Under the assumptions below, Fuyu protects encrypted openings with X-Wing and ownership links with secret per-note hash tweaks. For a deferred settlement, the holder fixes its tweak from its viewing key and recovery salt before execution; the Pool later binds the measured amount q to the fixed owner commitment, and finalization needs no disclosure of the tweak. These historical privacy guarantees allow the public-operation and timing leakage described here. Quantum forgery against a live verifier is a separate threat, addressed below in Section 6.

Assumptions. Fuyu's guarantees rest on four assumptions. Custody and exit also require assets with truthful balances, exact transfers, and no autonomous balance reduction or transfer blocking.

- **Sound, zero-knowledge proofs.** The zkSNARK is knowledge sound: an accepted proof shows that its prover knows a valid witness, even if the adversary has seen other proofs. It is also zero-knowledge, so a proof reveals nothing else. The adversary knows no trusted-setup secret.
- **Secure hash functions.** Poseidon, SHA-256, Keccak-256, and X-Wing's SHA3-256 combiner are collision resistant, so a commitment cannot be opened to other note fields. On a secret high-entropy input, they are also pseudorandom, even to a quantum computer, so commitments hide the note fields, and nullifiers and tags look random.
- **Post-quantum key encapsulation.** ML-KEM-768 is secure against chosen-ciphertext attacks (IND-CCA) and anonymous, even against a quantum computer. Only the recipient and the sender can recover the secret behind a ciphertext, which does not reveal the recipient's key. For the delivery envelope, we additionally assume that publishing the derived tag preserves recipient anonymity and the combined secret's confidentiality.
- **Secure symmetric encryption and key derivation.** Note-opening and sender-recovery encryption is semantically secure, and domain-separated key derivation produces pseudorandom keys and tweaks from secret seeds or shared secrets, also against a quantum adversary.

The privacy guarantee covers note records and delivery ciphertexts, assuming the holder's secrets remain private. It does not cover signature-derived wallet seeds or public recovery backups unlocked by an ECDSA wallet signature, which a quantum attacker could reproduce after recovering that wallet's key.

4 The Fuyu Protocol

Fuyu implements private notes and deferred notes with one contract, the *Pool*, which holds the assets and stores the notes. Figure 4 shows how Fuyu computes its six data structures, the note tree (a), a payment's delivery tag (b), a note's owner tag (c), its commitment (d), a deferred note's commitment (e), and a note's nullifier (f). We write H_x for a Poseidon hash construction [21] with *domain tag* x , a constant number that keeps different uses of the hash apart. The numbers are fixed labels for hashing purposes, not leaf indices. Note commitments, nullifiers, and request hashes also take the deployment's *domain* D , which binds its chain and Pool address, so that no note or proof can be replayed on another Pool. We omit domain fields and fixed zero padding below.

4.1 Keys and Notes

Keys and addresses. A holder picks a random *spend secret* k and derives two values from it (Figure 4c and f), its *owner root* $R = H_{218}(k)$ and its *nullifier key* $nk = H_{219}(k)$. Because the two hashes use different tags, one value reveals nothing about the other. The holder publishes R but keeps nk secret, and neither reveals k . To receive payments, the holder also generates an X-Wing key pair (Section 2.1). Its *receiving address* is the pair (R, pk) of its owner root and its X-Wing public key. The address holds no secret, so the holder may give it to payers or publish it.

Notes. We describe notes with no *spend controller*, an optional wallet or contract whose approval is required in addition to the

tinted blue secret

tinted blue secret

tinted orange set b

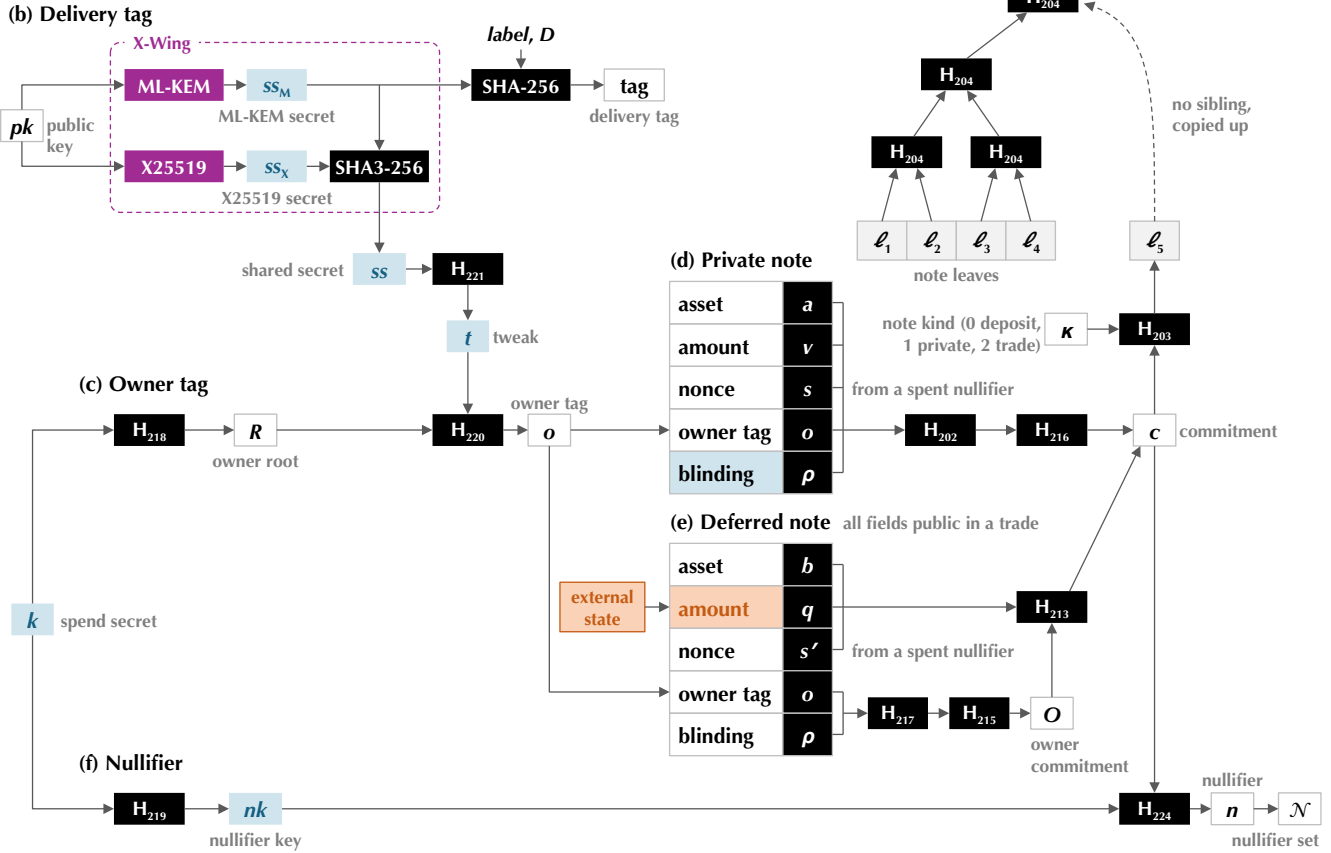


Figure 4: Fuyu’s data structures. Here *label* is `fuyu.tag.view`, and *D* identifies the chain and Pool. In a trade, the trader’s wallet derives *t* from its viewing key instead of *ss*. Domain and fixed inputs to Poseidon are omitted.

holder's spend proof. Other notes publish the controller's identity when spent. A private note holds an asset a , an amount v , an owner tag o , a nonce s , and secret randomness ρ , and its commitment c is (Figure 4d)

$$c = H_{216}(H_{202}(a, v, o, s, \rho), 0). \quad (1)$$

The final zero here and in the owner commitment below means no extra approval is required; the holder must still prove ownership. The *owner tag* $o = H_{220}(R, t)$ hashes the owner’s root with a *tweak* t , a fresh secret for every note (Figure 4c). Without the tweak, all of a holder’s notes would carry the same owner value, and publicly exposed owner tags could be linked. With it, each note’s owner tag looks unrelated to the others, yet only the holder of k can prove that a tag belongs to its root. In a private note, the randomness ρ hides the note’s fields, and the *nonce* s gives the note a unique identity. Without it, a payer could send two notes with identical contents, which would share one nullifier, so the recipient could spend only one of them. Fuyu therefore derives s from a nullifier that the creating transaction spends, which never repeats, and from the note’s position in that transaction.

Nullifiers. A note’s nullifier is $n = H_{224}(nk, c)$ (Figure 4f), using its commitment c and the holder’s secret nullifier key nk . Only the note’s owner can compute it, because nk is secret, and each note has exactly one, because k determines nk . To spend a note, its holder publishes n , and the Pool rejects a nullifier that it has seen before. The spend proof links n to an owned commitment without revealing which commitment was spent.

Deferred notes. A deferred note’s amount is unknown when the holder makes its proof, so the proof cannot contain the note’s commitment. It contains an *owner commitment* O instead (Figure 4e),

$$O = H_{215}(H_{217}(o, \rho), 0),$$

which binds the new note's owner tag o and fresh randomness ρ . For a trade, the Pool receives o and ρ publicly; the secret tweak t hides o 's connection to the holder's root R . The wallet derives t and ρ separately from its secret viewing key and a fresh public recovery salt. When the trade executes, the Pool measures the amount q of the output asset b that the venue returned and computes the note's

commitment c

$$c = H_{213}(b, q, s', O). \quad (2)$$

Like a private output, the nonce $s' = H_{214}(n_0)$ comes from a nullifier that the trade spends, here n_0 , the nullifier of the trade's first input note. Everyone sees O and q , but only the holder, who knows k and t , can spend the note. A deposit note uses the same owner commitment. The depositor supplies only O , keeping o and ρ private, and the Pool computes $H_{201}(a, v, \text{src}, h, s, O)$ from the amount v of asset a that it received, the depositing source src , the deposit block h , and a nonce s from a counter that increases with every deposit.

Ledger state. The Pool keeps four pieces of state. The *note tree* T is a Merkle tree with up to 2^{32} leaves (Figure 4a). Each leaf is $\ell = H_{203}(\kappa, c)$, where c is the note's commitment and its *kind* κ is 0 for a deposit note, 1 for a private note, or 2 for a trade note, so that a spend proof knows which of the three commitment formulas to open. An inner node hashes its two children with H_{204} . The tree is a *lean incremental Merkle tree* (LeanIMT) [65]. A node without a right sibling moves up unchanged, as ℓ_5 does in Figure 4a, so adding a note takes one hash per set bit of its position rather than one per level, about $\frac{1}{2} \log_2 L$ hashes on average over the first L insertions instead of 32, where L counts appended leaves. Every completed nonzero root that the tree publishes stays *accepted*, so membership against an older root still verifies; the other spend checks still apply. The *nullifier set* $\mathcal{N}$ holds the spent nullifiers, and the *receipts* J hold executed trades whose notes are not yet in the tree. The *reserve* $C[a]$ records how much of each asset a backs the notes and receipts.

4.2 Private Delivery

A commitment does not tell the recipient how to open its note, so the sender also sends the note's opening, encrypted for the recipient. The sender takes the recipient's X-Wing public key pk from its receiving address and proceeds in four steps (Figure 4b).

- (1) *Encapsulate.* The sender runs X-Wing's encapsulation on pk , which returns a fresh shared secret ss and a ciphertext from which only the recipient, with its private key, recovers ss .
- (2) *Derive.* From ss , the sender derives the note's tweak t , and hence its owner tag $o = H_{220}(R, t)$ using the recipient's owner root R , as well as a symmetric key. The recipient recomputes both from ss , whereas observers without the shared secret see an owner tag unrelated to the recipient's address and cannot link the payment to that address from the tag.
- (3) *Encrypt.* The sender encrypts the note's opening under the symmetric key. The spend proof shows that this ciphertext decrypts to the opening of the new commitment under the supplied shared secret. It does not prove the KEM ciphertext's correctness; the recipient checks the commitment after decapsulation before crediting the note.
- (4) *Tag.* The sender attaches a 32-byte *delivery tag*, a SHA-256 hash of the ML-KEM-768 component's shared secret ss_M , the fixed label `fuyu.tag.view`, and the deployment domain D . The subscript M denotes ML-KEM; Figure 4b shows its component secret ss_M beside the X-Wing secret ss . The tag uses SHA-256 rather than Poseidon, because Poseidon's outputs are field elements whose biased top bits would set them apart from random

strings. The recipient, who can recover ss_M , can recompute the tag. Delivery-tag privacy requires ML-KEM-768 security, even if X25519 remains secure.

A note's *delivery envelope*, its tag followed by the X-Wing ciphertext, is published with the transaction. Wallet-generated payment and change envelopes use the same size and format; the envelope itself does not identify their role. The proof covers every envelope in order, so a relayer cannot remove or alter one, and the Pool never reads a tag but only hashes each envelope into the request.

Finding payments. A recipient must examine every new note to find its own. Decapsulating an ML-KEM ciphertext takes two steps. The first, a decryption, derives a candidate shared secret with a few polynomial multiplications. The second re-encrypts the result and compares it with the ciphertext, which rejects forged ciphertexts but costs most of the time. To check a note, the delivery-tag filter runs only the first step and compares the note's delivery tag with the hash of the candidate secret. Only when the tag matches does the wallet run the second step, finish the X-Wing decapsulation, and decrypt the note's opening. Filter matches remain local until full decapsulation and commitment validation succeed. This optimization assumes that adversaries cannot observe partial-filter outcomes or timing. Change carries a tag for the holder's own key, so a wallet finds its change in the same way.

4.3 The Spend Circuit

Every spend, whether a transfer, a withdrawal, or a trade, proves one rule $\mathcal{R}_{\text{spend}}(X; w)$ with one arithmetic circuit (Section 2.1). A spend consumes one or two notes of the same asset and creates at most two private notes. The public statement X holds an accepted note-tree root rt , the nullifiers, the new notes' commitments, a hash of the request, and any public payout and fee. For a trade, X binds the route through the request hash and holds the exact input d that the route receives, the minimum return $q_{\min}$, and the owner commitment O of the deferred note. The private witness w holds the spend secret k , the openings of the input notes with their tweaks and tree paths, and the openings and shared secrets of the new notes. The circuit makes three groups of checks.

- (1) *Ownership.* Each input's opening gives a commitment whose leaf lies in the tree under the stated root. Here k is the holder's spend secret, and t , ρ , and c are that input note's tweak, randomness, and commitment. The input's owner tag o equals $H_{220}(H_{218}(k), t)$, inside $O = H_{215}(H_{217}(o, \rho), 0)$ for a deposit or trade note, and its nullifier equals $H_{224}(H_{219}(k), c)$.
- (2) *Value.* Each new note's commitment matches its opening, and its ciphertext decrypts to that opening under the key derived from its shared secret. In the input asset, let v_0, v_1 be the input amounts, y_0, y_1 the new private amounts, v_{pay} the public payout, v_{fee} the relayer's fee, and d the amount sent to a venue, and let α be 1 for a trade and 0 otherwise. Missing inputs or outputs have amount zero. The circuit range-checks every amount and enforces

$$v_0 + v_1 = y_0 + (1 - \alpha)y_1 + v_{\text{pay}} + v_{\text{fee}} + d. \quad (3)$$

For a trade, $\alpha = 1$ removes y_1 , because the proof accounts only for the venue input and the private change y_0 , and the Pool measures the venue's output later.

- (3) *Binding*. The request hash in X covers the payout’s recipient, the relayer, the fee, a deadline, and the delivery envelopes, and for a trade also its route, d , $q_{\min}$, and O . A relayer that submits the proof can therefore delay or withhold it, but cannot change any of these.

4.4 Operations

The Pool has five operations. DEPOSIT brings assets in, TRANSFER pays privately, WITHDRAW pays out to a public address, and a trade runs in two stages, EXECUTE and FINALIZE. TRANSFER, WITHDRAW, and EXECUTE each take a spend proof. The Pool first checks that the proof verifies, that its root is accepted, that its nullifiers are not in $\mathcal{N}$, and that its deadline has not passed, and it then adds the nullifiers to $\mathcal{N}$. Deposit notes must additionally pass source screening and its delay for private spending; a full withdrawal revealing the source bypasses screening. Each operation applies in full or not at all, so no intermediate state can be spent.

Deposit. The depositor sends v of asset a with an owner commitment O . The Pool measures what it received and adds a deposit note for v of a , owned through O , to the tree. A deposit needs no proof, because the Pool sees the amount itself.

Transfer and withdraw. A transfer adds its new private notes, the payment and any change, to the tree in the same step that spends its inputs. A withdrawal sends v_{pay} to a public address and adds any private change.

Execute and finalize. A route names an *adapter*, a contract that performs one operation at one or more venues, and the pair of assets it trades. A trade runs in two stages (Algorithm 1), with spend proof π , public statement X , and trade request A .

- (1) EXECUTE spends the inputs, sends exactly d of the input asset a to the adapter, and calls it. The Pool then measures the output q of asset b from its own balance and requires $q \geq q_{\min}$, or the whole transaction reverts. It computes the trade note’s commitment from q and O by Equation (2) and stores this commitment c_{trade} with the private-change commitment c_{change} , zero if absent, in a receipt $J[n_0]$, keyed by the first input’s nullifier n_0 .
- (2) FINALIZE deletes a receipt and adds its notes to the tree. Anyone may call it, and it takes no input but the receipt’s key $j = n_0$, so the finalizer cannot change the amount or the owner.

Every spend proves that its inputs are in the tree, and a receipt’s notes are not yet there, so they become spendable only after FINALIZE. The two stages usually run in one transaction. They are separate so that anyone can complete a trade whose submitter disappears.

Algorithm 1 uses the note tree T , spent-nullifier set $\mathcal{N}$, reserves C , and pending receipts J . SpendOK checks the proof, the accepted root, that the nullifiers are distinct and unused, applicable deposit-source admission, the request hash, and the deadline. RouteOK checks that the admitted adapter’s runtime code and Pool binding match the route and that it trades a for b . $|T|$ counts appended leaves, and $\text{slots}(J)$ counts the notes reserved by pending receipts; all other appends preserve these slots. The indicator $\mathbf{1}[P]$ is 1 if P holds and 0 otherwise; $\emptyset$ means no receipt. The input nullifiers

Algorithm 1 The two stages of a trade.

```

1: procedure EXECUTE( $\pi, X, A$ )
2:   Decode  $(a, b, d, q_{\min}, O, v_{\text{fee}}, n_0, n_1, c_{\text{change}})$  from  $X, A$ 
3:   require SpendOK( $\pi, X, A, T, \mathcal{N}$ )
4:   require  $a \neq b$ , RouteOK( $A, a, b$ )
5:   require  $C[a] \geq d + v_{\text{fee}}$ ,  $J[n_0] = \emptyset$ 
6:   require  $|T| + \text{slots}(J) + 1 + \mathbf{1}[c_{\text{change}} \neq 0] \leq 2^{32}$ 
7:    $\mathcal{N} \leftarrow \mathcal{N} \cup \{n_0, n_1\}$ ;  $C[a] \leftarrow C[a] - d - v_{\text{fee}}$   $\triangleright$  before the call
8:    $(\hat{b}, \hat{q}) \leftarrow \text{CallAdapter}(A, d, q_{\min})$ 
9:   Measure the input debit  $\Delta_a$  and the output credit  $q$ 
10:  require  $\Delta_a = d$ ,  $\hat{b} = b$ ,  $\hat{q} = q$ 
11:  require  $0 < q < 2^{128}$ ,  $q \geq q_{\min}$ 
12:   $C[b] \leftarrow C[b] + q$ 
13:   $c_{\text{trade}} \leftarrow H_{213}(b, q, H_{214}(n_0), O)$ 
14:   $J[n_0] \leftarrow (c_{\text{change}}, c_{\text{trade}})$ 
15:  Pay the fee  $v_{\text{fee}}$  to the relayer
16: end procedure
17: procedure FINALIZE( $j$ )
18:   require  $J[j] = (c_{\text{change}}, c_{\text{trade}})$  exists
19:   Delete  $J[j]$ 
20:   Append  $H_{203}(1, c_{\text{change}})$ , if any, then  $H_{203}(2, c_{\text{trade}})$  to  $T$ 
21:   Accept the new root; leave  $\mathcal{N}$  and  $C$  unchanged
22: end procedure

```

are n_0, n_1 ; for a one-note spend, n_1 is a fresh padding nullifier. Hats mark the values that the adapter reports, and Δ_a and q are the debit and the credit that the Pool measures itself. EXECUTE marks both nullifiers as spent and debits $C[a]$ before it calls the adapter, and a failed check reverts the whole transaction.

Non-atomic DeFi operations. A non-atomic DeFi operation, such as a redemption from a queue that follows the ERC-7540 vault standard [41], becomes two atomic trades. In the first, the venue’s adapter holds the pending position for all its ticket holders and returns a fungible *ticket*, which EXECUTE measures like any other output. Once the position settles, a second trade exchanges the ticket for the settled assets, and the holder’s ownership stays private between the two trades. Our ERC-7540 test adapter uses aggregate request ID zero and requires the venue to enforce controller consent; settlement depends on the venue. Prediction markets fit in the same way, because all shares of one outcome pay the same, so an adapter can hold them and issue tickets for them.

4.5 Security Analysis

Under the assumptions of Section 3.2, we prove in Appendix B that Fuyu has the following properties.

- (1) **Ownership.** Only the holder of a note’s spend secret can spend the note.
- (2) **No double spending.** Each note has exactly one nullifier, and the Pool accepts it only once, so no note can be spent twice.
- (3) **Solvency.** For every asset, the Pool’s balance covers all unspent notes and pending receipts of that asset.
- (4) **Unlinkability.** No observer without the opening or recipient keys, even one with a quantum computer, can tell which note a spend consumes or who owns a new note, beyond what the public operation fields and timing reveal.

```

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
  nonce 0x2a91 {
    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 5: The treasury’s disclosure policy and the lender’s quarterly review in FPL. The query’s 32-byte nonce is shortened, and its snapshot height belongs to the illustrative fixture.

5 The Fuyu Policy Language

A holder answers an auditor’s questions with proofs instead of records. The questions are written in the Fuyu Policy Language (FPL), a typed language for Yes/No questions about one account’s history. FPL can express a lender’s questions, including those whose desired answer is No, and its type system infers sufficient information to certify each answer. For example, enough unspent notes prove a minimum balance, but chosen deposits cannot show that no prohibited deposit exists.

How an audit runs. An auditor sends the holder a *query*, a named list of questions about one account, the query’s *subject*, at a finalized block, the query’s *snapshot*. The holder decides whether to answer, proves its answers on its own device, and sends the proofs to the auditor, who verifies them against the chain. No one else receives the holder’s records or a viewing key.

Programs and names. An FPL program is either a disclosure policy or a query. The holder’s *disclosure policy* states in advance which auditors it accepts, which kinds of questions each may ask, and which public facts an answer may reveal. FPL refers to the outside world only through names, such as a currency, a token, or a list, which a *registry* that both parties share maps to an address, an asset, a price feed, an address list, or a credential issuer.

Example. Figure 5 shows a disclosure policy and a quarterly query for the treasury. The policy has two *grants*, each written with **permit**. A grant lets one auditor ask certain kinds of questions, called *capabilities*. Besides the Yes/No result, the answers may reveal only the public facts listed in a reviewed *effect manifest*. **holdings** covers balances and values, and **provenance** covers deposits and their sources. **activity** covers trades, transfers, withdrawals, and received notes, and **identity** covers credentials. The first grant allows holdings questions within the CurrentView manifest, and the second allows all four capabilities within AuditReview.

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

$::=$ defines forms, $\mid$ separates alternatives, and $\oplus$ collects grants. Arguments name auditor u , capabilities K , public effects E , optional subject s and policy digest h , snapshot η , expiry δ , nonce v , and questions $\vec{\theta}$. hold/prov/act/id mean holdings/provenance/activity/identity. x names formula φ ; τ is a numeric term, c a bound, R a record relation, F, G filters, f a field, v a value, $\vec{L}$ list unions, I a time interval, and P a period. a, N, t denote asset, numeraire, time; is, cl denote issuer, credential claim.

Figure 6: Abstract syntax of FPL. $\bowtie$ ranges over $<, \leq, =, \neq, \geq, >$, and $[N]$ marks an optional numeraire. An omitted filter means the always-true $\top$.

The query is addressed to auditor and asks about treasury, names that the registry maps to the auditor’s and the subject’s addresses. **finalized**(24000100) sets the snapshot block, **expires** the query’s deadline in Unix seconds, and **nonce** a fresh value from the auditor. The registry also defines the remaining names. USD is a numeraire with a price feed for each asset, and stataUSDC is the vault’s share token. OFAC@17 is a list of sanctioned addresses, where @17 names its 17th published version, whose contents the registry identifies by a digest, so both parties check the same addresses. KycIssuer is a credential issuer that attests that an account belongs to a verified entity. The query asks five questions that a lender might ask before it renews a loan, and a sixth that combines four of them into one decision.

- (1) **over_1m** asks whether holdings, valued in USD, reach \$1M.
- (2) **sanctioned_funds** tests for a **deposit** from OFAC@17.
- (3) **held_sep_30** asks whether the stataUSDC balance reached 25,000 units at the end of September 30, 2026.
- (4) **within_limits** asks whether the trade inputs of each UTC day from July 1 through September 30 were at most \$50,000 in USD.
- (5) **verified_entity** asks whether KycIssuer has given the treasury a valid, unrevoked **verified_entity** credential.
- (6) **cleared** combines four earlier answers, where **and** requires all four conditions and **not** requires a No to **sanctioned_funds**.

We checked both programs with our implementation of FPL’s type checker. Under the registry, the query is well typed, the policy’s second grant allows each of its questions, and each question receives the evidence profile that Section 5.2 derives for it. The checker

rejects a query that names a list missing from the registry, such as OFAC@18. Under a policy with only the first grant, it even rejects over_1m, because the answer would reveal its price feed, which CurrentView does not list.

5.1 Syntax and Semantics

Figure 6 gives FPL’s abstract syntax, in which $::=$ defines a form, $|$ separates the alternatives, and an arrow over a symbol marks a list. A disclosure policy p consists of grants g , and each grant gives an auditor u capabilities K and public effects E . A query Q records its auditor u , an optional subject s ($\perp$ if absent), an optional policy digest h , the snapshot η , an expiry δ , and an auditor nonce v . It also lists questions $\theta = x:\varphi$, each a formula with a name. A formula can compare a term τ with a bound c in the same unit. Here bal totals the notes held at time t in asset a , and val totals their value in a numeraire N such as USD. Likewise, count counts the records of a relation R , and sum totals their field f . A formula can also ask whether some, no, or every record chosen by a filter F qualifies; in $\forall R.F \Rightarrow G$, every record of R passing F must also pass G . Filters compare fields with values, intervals I , and unions $\bar{L}$ of registered address lists. Finally, a formula can test a valid, unrevoked credential claim cl from issuer is at time t , test another formula in every or some hour, day, week, or month of an interval, combine formulas, or refer to an earlier answer x . Every formula has finite depth and reads only account history and registered inputs.

History and denotations. A query looks at the subject’s history M up to the snapshot. M contains every note under any receiving key the subject had registered by then, including retired ones. M has five relations. deposit holds each deposit’s source, asset, and amount. trade holds each EXECUTE that spends an account note, with its venue, input, and measured output. transfer holds private transfers in or out, withdrawal holds withdrawals and their recipients, and received holds each private transfer output the account can spend, including change. M also holds the set $\mathcal{H}$ of notes the account ever held, with their creation and spend times, and the subject’s credentials. A holding’s source field is its set of direct deposit sources; both list membership and exclusion require a nonempty set whose every member satisfies the test. A trade or a withdrawal counts for an account when one of its notes takes part, and a date used as a time point means the last second of its UTC day. We write $[[\varphi]](M)$ for the answer to φ over M , and $R(w)$ for the records of R up to the snapshot within an optional window w . $\mathcal{H}(t)$ is the set of notes created by time t and still unspent at t . Below, M is implicit, r denotes a record, and o a holding. Dots select fields, $F(r)$ means that r passes F , a_r is the summed amount’s asset, and $|S|$ counts the members of a set S . The main denotations are

$$\begin{aligned} [[\exists R.F]] &= |\{r \in R(w) : F(r)\}| > 0, \\ [[\text{count}(R, F)]] &= |\{r \in R(w) : F(r)\}|, \\ [[\text{sum}(R, f, N, F)]] &= \sum_{r \in R(w), F(r)} \text{px}_N(a_r, r.f, r.\text{time}), \\ [[\text{bal}(a, F, t)]] &= \sum_{o \in \mathcal{H}(t), o.a=a, F(o)} o.\text{amount}, \end{aligned} \quad (4)$$

where px_N uses the latest registered price at or before the valuation time to convert an asset amount into numeraire N , and fails if the price is missing or stale. For an atomic amount x and price p , with asset, price, and numeraire decimal counts d_a, d_p, d_N , it returns

$\lfloor xp 10^{d_N} / 10^{d_a+d_p} \rfloor$ atomic units of N . Each amount is rounded down ($\lfloor \cdot \rfloor$) before summation. val prices holdings in the same way. $\forall$ requires its condition for every selected record. every_p requires its body in every calendar period of the interval, and some_p in at least one. Each period is clipped to the interval, and its relations contain only records in that span. Time points and window ends must not follow the snapshot. Without an explicit time, balances, values, and credentials use the span’s last second, or the snapshot outside a window. Windows do not nest. A reference takes the earlier answer. Every operand is evaluated, and all arithmetic is exact.

5.2 Type Checking and Certification

Units and capabilities. The type checker gives each term a sort. A balance of asset a has sort $\text{Amount}[a]$, a converted sum or value has sort $\text{Value}[N]$, and a count has sort Count . A comparison needs a bound of the same sort, so $\text{balance}(\text{USDC}) >= 1 \text{ stataUSDC}$ is ill typed, and a sum without conversion must filter its records to one asset. The checker also collects the capabilities $K(\varphi)$ that a question uses and its public effects $E(\varphi)$. These effects include the auditor, the snapshot, and any list, price source, window, or issuer it uses. A grant $\text{grant}(u', K, E)$ allows the question only if $u' = u$, $K(\varphi) \subseteq K$, and $E(\varphi) \subseteq E$. The answer then reveals its Yes/No bit and at most those effects. A query that refers to the holder’s disclosure policy compiles only if a grant allows every question.

Certification. Let $\hat{M} \subseteq M$ be the records that a holder’s proof uses. The proof shows that each is a distinct record of the subject, with all its fields, such as a note’s spend time. Both histories use the same authenticated registry inputs, and the formula must be defined on both. The checker decides separately what evidence can prove a Yes and what can prove a No.

- φ is *upward* if $[[\varphi]](\hat{M}) \Rightarrow [[\varphi]](M)$ for every such $\hat{M}$. More records cannot then overturn a Yes, so the records are a *witness* for Yes, as enough qualifying notes are for a minimum balance.
- φ is *downward* if $\neg[[\varphi]](\hat{M}) \Rightarrow \neg[[\varphi]](M)$. The records are then a witness for No, as enough notes to exceed an upper bound are.
- Otherwise, the checker requires the *complete history*, a proof of the history-completeness relation $\mathcal{R}_{\text{hist}}$ requiring $\hat{M} = M$. For example, a few deposits outside a prohibited list cannot show that all deposits are outside it.

The *profile* $\Gamma \vdash \varphi : \text{Ans}[\kappa^+, \kappa^-]$ records, under the registry Γ , the methods κ^+ for a Yes and κ^- for a No. Each method is *wit* (a witness) or *cmp* (the complete history). Here $\vdash$ means that the checker assigns a profile, and $\text{wit} \sqsubseteq \text{cmp}$ means a witness requires no more evidence than complete history. Rules put premises above the line and inferred profiles below, for well-typed formulas. Lower bounds, existence, and a credential test about a given subject are upward.

$$\begin{array}{c} \frac{\triangleright \in \{\geq, >\}}{\Gamma \vdash \tau \triangleright c : \text{Ans}[\text{wit}, \text{cmp}]} \quad \frac{}{\Gamma \vdash \exists R.F : \text{Ans}[\text{wit}, \text{cmp}]} \\ \frac{s \neq \perp}{\Gamma \vdash \text{attested}(is, cl, t) : \text{Ans}[\text{wit}, \text{cmp}]} \end{array} \quad (5)$$

Upper bounds and absence reverse these methods, because $\tau \leq c$ is $\neg(\tau > c)$ and $\forall R.F \Rightarrow G$ is $\neg \exists R.(F \wedge \neg G)$. Negation swaps them, and $\wedge$ (and) and $\vee$ (or) join them. The join $\sqcup$ takes *cmp* if either

operand needs it and wit otherwise.

$$\frac{\Gamma \vdash \varphi : \text{Ans}[\kappa^+, \kappa^-]}{\Gamma \vdash \neg \varphi : \text{Ans}[\kappa^-, \kappa^+]} \quad \frac{\Gamma \vdash \varphi_i : \text{Ans}[\kappa_i^+, \kappa_i^-]}{\Gamma \vdash \varphi_1 \circ \varphi_2 : \text{Ans}[\kappa_1^+ \sqcup \kappa_2^+, \kappa_1^- \sqcup \kappa_2^-]} \quad (6)$$

for $\circ \in \{\wedge, \vee\}$, with a premise for each operand $i \in \{1, 2\}$. Comparisons with $=$ or $\neq$ get $\text{Ans}[\text{cmp}, \text{cmp}]$. A window takes its body's profile, and a reference takes its question's profile. Without a subject, only witnessed answers are allowed.

PROPOSITION 5.1 (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.*

More records can only enlarge $\{r \in R(w) : F(r)\}$ and $\mathcal{H}(t)$. Existence, counts, and valid credentials can therefore only grow, and so can sums, balances, and values, because amounts and prices are non-negative. A lower bound that holds on $\hat{M}$ thus still holds on M , and an upper bound that fails on $\hat{M}$ still fails on M . Negation swaps these directions, and the connectives and finite windows keep them, so induction on the typing derivation proves the proposition.

In the example query, a few notes witness a Yes to `over_1m` and to `held_sep_30`, and a single credential witnesses a Yes to `verified_entity`. A No to `sanctioned_funds`, however, needs all deposits. A Yes to `within_limits` needs every day's trading, although one day above the limit witnesses a No. By the join, the checker assigns both answers to `cleared` a complete-history profile.

Encoding into zkSNARKs. FPL compiles supported positive lower-bound questions into statements of zkSNARK relations. The *notes relation* checks that the subject's notes, received notes, or deposits reach a threshold, and the *spend-records relation* checks the same for its trades or withdrawals. The holder proves the statement on its own device, and the auditor verifies the proof against the note commitments. A proof reveals only the answer, a hash of the query, blinded accumulator commitments, and an *account tag*, which ties the proof to all of the subject's receiving addresses, so the auditor learns neither which records the holder counted nor how many, up to a preset cap. By the knowledge soundness of the zkSNARK and Proposition 5.1, every answer that the auditor accepts holds on the subject's complete history. The current compiler refuses a program containing any unsupported question, naming each refusal, including source-list tests, pricing, credentials, windows, transfer amounts, and negative or complete-history answers. A missing or failed proof leaves a question unanswered; it is never a No.

6 Implementation

We implemented Fuyu on Ethereum. Solidity contracts implement the Pool and venue adapters for Uniswap V3 and lending vaults. We write the circuits in Circom and prove them with Groth16 [22] in snarkjs, in a browser wallet that also scans for payments. The spend circuit has 52,429 constraints, and the Pool's verifier checks a spend proof for about 444k gas. The partial-decapsulation filter below is evaluated in the scanning harness and used by the wallet; tag matches undergo full decapsulation and commitment validation before a note is accepted.

Our proofs use the BN254 curve, because Ethereum supports it with precompiled contracts. A quantum computer that computes

Table 1: Fuyu and privacy systems on Ethereum and OP Mainnet. ● means yes or private, ○ partly or at risk, and ○ no or exposed. Link ratings assume normal private operations and a sufficient anonymity set.

	Tornado Cash [47]	Privacy Pools v1 [11]	RAILGUN [52]	Privacy Boost [48]	Fuyu
<i>Functionality and control (RQ1)</i>					
Private transfers	○	○	●	●	●
DeFi applications	○	○	●	●	●
Decentralization	●	●	●	●	●
History hidden from third parties	●	●	●	○	●
Zero-knowledge proof of funds	○	○	○	○	●
<i>Security (RQ2), link privacy against</i>					
A chain analyst	●	●	●	●	●
A future quantum adversary	●	●	●	○	●
<i>Security (RQ2), the worst case</i>					
Exit window before upgrades	●	●	●	○	●

discrete logarithms on BN254 could therefore forge a spend proof and withdraw funds it does not own. The circuits do not depend on Groth16, however, so a successor Pool could verify them with a hash-based proof system such as a scalable transparent argument of knowledge (STARK) [8], which resists quantum forgery. Groth16 is also perfectly zero-knowledge for a well-formed proving key, so recorded proofs reveal nothing even to that computer.

7 Evaluation

Our evaluation answers three research questions (RQs). **RQ1** (functionality and control) asks what a holder can do privately and who can finalize its trades or read its history. **RQ2** (cost and security) asks what each operation costs and what an adversary can learn. **RQ3** (expressiveness) asks which compliance questions FPL can express and how long their proofs take. Table 1 answers RQ1 and RQ2 for Fuyu and four privacy protocols deployed on Ethereum and OP Mainnet. Following L2BEAT [31], it considers a *chain analyst*, who matches public amounts and timing, and a *future quantum adversary*, who saves today's data for a quantum computer.

Experimental setup. We deployed the Pool on a fork of Ethereum mainnet and called the deployed Uniswap V3, Aave, and sDAI contracts. Proving times are medians of at least three runs on an AMD EPYC 9654 server, natively with rapidsnark on 16 cores and in a browser on four cores. Times for finding payments benchmark the delivery-tag filter with the wallet's JavaScript cryptography library on one core.

7.1 Functionality and Control (RQ1)

DeFi applications. Fuyu supports non-atomic DeFi operations through ticket adapters. RAILGUN measures a venue's output and

shields it again within the transaction that calls the venue [52]; ticket adapters can likewise represent pending positions between calls. Privacy Boost’s Earn credits a vault’s output through a later deposit proof, which its operator builds [49], so we rate its DeFi support as partial. Tornado Cash here denotes its original fixed-denomination Ethereum pools; these and Privacy Pools v1 offer no DeFi operations.

Decentralization and history. Tornado Cash, RAILGUN, and Fuyu need no operator, and anyone can finalize a Fuyu trade. A Privacy Pools v1 withdrawal stays private only if the pool’s association set provider has approved the deposit, and the depositor can otherwise exit only publicly [11]. Privacy Boost depends on a centralized operator, which runs normal private operations inside a *trusted execution environment* (TEE), a sealed area of a processor that is meant to hide its data even from the machine’s owner. Its privacy therefore rests on trusting this hardware and the operator, who can also give registered auditors an account’s full history [48]. A Fuyu auditor gets only the answers that the holder chooses, whereas a shared viewing key shows far more. A *zero-knowledge proof of funds*, which shows that a balance reaches a bound without revealing the account’s notes, is offered only by Fuyu.

Results for RQ1. Fuyu supports private transfers and non-atomic DeFi, with permissionless trade finalization and holder-selected disclosure.

7.2 Cost and Security (RQ2)

Gas. On a new Pool, Fuyu’s deposits and transfers use less gas than RAILGUN’s (Figure 7).

DeFi trades. For the same 0.1 WETH swap on Uniswap V3, Fuyu used 1.20M gas against RAILGUN’s 1.35M, saving 11%. For the same 500 USDC supply to Aave, Fuyu used 1.32M against RAILGUN’s 1.46M, saving 10%. Both start with one funded note in newly deployed Pools on the same Ethereum fork; each operation consumes one private note and returns its result privately. RAILGUN’s shield and unshield fees remain enabled. These totals include gas spent inside the venues.

Finding payments. Fuyu’s delivery-tag filter checks a note’s delivery tag in 0.14 ms, 5.7× faster than a full ML-KEM decapsulation and 11× faster than an X25519 key exchange in the same library, whereas RAILGUN’s wallet spends 0.83 ms on each transfer output [54]. Fuyu’s delivery-tag filter identified its outputs in a history of one million notes in 2.4 min, 6.2× faster than RAILGUN’s 14.6 min.

Proving. Fuyu’s spend proof takes 6.3 s in a browser and 0.32 s natively, including the encryption of its outputs and the fields of a trade.

Exit. No party can pause the Pool or move its notes. Holders of the zero-controller notes described here can always exit, revealing a deposit’s source if screening prevents private spending. This is an infinite exit window under the assumptions of Section 3.2. Table 1 rates the window before an upgrade can prevent exit: infinite for Fuyu, Tornado Cash, and Privacy Pools v1, finite for RAILGUN, and absent for Privacy Boost. Privacy Boost does offer forced withdrawals without its TEE, but its owner can replace the verifier without an upgrade delay [50, 51].

Future privacy. Of the compared privacy protocols that support private payments and DeFi, only Fuyu keeps payments unlinkable against a future quantum computer, because it protects note owners and payment openings with hashes, symmetric encryption, and post-quantum key encapsulation.

Results for RQ2. On one million notes, Fuyu’s delivery-tag filter is 6.2× faster than RAILGUN’s wallet. On a new Pool, the matched swap and supply operations use 10–11% less total gas than RAILGUN.

7.3 Expressiveness (RQ3)

We constructed sixteen representative compliance questions about holdings, sources of funds, activity, and identity, informed by the anti-money-laundering guidance of the Financial Action Task Force (FATF) [18], the SEC’s definition of an accredited investor [60], and the covenants that lenders set (Appendix A). FPL expresses all sixteen. Its type system assigns witness-Yes profiles to twelve of them and complete-history Yes profiles to the other four. The current proof compiler supports five of these questions. A zero-knowledge proof of funds over up to 64 notes took 15 s natively and 201 s in a browser, and the auditor verified it in 0.34 s.

Results for RQ3. FPL expresses all sixteen compliance questions and assigns witness-Yes profiles to twelve of them. The current proof compiler supports five. A proof of funds over up to 64 notes took 15 s natively and 201 s in a browser.

8 Related Work

Shielded payments. Zerocash introduced the note model [9]. Zexe extended it to private computation [10], Tornado Cash brought mixing to Ethereum [47], and Namada’s MASP and MantaPay shield many assets in one set [14, 23]. Public timing and amounts can narrow the set of possible senders even when a proof hides the spent note [29], so Fuyu lists what each operation reveals. Zcash’s ZIP 2005 proposes a way to recover funds once quantum computers arrive, but not to keep earlier payments private [27].

UTXO and account models. The Extended UTXO model of Cardano and Ergo and the Cell model of Nervos keep contract state inside UTXOs, as data guarded by scripts [13, 17, 62]. Their transactions specify every output before they are sent. In SundaeSwap on Cardano, for example, users lock funds in order UTXOs, and authorized scoopers execute batches of orders against the UTXO that holds a pool’s reserves [33]. Everything stays public. Fuyu instead keeps private notes as UTXOs and leaves state in account-model contracts, whose effect the Pool measures at execution.

Private DeFi. Hawk, Zapper, and zkay protect contract state and execution [30, 58, 59]. Aztec’s alpha mainnet, launched in 2026, combines private functions proved on the user’s device with public ones [6, 55]. Fuyu, in contrast, keeps the venue public and measures what it returns. Penumbra batches swaps on its own chain, and a user claims the output in a later block with a proof that needs its full viewing key [46]. Aztec Connect batched bridge calls [5], and zkBob’s operators submit its tree updates [66]. RAILGUN’s relay contract makes public calls and re-shields the returned tokens in the same transaction [52]. Privacy Pools v2 adds private

transfers, swaps, and yield operations [1], but uses X25519 for note delivery [2]. Our comparison uses v1. Nocturne, shut down in 2024 [40], chose an operation’s refund owner before the operation ran and turned the measured balances into notes in the same transaction, which any account could submit [39]. Hinkal and Privacy Boost’s Earn also measure venue output onchain [25, 49]. Fuyu also supports non-atomic operations, whose pending positions it holds privately as tickets. Zama’s FHEVM computes on values encrypted with fully homomorphic encryption through a threshold key-management network [63], which Fuyu’s payments do not need.

Accountable privacy. Garman, Green, and Miers add policy enforcement and selective tracing to shielded payments [20]. Hinkal proves on every spend, against current screening data, that the funds do not come from a flagged deposit [24], and Nocturne screened deposits offchain with delays and caps [39]. Panther divides its pool into zones whose managers set KYC and transaction rules backed by zero-knowledge attestations, and it has run on Polygon and Base since 2026 [43–45]. Canton gives institutions sub-transaction privacy without zero-knowledge proofs, because it sends each stakeholder’s node only its own view of a transaction [12, 16]. zkLedger can give complete answers about totals, because its ledger commits to every bank’s part of every transaction [37]. A Fuyu note tree, however, cannot show that a holder presented every record, so FPL separates the answers that a witness proves from those that need the complete history.

Privacy languages. zkay checks privacy annotations on contracts, Zapper compiles private contract execution, and ZQL compiles private queries [19, 58, 59]. vSQL proves that answers over outsourced data are correct and complete [64], and work on incomplete databases asks which answers hold in every completion [28]. FPL instead sorts each Yes and No by the evidence that proves it.

9 Conclusion

On a public blockchain, a fund that moves its assets onchain exposes every payment, permanently. Fuyu addresses this for smart-contract blockchains. Hashes, symmetric encryption, and ML-KEM-768 keep recorded payments as private against a future quantum computer as they are today, and because a recipient checks a delivery tag with a partial decryption, the delivery-tag filter finds the recipient’s outputs on a ledger with one million notes 6.2× faster than in RAILGUN. Deferred notes let the Pool measure each DeFi trade’s output and make finalization permissionless, and FPL makes disclosure programmable.

As a result, the fund can pay its suppliers without revealing whom it paid or how much. It can also answer its lender with proofs that reveal only the answers it chooses. As for any trade at a public venue, however, Fuyu’s trade amounts stay public. Even so, we believe Fuyu lets institutions bring their assets onchain while keeping what they have today.

References

- [1] 0xbow. 2026. Ethereum Mainnet. Privacy Pools v2 Documentation. <https://privacy-pools-v2-docs.vercel.app/deployments/mainnet>
- [2] 0xbow. 2026. Keys. Privacy Pools v2 Documentation. <https://privacy-pools-v2-docs.vercel.app/concepts/keys>

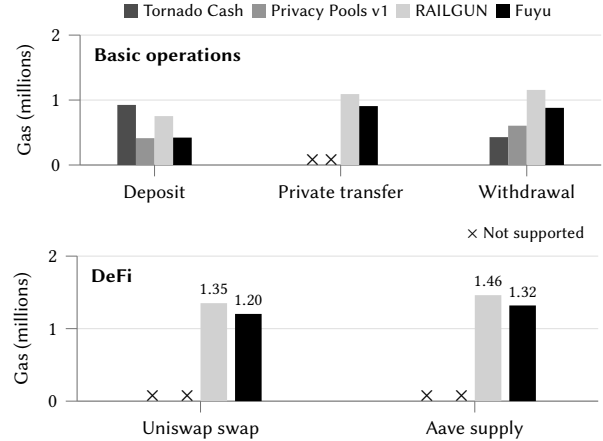


Figure 7: Gas consumption on Ethereum.

- [3] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. 2021. Uniswap v3 Core. Whitepaper. <https://app.uniswap.org/whitepaper-v3.pdf>
- [4] 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>
- [5] 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>.
- [6] Aztec Labs. 2026. Alpha Network. Aztec Documentation. <https://docs.aztec.network/participate/alpha> See also the pages Call Types and Transactions on Aztec.
- [7] 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
- [8] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2019. Scalable Zero Knowledge with No Trusted Setup. In *Advances in Cryptology – CRYPTO 2019, Part III (Lecture Notes in Computer Science, Vol. 11694)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, Cham, 701–732. doi:10.1007/978-3-030-26954-8_23
- [9] 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
- [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 Illum, 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] Canton Network. 2026. Privacy Model Explained. Canton Network Docs. <https://docs.canton.network/overview/learn/privacy-model> See also the page Trust Model Overview.
- [13] Manuel M. T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Michael Peyton Jones, and Philip Wadler. 2020. The Extended UTXO Model. In *Financial Cryptography and Data Security, FC 2020 International Workshops (Lecture Notes in Computer Science, Vol. 12063)*. Springer, 525–539. doi:10.1007/978-3-030-54455-3_37
- [14] Shumo Chu, Yu Xia, and Zhenfei Zhang. 2021. Manta: a Plug and Play Private DeFi Stack. Cryptology ePrint Archive, Paper 2021/743. <https://eprint.iacr.org/2021/743>
- [15] 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
- [16] Digital Asset. 2024. Canton Network: A Network of Networks for Smart Contract Applications. Whitepaper. <https://www.canton.network/hubs/Canton/Canton%20Network%20-%20White%20Paper.pdf> Updated January 2024.
 - [17] Ergo Platform. 2026. Extending the Power of the UTXO Model. Ergo Documentation. <https://docs.ergoplatform.com/dev/protocol/eutxo/>
 - [18] 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.
 - [19] 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>
 - [20] 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
 - [21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. 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>
 - [22] 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
 - [23] HeliAx AG. 2023. Multi-Asset Shielded Pool Specification. GitHub repository [namada-net/masp, docs/multi-asset-shielded-pool.pdf](https://github.com/namada-net/masp/blob/main/docs/multi-asset-shielded-pool.pdf). <https://github.com/namada-net/masp/blob/main/docs/multi-asset-shielded-pool.pdf> Dated September 21, 2023; based on the Zcash protocol specification by Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox.
 - [24] Hinkal. 2026. Compliance. Hinkal Documentation. <https://hinkal-team.gitbook.io/hinkal/security-and-compliance/compliance> Section Security and Compliance; see also the pages Access Tokens and User Authentication, and Risks.
 - [25] Hinkal. 2026. Swaps. Hinkal Documentation. <https://hinkal-team.gitbook.io/hinkal/protocol/transactions/swaps> Section Protocol, Transactions; see also the page Emporium.
 - [26] 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).
 - [27] Daira-Emma Hopwood and Jack Grigg. 2025. ZIP 2005: Ironwood Quantum Recoverability. Zcash Improvement Proposal, Proposed. <https://zips.z.cash/zip-2005> Created 2025-03-31; activates with the NU6.3 network upgrade.
 - [28] Tomasz Imieliński and Witold Lipski. 1984. Incomplete Information in Relational Databases. *J. ACM* 31, 4 (1984), 761–791. doi:10.1145/1634.1886
 - [29] 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>
 - [30] 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
 - [31] L2BEAT. 2026. Privacy. <https://l2beat.com/privacy/summary>
 - [32] L2BEAT Research. 2026. Onchain Privacy Best Practice. <https://l2beat.com/publications/privacy-best-practices>
 - [33] Pi Lanningham. 2024. SundaeSwap v3 Protocol Specification. Sundae Labs. <https://cdn.sundaeswap.finance/SundaeV3.pdf>
 - [34] Varun Maram and Keita Xagawa. 2023. Post-Quantum Anonymity of Kyber. In *Public-Key Cryptography (PKC 2023)*. Springer. <https://eprint.iacr.org/2022/1696>
 - [35] 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
 - [36] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. <https://bitcoin.org/bitcoin.pdf>
 - [37] 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>
 - [38] 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
 - [39] Nocturne. 2023. Protocol Overview. Nocturne Documentation. <https://nocturne-xyz.gitbook.io/nocturne/introduction/protocol-overview> See also the pages Compliance, Deposit Screener, and Operations.
 - [40] Nocturne. 2024. Bittersweet news. Today we’re shutting down Nocturne v1 Post on X by nocturne_xyz. https://x.com/nocturne_xyz/status/1749510390906511693 Thread of 22 January 2024; deposits, swaps, and staking paused, withdrawals open for one year.
 - [41] Jeroen Offerijns, Alina Sinelnikova, Vikram Arun, Joey Santoro, Farhaan Ali, and João Martins. 2023. ERC-7540: Asynchronous ERC-4626 Tokenized Vaults. Ethereum Improvement Proposals, no. 7540. <https://eips.ethereum.org/EIPS/eip-7540>
 - [42] 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 Lucianna Kiffer (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 24:1–24:24. doi:10.4230/LIPIcs.AFT.2024.24
 - [43] Panther Protocol. 2024. Shielded Pool. Panther Protocol Documentation. <https://docs.pantherprotocol.io/docs/panther-core/panther-core/shielded-pool> See also the pages zAssets, Zone Manager, Panther Compliance, and Forensic Reveals.
 - [44] Panther Team. 2026. Panther Brings Programmable Privacy to Base. Panther Protocol Blog. <https://blog.pantherprotocol.io/panther-brings-programmable-privacy-to-base/> Published 15 September 2026.
 - [45] Panther Team. 2026. Programmable Privacy Is Live: Panther Protocol Deploys on Polygon. Panther Protocol Blog. <https://blog.pantherprotocol.io/programmable-privacy-is-live-panther-protocol-deploys-on-polygon/> Published 7 May 2026.
 - [46] 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>.
 - [47] 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.
 - [48] Privacy Boost. 2026. Auditability. Privacy Boost Documentation. <https://docs.privacyboost.io/product/auditability>
 - [49] Privacy Boost. 2026. Earn (ERC-4626 Vaults). Privacy Boost Documentation. <https://docs.privacyboost.io/sdk/concepts/earn>
 - [50] Privacy Boost. 2026. PrivacyBoost: Verifier Configuration. Protocol source, commit 5792c139b9529ed80643262d75b5489055be11b0. <https://github.com/sunnyside-io/privacy-boost-protocol/blob/5792c139b9529ed80643262d75b5489055be11b0/contracts/src/PrivacyBoost.sol#L391-L408>
 - [51] Privacy Boost. 2026. Protocol Deep Dive. Privacy Boost Documentation. <https://docs.privacyboost.io/technical-explainer/protocol>
 - [52] RAILGUN. 2025. Cross-Contract Calls. RAILGUN Developer Guide. <https://docs.railgun.org/developer-guide/wallet/transactions/cross-contract-calls>
 - [53] RAILGUN. 2026. Private Proofs of Innocence. RAILGUN Wiki. <https://docs.railgun.org/wiki/assurance/private-proofs-of-innocence>
 - [54] RAILGUN Contributors. 2026. @railgun-community/engine. npm package, version 9.8.0. <https://www.npmjs.com/package/@railgun-community/engine>
 - [55] Kelsey Ruiz. 2026. Announcing the Alpha Network. Aztec Network Blog. <https://aztec.network/blog/announcing-the-alpha-network> Published 31 March 2026.
 - [56] RWA.xyz. 2026. Tokenized Stocks. <https://app.rwa.xyz/stocks> Distributed Value; dashboard snapshot accessed 29 September 2026.
 - [57] 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>
 - [58] 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
 - [59] Samuel Steffen, Benjamin Bichsel, and Martin Vechev. 2022. Zapper: Smart Contracts with Data and Identity Privacy. In *Proceedings of the 2022 ACM*

```

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 8: The sixteen benchmark questions in FPL. The query’s 32-byte nonce is shortened, and the snapshot height belongs to the benchmark fixture. OFAC@17 is version 17 of the sanctioned-address list, and KycIssuer is a credential issuer.

- SIGSAC Conference on Computer and Communications Security (CCS). ACM, New York, NY, USA, 2735–2749. doi:10.1145/3548606.3560622
- [60] U.S. Securities and Exchange Commission. 2026. Definitions and Terms Used in Regulation D. 17 CFR 230.501. <https://www.ecfr.gov/current/title-17/section-230.501>
- [61] World Federation of Exchanges. 2026. H1 2026 Market Highlights. <https://www.world-exchanges.org/our-work/articles/h1-2026-market-highlights> Chart 1: Domestic market capitalisation; June 2026 data.
- [62] Jan Xie. 2018. Nervos CKB: A Common Knowledge Base for Crypto-Economy. Nervos Network RFC 0002. <https://github.com/nervosnetwork/rfcs/blob/master/rfcs/0002-ckb/0002-ckb.md>
- [63] Zama. 2026. Zama Confidential Blockchain Protocol Litepaper. <https://docs.zama.org/protocol/zama-protocol-litepaper> Protocol documentation, accessed 29 September 2026.
- [64] 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
- [65] ZK-Kit. 2024. LeanIMT: Lean Incremental Merkle Tree. ZK-Kit repository. <https://github.com/zk-kit/zk-kit/tree/main/packages/lean-imt>
- [66] zkBob. 2024. ZkBobPool Contract (src/zkbob/ZkBobPool.sol). GitHub repository 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 Benchmark Questions

Figure 8 lists the sixteen compliance questions of Section 7.3 in FPL.

B Security Proofs

We prove the properties of Section 4.5 under Section 3.2, whose four cryptographic assumptions we state here in full.

- (1) *Sound, zero-knowledge proofs.* The setup of each relation leaves its trapdoor unknown to the adversary. An accepted proof then implies knowledge of a witness for its statement, which Groth proves for Groth16 under that setup condition in the generic group model [22], and a proof made with a well-formed proving key is perfectly zero-knowledge.
- (2) *Secure hash functions.* Poseidon, SHA-256, Keccak-256, and X-Wing’s SHA3-256 combiner are collision resistant, so commitments are binding. On a secret high-entropy input, they are pseudorandom, also against a later quantum adversary, so commitments are hiding, and nullifiers, owner tags, and delivery tags look random.
- (3) *Post-quantum key encapsulation.* We assume that ML-KEM-768 is IND-CCA secure and anonymous against a later quantum adversary; related analysis covers Kyber [34]. For the delivery envelope, we additionally assume that publishing the derived tag preserves recipient anonymity and the combined secret’s confidentiality.
- (4) *Secure symmetric encryption and key derivation.* Note-opening and sender-recovery encryption is semantically secure, and domain-separated key derivation yields pseudorandom keys and tweaks from secret seeds or shared secrets, also against a quantum adversary.

Ownership and no double spending. Knowledge soundness gives an opening for each input commitment c under an accepted root (Section 4.1), together with a spend secret k whose owner root R and the note’s secret tweak t hash to its owner tag o . By the hash assumptions, finding another such secret is infeasible, and k and c determine the note’s one nullifier. The Pool rejects the nullifier if it is already spent, and otherwise records it in the same step as the rest of the operation.

Solvency. Under the asset assumptions of Section 3.2, let Bal_a be the Pool’s balance of asset a , and U_a and Res_a the totals of its unspent notes and pending receipts. A deposit credits exactly what the Pool collects, and Equation (3) accounts for every range-checked amount of a spend. A trade credits only the measured balance change q , and FINALIZE appends only stored commitments and deletes the receipt, so no receipt is appended twice. Every operation thus moves value between the Pool’s custody, unspent notes, and pending receipts without counting it twice, so $Bal_a \geq U_a + Res_a$.

Unlinkability. Zero knowledge hides the witnesses, and hiding commitments and semantically secure ciphertexts hide the notes. Nullifiers and delivery tags are pseudorandom under their keys, and fresh tweaks make owner tags unlinkable. An observer’s view therefore follows from the fields that each operation makes public and the order of transactions, although actual anonymity still depends on how people use the Pool. Against a later quantum adversary, Groth16 proofs stay perfectly zero-knowledge for a well-formed proving key, and no circuit uses an elliptic curve. Owners, commitments, and nullifiers are Poseidon hashes, and the key to each encrypted payment opening rests on ML-KEM-768 once X25519 falls [7].