Bonsai: Scalable Private Payments

Patrick O'Grady, Lúcás Críostóir Meier, and Guru-Vamsi Policharla

Commonware, Inc.

Abstract

Virtually all deployed private payment systems publish a nullifier for every transaction to prevent double spending. At a million transactions per second, the nullifier set grows by a petabyte each year. In this work, we tackle the question of sustaining massive throughput in private payments while ensuring the system can be run on commodity hardware.

We construct Bonsai, an account-based private payment scheme where validators store a single commitment per account and never store any nullifiers. Instead, each user privately maintains the nullifiers of the payments it has received, and can prune older nullifiers to cold storage so that its active state remains small. An external observer only learns that an account performed *some* action (send/receive) but never learns the amount or counterparty of a payment.

To verify proofs at this rate, we add zero-knowledge to PARI [USENIX '26] with no increase in proof size and negligible prover overhead, and use it with a batch verification strategy. Our prototype verifies over a million operations per second on an M5 MacBook Pro.

1 Introduction

Private payments have evolved over four decades, from Chaum's electronic cash [Cha82] to shielded notes on public ledgers [BCG+14, HBHW26], shielded computation [BCG+20], ring-based anonymity [Noe15, GK15, LRR+19], and confidential amounts [BAZB20]. The design space spans a broad range of privacy guarantees, trust models, setup assumptions, and performance tradeoffs [BCMR24]. Implemented systems report performance on the order of a thousand transactions per second [TBA+22, BLKZ25] but this is not sufficient for real world systems that may see hundreds of thousands of transactions per second, especially with the rise of agentic commerce.

This gap in throughput motivates our central question:

How do we process one million private transactions per second on commodity hardware?

To get there, we need to rethink what validators and wallets are responsible for. Concretely, a system that runs at this rate indefinitely needs:

1. **Succinct validator state.** Validator storage may grow with the number of accounts, but not with the number of transactions, and processing a transaction takes a constant amount of work regardless of how many accounts exist and anonymity set (if any).

2. **Offline wallets.** Wallets can go offline for arbitrarily long periods and, upon returning, send and receive payments without replaying intervening ledger history. The work done by a wallet only depends on the transactions they participate in.

The first requirement is enforced by limits on storage and computation. Virtually every private payment system that hides the link between senders and receivers publishes a unique nullifier for every transaction to prevent double spending. This set of nullifiers is typically held by the validators indefinitely. At a million

TPS, validator state grows by a petabyte each year, and nullifiers must be held in fast storage and checked against every incoming transaction. Any new validator must synchronize a massive amount of data before participating. To avoid imposing an unreasonable hardware burden on validators, we demand that storage requirements only grow with the number of accounts for which validators can charge fees appropriately. We also demand that validator work is independent of the number of accounts/anonymity sets to ensure that the system can support the desired throughput without resorting to sharding or forcing small anonymity sets onto users.

The second requirement arises from usability of the system. Users should be allowed to go offline indefinitely and be able to resume payments without having to worry about the payments that happened while they were offline. In fact, for many applications such as payroll or subscription services, users may want to "set it and forget it" where a simple cron job periodically comes online, sends out/collects payments and goes offline without having to rely on an external service provider.

Our contributions. We construct Bonsai, a private payment scheme that verifies over a *million* operations per second on an M5 MacBook Pro. The ledger learns that a particular account came online and performed some action (a send/receive) but it does not learn the amount moved or to whom the funds were sent/received from. While this offers weaker on-chain privacy than shielded notes [BCG+14], it allows for a much more efficient solution to nullifier management. In practice the gap may be narrower: when a wallet submits transactions through an RPC node rather than its own node, that RPC node can already link each submitted transaction to the corresponding party.

Validators store a single 32-byte commitment per account, and every transaction is 256 bytes: a 32-byte account identifier, the new account commitment, a receipt, a state root, and a 128-byte proof. Importantly, validators do not need to remember transactions – they can be added to a merkle mountain range which can be extended using just the frontier ($O(\log(\mathsf{txs}))$ hashes).

Nullifier storage and management is delegated to users but we ensure that a user's nullifier storage only grows with the number of transactions they participated in. Because nullifiers remain private, we use the canonical ledger position of each send operation as the nullifier for its receipt. This ordering lets wallets move older nullifiers to cold storage, keeping only recent nullifiers and a frontier of logarithmic size on hand. No protocol changes are needed: wallets can update the nullifier tree for older receipts by retrieving the necessary nullifier data from cold storage. Like its namesake, each user's active nullifier state in Bonsai can be kept to a chosen window plus a logarithmic frontier. Total archived state still grows with lifetime activity.

2 Technical Overview

We now present a technical overview of Bonsai. The full construction can be found in Section 6. The main goal of Bonsai is to limit validator state growth proportional to the number of accounts but not with the throughput (number of payments) made in the system and therefore work in the account model which offers weaker on chain privacy than the shielded notes model [BCG+14, HBHW26] but as we will see, it offers a much more elegant solution to managing an ever growing set of nullifiers.

Transfers in Bonsai are a two step process:

- the sender first initiates the transfer by invoking the Send algorithm and posting a receipt on chain.
- the sender privately communicates the receipt opening information to the receiver. We assume a private channel from the sender to the receiver for simplicity but this can be replaced with an oblivious message routing system such as on-chain encrypted memos [HBHW26], fuzzy message detection [BLMG21] or oblivious message retrieval [LT22].
- the receiver then invokes the Receive algorithm to produce a zero-knowledge proof that can be submitted on chain to (anonymously) claim the funds addressed to them

This is reminiscent of ecash [Cha82] style payments but without a centralized intermediary that can track inflows/outflows of an account. In the base protocol described below, the ledger learns whether an account performed a send or a receive: a send publishes a receipt and appends it to the receipt MMR, whereas a receive publishes an MMR root anchor and does not append. We then describe a simple operation-hiding extension that makes the two operations indistinguishable.

More formally our protocol is a tuple of five algorithms $(\mathsf{Setup}, \mathsf{Create}, \mathsf{Register}, \mathsf{Send}, \mathsf{Receive})$. Setup samples the public parameters needed for commitments and zero-knowledge proofs and Create is a local algorithm run to sample randomness necessary to register an account. The public state of an account A is a single hiding commitment

$$\mathsf{com} = \mathsf{Com}_{\mathsf{acct}}(b, \mathsf{root}_{\mathsf{null}})$$

to its balance and the root of a sparse Merkle tree [DPP16] that holds the nullifiers of every payment A has received.

Send. Sen creates a hiding receipt

$$\rho = \mathsf{Com}_{\mathsf{rec}}(v, \mathsf{Sen}, \mathsf{Rec})$$

to the amount v, the sender, and the receiver. It additionally computes an updated account commitment

$$\mathsf{com}'_{\mathsf{Sen}} = \mathsf{Com}_{\mathsf{acct}}(b - v, \mathsf{root}_{\mathsf{null}})$$

and publishes $(\mathsf{Sen}, \mathsf{com}'_{\mathsf{Sen}}, \rho, \pi)$, where π proves that:

- the old commitment $\mathsf{com}_{\mathsf{Sen}} = \mathsf{Com}_{\mathsf{acct}}(b, \mathsf{root}_{\mathsf{null}})$,
- the new commitment $\mathsf{com}'_{\mathsf{Sen}} = \mathsf{Com}_{\mathsf{acct}}(b - v, \mathsf{root}_{\mathsf{null}})$
- v and $b - v$ are in the balance range
- the receipt ρ opens to $(v, \mathsf{Sen}, \mathsf{Rec})$.

Validators then verify π, update $\mathsf{com}_{\mathsf{Sen}} \to \mathsf{com}'_{\mathsf{Sen}}$ and append ρ to the MMR. Sen then forwards the opening of ρ and its position pid in MMR to Rec over the private channel. Validators only maintain:

- account commitments
- the frontier of the receipt Merkle Mountain Range MMR [Tod12]
- a set of recent MMR roots $\mathcal{T}$ against which they accept receive proofs

Receive. To prevent double spending, Rec uses the receipt's position as its nullifier, $\mathsf{null} = \mathsf{pid}$, and inserts it into its local nullifier tree. It then publishes $(\mathsf{Rec}, \mathsf{com}'_{\mathsf{Rec}}, \mathsf{root}_\rho, \pi)$ for some $\mathsf{root}_\rho \in \mathcal{T}$, where π proves that:

- some receipt ρ at some position pid under root_ρ opens to $(v, \cdot, \mathsf{Rec})$,
- the old commitment $\mathsf{com}_{\mathsf{Rec}} = \mathsf{Com}_{\mathsf{acct}}(b, \mathsf{root}_{\mathsf{null}})$,
- inserting pid into the tree with root $\mathsf{root}_{\mathsf{null}}$ yields $\mathsf{root}'_{\mathsf{null}}$, and
- the new commitment $\mathsf{com}'_{\mathsf{Rec}} = \mathsf{Com}_{\mathsf{acct}}(b + v, \mathsf{root}'_{\mathsf{null}})$.

Validators then verify π, check that $\mathsf{root}_\rho \in \mathcal{T}$, and update $\mathsf{com}_{\mathsf{Rec}} \to \mathsf{com}'_{\mathsf{Rec}}$. Nullifiers are never published on chain and are only maintained locally by each user, which is why the position itself can serve as the nullifier.

Operation hiding. The base protocol above reveals whether each transaction is a send or a receive. To hide the operation type, both branches publish the common record $\mathsf{tx}_{\mathsf{op}} = (A, \mathsf{com}', \rho, \mathsf{root}_\rho, \pi)$, where π proves that either the send or receive relation is satisfied. A send publishes a real receipt while a receive publishes an unspendable dummy receipt. The ledger appends ρ to the MMR in both cases (see Section 6).

Verification. [Gro16] is a popularly used proof system with a circuit specific CRS and a very efficient verifier, requiring a few pairings during verification. While there is some improvement when verifying batch of B proofs together [HBHW26, Appendix B.2], it still requires $O(B)$ pairings, each of which costs hundreds of microseconds (amortized).

PARI [DMS24, Eag25] is a SNARK for Square R1CS constraint systems, whose proof is two $\mathbb{G}_1$ elements and one field element. Its verifier checks a single batched KZG [KZG10] opening at a challenge point ζ amounting to three pairings. But verifying a batch of B proofs can be reduced to three multi-scalar multiplications of size B followed by three pairings [Pol26] – thus making verification of a batch of PARI proofs much faster than verifying a batch of [Gro16] proofs. As written in [DMS24, Eag25] PARI is not zero knowledge. In Section 7, we show that it can be made zero-knowledge without increasing the proof size and with negligible overhead on prover time.

Security. We define security using an ideal functionality for account-based private payments parametrized by a leakage function (Section 5). The functionality maintains balances and a log of payments, lets a corrupted sender withhold a payment's opening from the receiver, and broadcasts the leakage of each operation. We prove that Bonsai securely realizes the functionality in Theorem 1 and Theorem 2.

Performance. We implement ZK-PARI in Rust over BLS12-381 together with the send and receive circuits and evaluate it on an M5 MacBook Pro (Section 8). Proofs are 128 bytes and an individual verification takes about 0.7 ms, dominated by its three pairings. Batching removes this cost: a batch of 2^{16} proofs is verified at an amortized 11.7 μs per proof, a $64\times$ speedup, with the three multi-scalar multiplications accounting for over 90% of the time and the final pairings for less than 0.1%. This corresponds to $86{,}929$ proofs per second on one core and 1.05 million on 18 cores, or roughly $525{,}000$ payments per second since each payment is one send and one receive.

We only use a collision resistant hash function and benchmark two possible choices: Pedersen hash over Jubjub or Poseidon. The base receive relation is the more expensive of the two, as its cost is dominated by the sparse Merkle tree insertion and the receipt opening proof. Under Poseidon the send relation has $1{,}647$ R1CS constraints (0.14 s single-threaded, 0.03 s on 8 threads) and the receive relation has $30{,}729$ (1.6 s, 0.29 s). Under Pedersen the send relation has $16{,}109$ constraints (0.84 s, 0.14 s) and the receive relation has $398{,}111$ (20 s, 3.8 s). The operation-hiding relation $\mathcal{R}_{\mathsf{op}}$ shares the common gadgets of the two branches and costs only $\sim 2\%$ more constraints than the receive relation.

3 Related work

We group prior private-payment constructions by how they represent funds and hide transfers. For each approach, we discuss its main variants and the obstacles to meeting the two requirements from Section 1: bounded validator state and work, and wallets that can remain offline without having to replay intervening ledger history.

Ecash. Chaumian ecash represents money as bank-signed coins. Blind issuance lets a user withdraw a coin and later pay with it without linking the payment to the withdrawal [Cha82, Cha83, Cha90]. Online schemes check a coin's serial number at redemption and offline schemes defer contact with the bank and instead penalize double spending from conflicting payment transcripts [CFN90, OO90]. Many follow-up variants [CFT98, CHL05, CPST15, BPS19, CP93, Bra94] improve the efficiency and usability of the protocol but still rely on a global nullifier set that cannot be pruned without expiring coins.

Credential-based payments. A related approach represents funds by credentials that users can rerandomize and prove possession of without revealing their issuance history. Randomizable signatures [PS16] and threshold credentials such as Coconut [SAB+19] provide building blocks for distributing issuance across authorities. UTT [TBA+22] represents coins as commitments to an owner, serial number, and value, signed by a set of validators. To pay, a user rerandomizes its input coins and signatures, reveals their nullifiers, and proves ownership and value conservation before the committee signs new coins for the recipient and change. ZEF [BSKD22] instead binds coins to accounts and represents them as off-chain certificates, using Coconut's blind threshold issuance to hide the accounts and values of newly created coins from the issuing authorities. A payment consumes certified input coins and produces blinded output coins, with zero-knowledge proofs enforcing value conservation. Building on FastPay, clients collect quorum certificates for account operations, allowing sharded authorities to process payments asynchronously without reaching consensus on a global transaction order.

PLATYPUS [WKDC22] replaces individual coins with a bank-signed commitment to each user's balance and a serial number. The sender and receiver prove matching debit and credit updates against a shared commitment to the payment amount, keeping their previous account states and signatures hidden. The bank checks the proofs and the revealed serial numbers of both old states, then signs the new states; this makes successive account updates unlinkable while permitting private checks on balances and spending limits. PEREDI [KKS22] uses threshold blind signatures from distributed maintainers to certify private account states containing balances and cumulative payment counters. Both parties prove valid account updates and compliance with balance and transfer limits, reveal one-time tags to prevent state reuse, and collect signature shares on their new states. PARSCOIN [SKK24] adapts this signed-account architecture to stablecoins and separates the sender's debit from the receiver's credit. The sender proves a balance reduction and leaves an encrypted transfer object with the issuers. The receiver can later identify the object and prove entitlement to its value when updating its own account, allowing payment initiation while the receiver is offline. In these payment systems, consuming a coin or replacing a credential reveals a serial number or commitment that the authorities must remember to reject reuse.

Shielded note pools. These constructions replace issuer signatures with membership in a public set of coin commitments. A spender proves that it owns a valid commitment without revealing which one, and publishes a serial number or nullifier to prevent a second spend. Sander and Ta-Shma [ST99] issue coins by inserting commitments to user-chosen serial numbers into a public Merkle database, whose authenticated roots replace the bank's signatures. To pay, a user reveals a serial number and proves in zero knowledge that it knows an opening of some database commitment to that serial number, hiding the commitment and its authentication path.

Zerocash [BCG+14], the follow-up of Zerocoin [MGGR13], supports private payments by committing to notes containing a recipient's address, a value, and randomness, and accumulating these commitments in a Merkle tree. Its pour transaction consumes existing notes and creates new ones, with a zk-SNARK proving input membership, spending authority, correct output formation, and value conservation while hiding the input notes, recipients, and amounts. The transaction publishes new note commitments and input serial numbers derived using the owners' secret keys, while encrypted note openings let recipients discover and later spend their funds. Qurrency extends the private UTXO approach with post-quantum confidentiality and authorized auditing [CGG+26]. The obstacle in these approaches remains the state growth as the published nullifiers accumulate with transaction history.

Rings and decoy accounts. Another approach hides the spent output among a ring of candidates and proves ownership of one member, with a nullifier-like mechanism for preventing double spends. RingCT combines this mechanism with confidential amounts and one-time recipient addresses [Noe15]. One-out-of-many proofs [GK15] and Omniring [LRR+19] reduce proof size to logarithmic in the anonymity set. Here validators still retain the tags needed to detect double spending, resulting in a growing double-spend record.

Account-based variants instead hide the participants by updating real accounts together with decoys. Basic Zether [BAZB20] stores encrypted balances for each account and proves valid updates, hiding amounts

but revealing the link between sender and receiver. Its optional anonymity mode updates an entire chosen account set. Quisquis similarly updates real accounts together with decoys, using updatable public keys to avoid historical nullifiers [FMMO19]. Both approaches incur work linear in the chosen anonymity set and require users to retrieve its current state before making a payment.

Accounts and transfer receipts. These systems keep private balances in accounts but to transfer a coin, the sender debits its balance and publishes a coin or receipt that the receiver later collects. Separating debit and credit hides their link and allows the receiver to be offline when the sender initiates payment.

BlockMaze [GWY+19] maintains both public and committed private balances under persistent account addresses. A send transaction reduces the sender's private balance and publishes a receipt that hides the amount and recipient, together with a zk-SNARK proving the debit is correct. The recipient later deposits the payment by proving ownership of a receipt in a Merkle tree of receipts and a matching balance increase, without revealing which receipt is being claimed. A revealed serial number prevents repeated collection. Veksel [CHA22] uses a similar send/receive pattern, but focuses on cryptographic efficiency without a trusted setup. Veksel obtains a constant-size proof by combining an accumulator membership proof with a rerandomization proof. DART [SKKK25] additionally hides the account being updated by having the user prove ownership and a valid update of an old account-state commitment in the ledger. The user then publishes a fresh commitment, and reveals a nullifier that prevents reuse of the old state. Thus, moving balances into accounts does not itself eliminate transaction-dependent state growth as these systems still publish serial numbers or nullifiers.

Pruning state and maintaining witnesses. A complementary line of work compresses or retires the state used to authenticate spends. Ecash coins can carry expiration dates so that expired spent-coin records may eventually be discarded [Sch98]. Short-lived issuance keys similarly bound redemption state for bearer tokens in Privacy Pass [DGS+18]. But both of these approaches conflict with our goal of allowing the user to be offline indefinitely.

A different approach is to periodically prune nullifiers but shift the burden of spending old notes to the users that want to spend them. Validators preserve historical roots but users must prove non-membership of the corresponding nullifier under every single root since the note was created [Pol23, Pal24].

[BM25] privately outsource proofs that notes remain unspent as the ledger advances using *oblivious synchronization*. In Bonsai, the receiver's saved nullifier state already records which receipts have been claimed and remains valid as unrelated transactions advance the ledger. An archive can supply a current MMR inclusion path when the receiver claims a receipt.

4 Preliminaries

Simulation-based security. In this work, we will sometimes prove security via simulation, following the standard real/ideal world paradigm [Gol04]. In this paradigm, a cryptographic scheme specifies an interactive protocol Π that takes place between some parties $P_1, \ldots, P_n$ initialized with inputs $x_1, \ldots, x_n$. The protocol Π is meant to emulate some ideal functionality $\mathcal{F}$ that takes an input $x_1, \ldots, x_n$ from each party and delivers an output $y_1, \ldots, y_n$ to each party.

The real execution. In the real execution, the protocol Π is executed in the presence of an adversary $\mathcal{A}$ that corrupts some subset $M \subset [n]$ of n parties. $\mathcal{A}$ takes as input the security parameter 1^λ, a set of input $\{x_i\}_{i \in M}$, and an auxiliary input z. The honest parties $[n] \setminus M$ follow the instructions of Π, while $\mathcal{A}$ sends messages on behalf of parties in M. If $\mathcal{A}$ is *malicious*, these messages may be computed following an arbitrary polynomial-time strategy, while if $\mathcal{A}$ is *semi-honest*, these messages must be computed following the instructions of Π. The interaction defines a random variable $\mathsf{REAL}_{\Pi,\mathcal{A}}[1^\lambda, \vec{x}, z, M]$ consisting of the output of $\mathcal{A}$, which may be an arbitrary function of its view, together with the outputs of the honest parties.

The ideal execution. In the ideal execution, a simulator Sim controlling some subset $M \subset [n]$ of n parties interacts with a trusted party $\mathcal{I}_{\mathcal{F}}$ implementing the functionality $\mathcal{F}$. Sim takes as input the security parameter 1^λ, a set of inputs $\{x_i\}_{i\in M}$, and an auxiliary input z. Each honest party $P_i \in [n] \setminus M$ sends their input x_i to $\mathcal{I}$, while Sim sends an input x'_i on behalf of each party $P_i \in M$. Let $x'_1, \ldots, x'_n$ be the entire set of inputs received by $\mathcal{I}$. Next, $\mathcal{I}$ computes $(y_1, \ldots, y_n) = \mathcal{F}(x'_1, \ldots, x'_n)$, delivers y_i to each honest party P_i, and delivers $\{y_i\}_{i\in M}$ to Sim. Finally, Sim outputs an arbitrary function of its view. The simulator's output together with the outputs of the honest parties defines the random variable $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim}}[1^\lambda, \vec{x}, z, M]$.

Definition 1. *An n-party protocol Π securely emulates an ideal functionality $\mathcal{F}$ in the presence of malicious (resp. semi-honest) adversaries corrupting a subset of parties $M \subset [n]$ if for any PPT malicious (resp. semi-honest) $\mathcal{A}$ corrupting parties M, there exists a PPT Sim such that for any set of inputs $\vec{x}$, and auxiliary input z,*

$$\mathsf{REAL}_{\Pi,\mathcal{A}}[1^\lambda, \vec{x}, z, M] \approx_c \mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim}}[1^\lambda, \vec{x}, z, M].$$

Digital Signatures. We will use a signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ that satisfies standard EUF-CMA security and additionally demand that it has a deterministic verification-key derivation algorithm $\mathsf{vk} = \mathsf{VKey}(\mathsf{sk})$.

Commitment schemes. A non-interactive commitment scheme $\mathsf{C} = (\mathsf{Setup}, \mathsf{Com})$ for message space $\mathcal{M}_\lambda$ consists of a setup algorithm $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and a deterministic algorithm $\mathsf{Com}_{\mathsf{pp}}(m; r)$. An opening (m, r) of c is valid if $c = \mathsf{Com}_{\mathsf{pp}}(m; r)$. We suppress pp when it is fixed.

Definition 2 (Computational hiding)**.** *For every stateful PPT adversary $\mathcal{A}$, consider the following game:*

- *Sample* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$.
- *Compute* $(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pp})$*, where* $m_0, m_1 \in \mathcal{M}_\lambda$.
- *Sample* $b \leftarrow\!\$\ \{0, 1\}$ *and* $r \leftarrow\!\$\ \mathcal{R}_\lambda^{\mathsf{com}}$*, and set* $c \leftarrow \mathsf{Com}_{\mathsf{pp}}(m_b; r)$.
- *Compute* $\widehat{b} \leftarrow \mathcal{A}(c)$.

The commitment scheme is computationally hiding if $\Pr[\widehat{b} = b] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$.

Definition 3 (Computational binding)**.** *For every PPT adversary $\mathcal{A}$, consider the following game:*

- *Sample* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$.
- *Compute* $(m, r, m', r') \leftarrow \mathcal{A}(\mathsf{pp})$.

The commitment scheme is computationally binding if

$$\Pr\left[m \neq m' \ \wedge\ \mathsf{Com}_{\mathsf{pp}}(m; r) = \mathsf{Com}_{\mathsf{pp}}(m'; r')\right] \leq \mathsf{negl}(\lambda).$$

Sparse Merkle trees. We use the *sparse Merkle trees* [DPP16] to authenticate a set $D \subseteq \{0,1\}^\ell$ of ℓ-bit keys. Conceptually it is a perfect binary Merkle tree of depth ℓ with one leaf per key, where the leaf of x holds 1 if $x \in D$ and 0 otherwise. Because the tree is sparse, the root of any subtree containing no member is a fixed *default* value that depends only on the subtree's height, so the tree can be represented and updated in time and space proportional to $|D| \cdot \ell$ rather than 2^ℓ. The empty tree has a canonical state and root $\mathsf{root}_\emptyset$ that anyone can recompute, and the tree supports two operations:

- $\mathsf{SMT.Insert}(\mathsf{st}, x) \to (\mathsf{st}', \mathsf{root}', \pi)$: set the leaf of x to 1 and output the updated tree, its new root, and a proof π consisting of the ℓ sibling digests on the path of x.

- $\mathsf{SMT.VerifyInsert}(\mathsf{root}, x, \pi) \to \{\mathsf{root}', \bot\}$: recompute the root along the path of x with leaf value 0 and check that it equals root (non-membership of x), then recompute it with leaf value 1 and output the result root'.

Both recomputations use the same ℓ siblings, so verification costs 2ℓ hash evaluations. When H is a collision resistant hash function and root commits to a set D, it is computationally infeasible for an adversary to produce (x, π) for which verification outputs a root if $x \in D$, or outputs a root that does not commit to $D \cup \{x\}$ [DPP16].

Merkle Mountain Ranges If we are only interested in membership proofs, it is possible to create a merkle accumulator where insertion does not require us to maintain the entire tree – just a logarithmic number of roots. We use a *Merkle Mountain Range* [Tod12] where appended elements are hashed into the largest possible perfectly balanced Merkle trees. Of course, producing an opening proof for an arbitrary element still requires the entire tree. The empty range has a canonical state st_0 and root root_0 that anyone can recompute.

- $\mathsf{MMR.Insert}(\mathsf{st}, x) \to (\mathsf{st}', \mathsf{root}')$: append one element and return the updated state and root.
- $\mathsf{MMR.Prove}(\vec{x}, \mathsf{pos}) \to \pi$: compute an opening proof for the element at position pos of the list $\vec{x}$ appended so far.
- $\mathsf{MMR.Verify}(\mathsf{root}, x, \mathsf{pos}, \pi) \to \{0, 1\}$: verify that x is the element at position pos.

As in a standard merkle tree, an opening proof is the path running from the element through its mountain's peak and the bagged peaks to the root, and the position determines the left–right orientation of every step of the path, so a proof binds the pair (x, pos). When H is a collision resistant hash function, it is computationally infeasible for an adversary to produce a list $\vec{x}$, a position pos, a value $x^\star$ that is not the element of $\vec{x}$ at pos, and a proof $\pi^\star$ with $\mathsf{MMR.Verify}(\mathsf{root}, x^\star, \mathsf{pos}, \pi^\star) = 1$ where root is obtained by inserting the elements of $\vec{x}$ in order.

Non-interactive zero-knowledge proofs. Let $\mathcal{L}$ be an NP language and $\mathcal{R}$ its corresponding NP relation. A simulation-extractable NIZK for $\mathcal{R}$ consists of the following algorithms:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{crs}, \mathsf{td})$: takes a security parameter and outputs a common reference string crs and trapdoor $\mathsf{td} = (\mathsf{td}_{\mathsf{sim}}, \mathsf{td}_{\mathsf{ext}})$.
- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$: takes $(x, w) \in \mathcal{R}$ and outputs a proof π for $x \in \mathcal{L}$.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi) \to b$: outputs a bit indicating whether verification succeeds.
- $\mathsf{SimProve}(\mathsf{crs}, \mathsf{td}_{\mathsf{sim}}, x) \to \pi$: outputs a simulated proof for an arbitrary statement x.

We drop the crs argument wherever it is implicit. When specifying a relation, we write $\{w \mid \varphi_1 \wedge \cdots \wedge \varphi_m\}$, where w is the witness and each φ_i is a constraint. At times a constraint may contain an otherwise unconstrained variable. This denotes a tag-based NIZK in which that variable is part of the statement.

Definition 4 (Straight-line simulation-extractable NIZK)**.** *A simulation-extractable NIZK satisfies the following properties.*

- Perfect completeness. *For every* $(x, w) \in \mathcal{R}$*,*

$$\Pr\left[\begin{array}{c}(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)\end{array} : \mathsf{Verify}(x, \pi) = 1\right] = 1.$$

- Proof of knowledge. *For every PPT adversary* $\mathcal{A}$*, there is a PPT extractor* Ext *such that*

$$\Pr\left[\begin{array}{c}(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ w \leftarrow \mathsf{Ext}(\mathsf{crs}, \mathsf{td}_{\mathsf{ext}}, x, \pi; \mathsf{aux})\end{array} : \begin{array}{c}\mathsf{Verify}(x, \pi) = 1 \\ (x, w) \notin \mathcal{R}\end{array}\right] \leq \mathsf{negl}(\lambda).$$

- Simulated-proof correctness. *For every statement x,*

$$\Pr\left[\begin{array}{c}(\mathsf{crs},\mathsf{td})\leftarrow\mathsf{Setup}(1^\lambda)\\ \pi\leftarrow\mathsf{SimProve}(\mathsf{crs},\mathsf{td}_{\mathsf{sim}},x)\end{array}:\mathsf{Verify}(x,\pi)=1\right]=1.$$

- Computational zero knowledge. *There exists an algorithm* SimProve *such that, for every PPT adversary $\mathcal{A}$, define*

$$p_{\mathsf{real}}=\Pr\left[\begin{array}{ccc}(\mathsf{crs},\mathsf{td})\leftarrow\mathsf{Setup}(1^\lambda) & & (x,w)\in\mathcal{R}\\ (x,w)\leftarrow\mathcal{A}(\mathsf{crs}) & : & \mathcal{A}(\pi)=1\\ \pi\leftarrow\mathsf{Prove}(\mathsf{crs},x,w) & &\end{array}\right],$$

$$p_{\mathsf{sim}}=\Pr\left[\begin{array}{ccc}(\mathsf{crs},\mathsf{td})\leftarrow\mathsf{Setup}(1^\lambda) & & (x,w)\in\mathcal{R}\\ (x,w)\leftarrow\mathcal{A}(\mathsf{crs}) & : & \mathcal{A}(\pi)=1\\ \pi\leftarrow\mathsf{SimProve}(\mathsf{crs},\mathsf{td}_{\mathsf{sim}},x) & &\end{array}\right].$$

  Computational zero knowledge requires

$$|p_{\mathsf{real}}-p_{\mathsf{sim}}|\le\mathsf{negl}(\lambda).$$

  We require the same indistinguishability for polynomially many valid statement–witness pairs chosen adaptively under one crs*.*

- Simulation extractability. *For every PPT adversary $\mathcal{A}$, there exists a PPT extractor* Ext*. On any statement x, the simulation oracle computes*

$$\pi\leftarrow\mathsf{SimProve}(\mathsf{crs},\mathsf{td}_{\mathsf{sim}},x),$$

  adds (x,π) to Q, and returns π. Then

$$\Pr\left[\begin{array}{ccc}(\mathsf{crs},\mathsf{td})\leftarrow\mathsf{Setup}(1^\lambda),\quad Q\leftarrow\emptyset & & \mathsf{Verify}(x,\pi)=1\\ (x,\pi)\leftarrow\mathcal{A}^{\mathcal{O}_{\mathsf{sim}}}(\mathsf{crs}) & : & (x,w)\notin\mathcal{R}\\ w\leftarrow\mathsf{Ext}(\mathsf{crs},\mathsf{td}_{\mathsf{ext}},x,\pi;\mathsf{aux}) & & (x,\pi)\notin Q\end{array}\right]\le\mathsf{negl}(\lambda).$$

The extractor is straight line: on input a valid proof and aux*, it extracts a witness with overwhelming probability without rewinding the prover. In the standard CRS model,* aux *is empty. In the random-oracle model, it may contain the adversary's random-oracle queries. The* crs *may then be empty.*

Definition 5 (Square R1CS)**.** *The indexed Square-R1CS relation is the set of all triples*

$$(\mathtt{i},\mathtt{x},\mathtt{w})=((\mathbb{F},n,k,m,A,B),x,w)$$

where $\mathbb{F}$ is a finite field, n, k, and m are natural numbers $x\in\mathbb{F}^n$ and $w\in\mathbb{F}^{k-n}$ are vectors over $\mathbb{F}$ and A,B are $m\times k$ matrices over $\mathbb{F}$ such that

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

where $z=(x,w)$ and $\circ$ denotes the hadamard product.

5 Definitions

We formalize the security guarantees of our protocol using the simulation-based paradigm [Gol04] by an ideal functionality $\mathcal{F}_{\mathsf{PrivPay}}[\mathcal{L}]$ that models an account-based payment service. The ideal functionality is

parametrized by a leakage function $\mathcal{L}$ which models the privacy guarantees from different instantiations of private payments. The ideal payment service provides the following interfaces:

$$\mathcal{F}_{\mathsf{PrivPay}} = (\mathsf{Create}, \mathsf{Register}, \mathsf{Send}, \mathsf{Receive}).$$

It additionally accepts the adversarial scheduling command Release defined below.

Let λ be the security parameter, $\mathcal{P}$ be the set of parties, P_{L} be the distinguished party that operates the ledger, and $\mathcal{N}_\lambda$ be the space of account names (pseudonyms), which is the verification-key space of Sig. Membership in $\mathcal{N}_\lambda$ is efficiently decidable and $\bot \notin \mathcal{N}_\lambda$. Fix the balance domain $\mathcal{B}$ with $0 \in \mathcal{B}$, a finite set of account names $\mathsf{Names} \subset \mathcal{N}_\lambda$, an initial owner map Owner with domain Names, and a public initial allocation $\mathsf{Init} : \mathcal{N}_\lambda \to \mathcal{B}$ satisfying $\mathsf{Init}[A] = 0$ for every $A \notin \mathsf{Names}$. Each initial account name is a valid verification key whose owner holds a corresponding signing key. The ledger party P_{L} is always assumed to be honest in our constructions. Apart from these finitely many account names, ownership is established only through the private Create interface.

A party samples $(\mathsf{sk}_A, A) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$, retains sk_A, and may privately share A with other parties to receive payments before registering with the ideal functionality. The ideal functionality verifies (sk_A, A) is a valid key pair and if so assigns that party to be the owner of A. Users publicly register an account only when they wish to send or receive through it. A sender with registered account S can send an amount v to a receiver's account R as follows:

- The sender sends (Send, S, R, v) to the ideal functionality. If S has sufficient balance, the functionality debits v and records the payment at position pid. An honest sender's payment is 'pending' until the receiver R accepts it, and the owner of R is privately notified of (pid, S, R, v).
- If the sender is corrupted the payment is assigned a pid and recorded in the log but under a 'withheld' status until the corrupt party 'releases' it by sending $\mathsf{Release}(\mathsf{pid})$ to the ideal functionality, at which point the owner of R is notified. While a transaction is withheld the recipient cannot claim the funds even if they are aware of the incoming transaction.
- The receiver sends $(\mathsf{Receive}, R, \mathsf{pid})$ to the ideal functionality. If pid is a pending payment intended for R, the functionality credits v to R.

If R was never created, it has no owner and the payment cannot be claimed.

Leakage functions. To appropriately model the privacy leakage in different constructions we parametrize the ideal functionality with leakage functions. Account creation is private and emits no leakage event. For an accepted payment operation, let A denote its acting account. The complete events are

$$(\mathsf{Register}, A), \qquad (\mathsf{Send}, S, R, v), \qquad (\mathsf{Receive}, S, R, v).$$

Registration always leaks $(\mathsf{Register}, A)$. Our constructions target the following two leakage functions.

- **Sender-Receiver Unlinkability.** Reveals the operation type and acting account but not the counterparty.
$$\mathcal{L}_{\mathsf{unl}}(\mathsf{Send}, S, R, v) = (\mathsf{Send}, S),$$
$$\mathcal{L}_{\mathsf{unl}}(\mathsf{Receive}, S, R, v) = (\mathsf{Receive}, R).$$
- **Send-Receive Indistinguishability.** Reveals nothing except the acting account.
$$\mathcal{L}_{\mathsf{ind}}(\mathsf{Send}, S, R, v) = (S),$$
$$\mathcal{L}_{\mathsf{ind}}(\mathsf{Receive}, S, R, v) = (R).$$

Ideal functionality $\mathcal{F}_{\mathsf{PrivPay}}[\mathcal{L}]$

The Owner map initially contains exactly the initial Names ownership entries. Initialize the set of registered account names $\mathsf{Reg} \leftarrow \emptyset$, and their corresponding balances $\mathsf{Bal} \leftarrow \emptyset$. Maintain a transaction log of the form

$$\mathsf{Log}[\mathsf{pid}] = (S, R, v, \mathsf{status}), \qquad \mathsf{status} \in \{\mathsf{withheld}, \mathsf{pending}, \mathsf{completed}\}.$$

where positions pid are assigned in increasing order. If $\mathcal{L} = \mathcal{L}_{\mathsf{ind}}$, the position counter is also incremented on every accepted Receive. If any *require* check below fails, the functionality outputs $\perp$ to the caller, leaves its state unchanged, and emits no leakage.

Create. On input $(\mathsf{Create}, \mathsf{sk}_A, A)$ from a party P,

- require $A \in \mathcal{N}_\lambda$, $A = \mathsf{VKey}(\mathsf{sk}_A)$, and $A \notin \mathrm{dom}(\mathsf{Owner})$
- $\mathsf{Owner}[A] \leftarrow P$
- $\mathsf{Names} \leftarrow \mathsf{Names} \cup \{A\}$
- output A to P

Register. On input $(\mathsf{Register}, A)$ from a party P,

- require $\mathsf{Owner}[A] = P$ and $A \notin \mathsf{Reg}$
- $\mathsf{Reg} \leftarrow \mathsf{Reg} \cup \{A\}$
- $\mathsf{Bal}[A] \leftarrow \mathsf{Init}[A]$
- publicly leak $(\mathsf{Register}, A)$

Send. On input (Send, S, R, v) from a party P,

- require $\mathsf{Owner}[S] = P$, $S \in \mathsf{Reg}$, $R \in \mathcal{N}_\lambda$, $0 \le v \le \mathsf{Bal}[S]$, and $\mathsf{Bal}[S], v, \mathsf{Bal}[S] - v \in \mathcal{B}$
- $\mathsf{Names} \leftarrow \mathsf{Names} \cup \{R\}$
- let pid be the next position in the receipt-log and set

  $$\mathsf{Bal}[S] \leftarrow \mathsf{Bal}[S] - v, \qquad \mathsf{Log}[\mathsf{pid}] \leftarrow (S, R, v, \mathsf{status}).$$

  where $\mathsf{status} = \mathsf{pending}$ if the sender is honest and $\mathsf{status} = \mathsf{withheld}$ otherwise
- output $(\mathsf{Sent}, \mathsf{pid}, R, v)$ to the sender
- if $\mathsf{status} = \mathsf{pending}$ and $R \in \mathrm{dom}(\mathsf{Owner})$, output (pid, S, R, v) to $\mathsf{Owner}[R]$
- broadcast $\mathcal{L}(\mathsf{Send}, S, R, v)$

Release. On input $(\mathsf{Release}, \mathsf{pid})$ from a corrupted party, require that $\mathsf{Log}[\mathsf{pid}] = (S, R, v, \mathsf{withheld})$, set its status to pending, and output (pid, S, R, v) to $\mathsf{Owner}[R]$ if $R \in \mathrm{dom}(\mathsf{Owner})$.

Receive. On input $(\mathsf{Receive}, R, \mathsf{pid})$ from a party P parse $\mathsf{Log}[\mathsf{pid}] = (S, R', v, \mathsf{status})$,

- require $\mathsf{Owner}[R] = P$, $R \in \mathsf{Reg}$, $R = R'$, $\mathsf{status} = \mathsf{pending}$, and $\mathsf{Bal}[R] + v \in \mathcal{B}$.
- $\mathsf{Bal}[R] \leftarrow \mathsf{Bal}[R] + v$
- $\mathsf{Log}[\mathsf{pid}].\mathsf{status} \leftarrow \mathsf{completed}$
- output $(\mathsf{Received}, \mathsf{pid}, S, v)$ to the receiver and broadcast $\mathcal{L}(\mathsf{Receive}, S, R, v)$

Figure 1: The ideal private-payment functionality.

6 Construction

Overview. In Bonsai, an account name is a signature verification key. To create an account, its owner locally samples $(\mathsf{sk}_A, A) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$, and may share A over a private channel. Registration proves knowledge of sk_A corresponding to A. After registration, knowledge of the opening of the current account commitment authorizes each state transition. In real world deployments, it may be desirable to authenticate transfers with both the signing key (stored securely via threshold/hardware) and the opening of the account commitment (which could be offloaded to a trusted server for proof generation). Thus if the commitment opening is compromised the account does not lose funds – only its privacy. The public state of account A is the single commitment

$$\mathsf{com}_A = \mathsf{Com}_{\mathsf{acct}}(b_A, \mathsf{root}_{\mathsf{null}}(A); r_A)$$

where b_A is the balance of the account, $\mathsf{root}_{\mathsf{null}}(A)$ is the root of the account's local nullifier tree, a sparse Merkle tree (Section 4) keyed by receipt position which contains the nullifiers for all transactions received by A, and r_A is the randomness in the commitment.

Send. A send transaction creates a hiding commitment

$$\rho = \mathsf{Com}_{\mathsf{rec}}(v, \mathsf{Sen}, \mathsf{Rec}; r'')$$

on chain which binds the amount being sent v, Sender Sen, and Receiver Rec. Alongside ρ, the sender publishes an updated account commitment $\mathsf{com}'_{\mathsf{Sen}}$ and a zero knowledge proof π which shows that:

- the old commitment $\mathsf{com}_{\mathsf{Sen}}$ opens to $(b, \mathsf{root}_{\mathsf{null}}(\mathsf{Sen}))$
- the new commitment $\mathsf{com}'_{\mathsf{Sen}}$ opens to $(b - v, \mathsf{root}_{\mathsf{null}}(\mathsf{Sen}))$ and $v, b - v$ are both in the balance range
- ρ opens to $(v, \mathsf{Sen}, \mathsf{Rec})$

State. Validators maintain a merkle mountain range of commitments and insert each accepted receipt into the MMR, and record the updated root root_ρ in a history $\mathcal{T}$ of recent roots. Importantly, validators only store the MMR frontier (a logarithmic number of roots), which suffices to compute the current root and to process every future insertion, together with the bounded history $\mathcal{T}$. Once the send is accepted, the sender forwards the opening of ρ, together with its payment identifier pid, to the receiver over a private channel. The identifier pid is the position that ρ occupies in the receipt log, and it serves directly as the transaction's nullifier:

$$\mathsf{null} = \mathsf{pid}.$$

Positions are unique, so distinct receipts always carry distinct nullifiers and no send can block another pending payment. As a by-product, we have additional privacy against the sender who does not learn if/when a send was claimed by the receiver.

Receive. To finish the transaction, Rec sends an updated commitment $\mathsf{com}'_{\mathsf{Rec}}$, a root_ρ — the latest receipt root recorded by the ledger — and a zero knowledge proof π which shows that:

- Rec knows an opening proof π_{MMR} for a receipt ρ at position pid under root_ρ
- ρ opens to $(v, \mathsf{Sen}, \mathsf{Rec})$
- the old commitment $\mathsf{com}_{\mathsf{Rec}}$ opens to $(b, \mathsf{root}_{\mathsf{null}})$
- the new commitment $\mathsf{com}'_{\mathsf{Rec}}$ opens to $(b + v, \mathsf{root}'_{\mathsf{null}})$
- $\mathsf{root}'_{\mathsf{null}}$ results from inserting $\mathsf{null} = \mathsf{pid}$ into $\mathsf{root}_{\mathsf{null}}$

The ledger verifies π, checks that the anchor appears in $\mathcal{T}$, and updates the receiver's account commitment to $\mathsf{com}'_{\mathsf{Rec}}$. Rather than proving knowledge of a validator signature on root_ρ inside the circuit, the receiver reveals the anchor in the clear and the ledger checks it against its own root history, following deployed designs [HBHW26, Pen24, PSS19].

Pruning user state. As described so far, a user's nullifier tree holds one leaf for every payment it has ever received, so user state grows with the user's lifetime activity. We now describe a simple strategy for users to prune their state.

Because nullifiers are receipt positions, they arrive in increasing order: a receipt created later has a larger position. For any threshold L, the sparse Merkle tree over positions splits into a prefix $[0, L)$, whose contents are fixed if we never insert below L again, and a suffix $[L, \infty)$. The prefix is summarized by its *frontier* F_L which is at most ℓ hashes and a user who stores the frontier together with the set $S_{\geq L}$ of positions it has claimed at or above L can produce π_{SMT} for any position $\mathsf{pid} \geq L$. The nullifiers below L can be moved to cold storage, from which the full tree, and hence a path for any position, can be recomputed if ever needed.

The threshold is chosen by the user and does not affect anything else in the protocol. So a wallet may set it as aggressively as it chooses and update it over time. Maintaining the w most recently claimed positions requires a hot state of $\ell + w$ hashes and supports claiming any receipt at a position above the w-th most recent claim.

We provide a formal description of the protocol in Fig. 4 and describe the relations proved using the NIZK in each phase below.

Registration relation $\mathcal{R}_{\mathsf{reg}}$. This relation shows that a new commitment for account A was initialized with initial balance $\mathsf{Init}[A]$, that the prover knows the signing key corresponding to A, and that the commitment contains the root $\mathsf{root}_\emptyset$ of the empty nullifier tree.

- *Statement:* $x_{\mathsf{reg}} = (A, \mathsf{Init}[A], \mathsf{com}_A)$.
- *Witness:* $w_{\mathsf{reg}} = (\mathsf{sk}_A, r_A)$.
- *Relation:* accept if all of the following constraints hold:
  - $\mathsf{Init}[A] \in \mathcal{B}$.
  - $A = \mathsf{VKey}(\mathsf{sk}_A)$.
  - $\mathsf{com}_A = \mathsf{Com}_{\mathsf{acct}}(\mathsf{Init}[A], \mathsf{root}_\emptyset; r_A)$.

Send relation $\mathcal{R}_{\mathsf{send}}$. This relation shows that the acting account debits a nonnegative amount from a valid committed balance and binds that amount, the acting sender, and a receiver identifier into the receipt. The receiver need not have registered when the receipt is created, but it must do so before it can receive the transfer.

- *Statement:* $x_{\mathsf{s}} = (\mathsf{Sen}, \mathsf{com}, \mathsf{com}', \rho)$.
- *Witness:* $w_{\mathsf{s}} = (b, \mathsf{root}_{\mathsf{null}}, r, r', r'', v, \mathsf{Rec})$.
- *Relation:* accept if all of the following constraints hold:
  - $\mathsf{com} = \mathsf{Com}_{\mathsf{acct}}(b, \mathsf{root}_{\mathsf{null}}; r)$.
  - $\mathsf{com}' = \mathsf{Com}_{\mathsf{acct}}(b - v, \mathsf{root}_{\mathsf{null}}; r')$.
  - $\rho = \mathsf{Com}_{\mathsf{rec}}(v, \mathsf{Sen}, \mathsf{Rec}; r'')$.
  - $\mathsf{Rec} \in \mathcal{N}_\lambda$.
  - $0 \leq v \leq b$ and $b, v, b - v \in \mathcal{B}$.

Receive relation $\mathcal{R}_{\mathsf{recv}}$. This relation shows possession of a receipt addressed to the acting account, included in the receipt MMR under a revealed anchor, and credits exactly the committed receipt amount. It also verifies the insertion of the receipt's position, which serves as its nullifier, into the account's committed nullifier tree.

- *Statement:* $x_{\mathsf{r}} = (\mathsf{Rec}, \mathsf{com}, \mathsf{com}', \mathsf{root}_\rho)$.
- *Witness:* $w_{\mathsf{r}} = (b, \mathsf{root}_{\mathsf{null}}, \mathsf{root}'_{\mathsf{null}}, r, r', \rho, \mathsf{pid}, \pi_{\mathsf{MMR}}, \pi_{\mathsf{SMT}}, v, \mathsf{Sen}, r'')$.
- *Relation:* accept if all of the following constraints hold:
  - $\mathsf{com} = \mathsf{Com}_{\mathsf{acct}}(b, \mathsf{root}_{\mathsf{null}}; r)$.
  - $\mathsf{com}' = \mathsf{Com}_{\mathsf{acct}}(b + v, \mathsf{root}'_{\mathsf{null}}; r')$.
  - $\rho = \mathsf{Com}_{\mathsf{rec}}(v, \mathsf{Sen}, \mathsf{Rec}; r'')$.
  - $\mathsf{MMR.Verify}(\mathsf{root}_\rho, \rho, \mathsf{pid}, \pi_{\mathsf{MMR}}) = 1$.
  - $\mathsf{SMT.VerifyInsert}(\mathsf{root}_{\mathsf{null}}, \mathsf{pid}, \pi_{\mathsf{SMT}}) = \mathsf{root}'_{\mathsf{null}}$.
  - $v \geq 0$ and $b, v, b + v \in \mathcal{B}$.

6.1 Hiding the operation type

The scheme above reveals whether a particular operation is a send or a receive. Although this is just one additional bit of (seemingly benign) leakage, we show below that it can actually have a massive impact on deanonymizing payments. Suppose there are four parties A, B, C, D that wish to carry out the following sequence of payments:

$$[(A \to B; 5),\ (B \to A; 10),\ (C \to D; 7)] \tag{1}$$

There are many ways to interact with our ideal functionality (Fig. 1) in order to realize this but consider the following order of operations

$$[\mathsf{Send}(A, B, 5),\ \mathsf{Receive}(B, \mathsf{pid}_1),\ \mathsf{Send}(B, A, 10),\ \mathsf{Receive}(A, \mathsf{pid}_2),\ \mathsf{Send}(C, D, 7),\ \mathsf{Receive}(D, \mathsf{pid}_3)]$$

where pid_i denotes the payment identifier of the i-th send in Eq. (1) and we assume all parties have sufficient balance that there are no other parties/transactions in the log. Depending on the leakage function, the observer sees:

$\mathcal{L}_{\mathsf{unl}}$: $(\mathsf{send}, A), (\mathsf{recv}, B), (\mathsf{send}, B), (\mathsf{recv}, A), (\mathsf{send}, C), (\mathsf{recv}, D)$

$\mathcal{L}_{\mathsf{ind}}$: A, B, B, A, C, D

With $\mathcal{L}_{\mathsf{unl}}$, at any point, there is at most one pending receive, which implies that each receive must claim the immediately preceding send and therefore there is exactly one possible sequence of operations that is possible, described in Fig. 2.

With $\mathcal{L}_{\mathsf{ind}}$, the observer sees only A, B, B, A, C, D. The first operation must be a send, but each subsequent operation could be another send or a receive claiming an earlier, unclaimed receipt. Self-payments are also possible, since $\mathcal{R}_{\mathsf{send}}$ does not require the sender and receiver to differ. After each of the six operations, there are respectively $1, 2, 4, 10, 26, 76$ ways to assign operation types and match receives to earlier sends, consistent with this leakage. In particular, all six operations could be sends whose recipients are not revealed by the log. Thus, hiding the operation type conceals not only the payment links but also how many payments even took place.

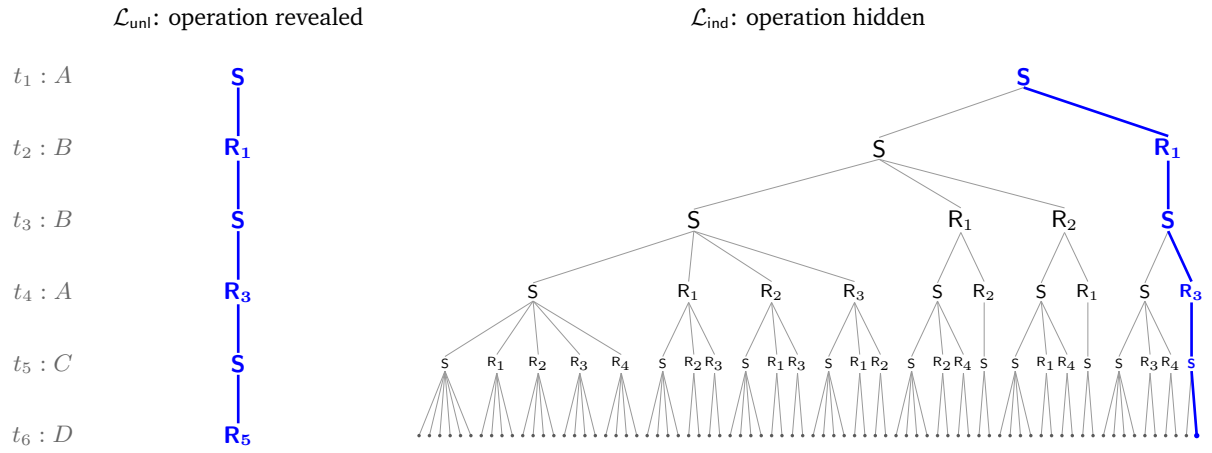

Figure 2: Operation choices for the transcript above, with the acting account shown at each level. The highlighted path is $\mathsf{S}, \mathsf{R}_1, \mathsf{S}, \mathsf{R}_3, \mathsf{S}, \mathsf{R}_5$, where R_i claims the receipt published in transaction t_i. $\mathcal{L}_{\mathsf{unl}}$ determines a unique path, whereas $\mathcal{L}_{\mathsf{ind}}$ admits 76 possibilities with self-payments allowed.

An easy way to incorporate operation hiding is to act like you're doing both operations by publishing the common record

$$\mathsf{tx}_{\mathsf{op}} = (A, \mathsf{com}', \rho, \mathsf{root}_\rho, \pi)$$

where the proof π shows that either the send or receive relation is satisfied. Together with the current account state, this determines

$$x = (A, \mathsf{com}, \mathsf{com}', \rho, \mathsf{root}_\rho).$$

A send sets ρ to be the valid receipt that a receiver can claim but leaves the root of the nullifier tree inside the account commitment unchanged. Conversely, a receive sets ρ to be a fixed-format dummy receipt, but updates the nullifier root by inserting the position of the consumed receipt. In both branches, the ledger checks $\mathsf{root}_\rho \in \mathcal{T}$, updates com to com′, and inserts ρ into the receipt MMR. To enforce that dummy receipts are unspendable, $\mathsf{Com}_{\mathsf{rec}}$ gains a final type slot: a send commits its real receipt with type 1, while a receive fixes the published dummy to type 0 and is only allowed to consume receipts of type 1.

Operation-hiding relation $\mathcal{R}_{\mathsf{op}}$.

- *Statement:* $x = (A, \mathsf{com}, \mathsf{com}', \rho, \mathsf{root}_\rho)$.
- *Witness:* $w = (\mathsf{op}, w_{\mathsf{op}})$, where $\mathsf{op} \in \{\mathsf{send}, \mathsf{receive}\}$ and

$$w_{\mathsf{send}} = (b, \mathsf{root}_{\mathsf{null}}, r, r', r'', v, \mathsf{Rec}),$$
$$w_{\mathsf{receive}} = (b, \mathsf{root}_{\mathsf{null}}, \mathsf{root}'_{\mathsf{null}}, r, r', r''', \rho_{\mathsf{in}}, \mathsf{pid}, \pi_{\mathsf{MMR}}, \pi_{\mathsf{SMT}}, v, \mathsf{Sen}, r'').$$

- *Relation:*
  - If $\mathsf{op} = \mathsf{send}$, set $\mathsf{Sen} = A$ and accept if all of the following constraints hold:
    - $\mathsf{com} = \mathsf{Com}_{\mathsf{acct}}(b, \mathsf{root}_{\mathsf{null}}; r)$.
    - $\mathsf{com}' = \mathsf{Com}_{\mathsf{acct}}(b - v, \mathsf{root}_{\mathsf{null}}; r')$.
    - $\rho = \mathsf{Com}_{\mathsf{rec}}(v, \mathsf{Sen}, \mathsf{Rec}, 1; r'')$.
    - $\mathsf{Rec} \in \mathcal{N}_\lambda$.
    - $0 \le v \le b$ and $b, v, b - v \in \mathcal{B}$.
  - Else if $\mathsf{op} = \mathsf{receive}$, parse $A = \mathsf{Rec}$ and accept if all of the following constraints hold:

* $\mathsf{com} = \mathsf{Com}_{\mathsf{acct}}(b, \mathsf{root}_{\mathsf{null}}; r)$.
* $\mathsf{com}' = \mathsf{Com}_{\mathsf{acct}}(b + v, \mathsf{root}'_{\mathsf{null}}; r')$.
* $\rho_{\mathsf{in}} = \mathsf{Com}_{\mathsf{rec}}(v, \mathsf{Sen}, \mathsf{Rec}, 1; r'')$.
* $\rho = \mathsf{Com}_{\mathsf{rec}}(0, \bot, \bot, 0; r''')$.
* $\mathsf{MMR.Verify}(\mathsf{root}_\rho, \rho_{\mathsf{in}}, \mathsf{pid}, \pi_{\mathsf{MMR}}) = 1$.
* $\mathsf{SMT.VerifyInsert}(\mathsf{root}_{\mathsf{null}}, \mathsf{pid}, \pi_{\mathsf{SMT}}) = \mathsf{root}'_{\mathsf{null}}$.
* $v \geq 0$ and $b, v, b + v \in \mathcal{B}$.

– Else, reject.

The branch selector op is part of the witness, so a proof for $\mathcal{R}_{\mathsf{op}}$ proves that one valid transition occurred without revealing which branch was taken.

Π_{Bonsai}: Shared operations

$\mathsf{Setup}(1^\lambda)$:
- ledger initializes $\mathsf{Acct} \leftarrow \emptyset$, sets $(\mathsf{st}_\rho, \mathsf{root}_\rho)$ to the canonical state and root of the empty MMR, and initializes the root history $\mathcal{T} \leftarrow \{\mathsf{root}_\rho\}$, which retains the W most recently recorded roots.
- generate and publish the public parameters for commitments and CRS for NIZK relations (if any).

$\mathsf{Create}(1^\lambda)$: locally sample $(\mathsf{sk}_A, A) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$

$\mathsf{Register}(A)$:
- party:
  – initialize $\mathsf{st}_A^{\mathsf{null}}$ to the canonical empty tree, sample fresh r_A, and compute $\mathsf{com}_A \leftarrow \mathsf{Com}_{\mathsf{acct}}(\mathsf{Init}[A], \mathsf{root}_\emptyset; r_A)$
  – prove $\pi_{\mathsf{reg}} \leftarrow \Pi.\mathsf{Prove}(\mathcal{R}_{\mathsf{reg}}, (A, \mathsf{Init}[A], \mathsf{com}_A), (\mathsf{sk}_A, r_A))$
  – submit $(A, \mathsf{com}_A, \pi_{\mathsf{reg}})$
- ledger:
  – verify that $A \in \mathcal{N}_\lambda$, $A \notin \mathrm{dom}(\mathsf{Acct})$, and $\Pi.\mathsf{Verify}(\mathcal{R}_{\mathsf{reg}}, (A, \mathsf{Init}[A], \mathsf{com}_A), \pi_{\mathsf{reg}}) = 1$
  – if all checks pass, set $\mathsf{Acct}[A] \leftarrow \mathsf{com}_A$. else return $\bot$

Figure 3: Setup, account creation, and registration shared by both variants of Bonsai.

Remark 1 (Public Fees)**.** *The above construction can be modified to support fee payment in a straightforward manner. In the send/receive operations, users simply reveal the amount v_{fee} which becomes a part of the statement in $\mathcal{R}_{\mathsf{send}}, \mathcal{R}_{\mathsf{recv}}$, and $\mathcal{R}_{\mathsf{op}}$ which additionally prove that the new commitment's balance is reduced by v_{fee}.*

Theorem 1 (Security of Π_{Bonsai})**.** *Suppose that:*

1. Sig *is a correct EUF-CMA-secure signature scheme.*
2. $\mathsf{Com}_{\mathsf{acct}}$ *and* $\mathsf{Com}_{\mathsf{rec}}$ *are secure commitment schemes (Definitions 2 and 3).*
3. Π *is a secure NIZK for the $\mathcal{R}_{\mathsf{reg}}$, $\mathcal{R}_{\mathsf{send}}$, and $\mathcal{R}_{\mathsf{recv}}$ relations (Definition 4).*
4. H *is a collision-resistant hash function.*

Then Π_{Bonsai} securely realizes $\mathcal{F}_{\mathsf{PrivPay}}[\mathcal{L}_{\mathsf{unl}}]$ against static malicious corruption of any set of users.

Proof. For a PPT adversary $\mathcal{A}$ corrupting an arbitrary set of users, we construct a PPT simulator Sim that interacts with $\mathcal{F}_{\mathsf{PrivPay}}[\mathcal{L}_{\mathsf{unl}}]$. It faithfully generates the commitment parameters and runs each relation-specific NIZK setup with fresh randomness while retaining the simulation and extraction trapdoors. During the protocol, it maintains a map Acct, which holds one account commitment per account, and the root history

$\Pi_{\mathsf{Bonsai}}[\mathcal{L}_{\mathsf{unl}}]$

$\mathsf{Send}(\mathsf{Sen}, \mathsf{Rec}, v)$:

- sender parses $\mathsf{com} = \mathsf{Acct}[\mathsf{Sen}]$ with opening $\mathsf{com} = \mathsf{Com}_{\mathsf{acct}}(b, \mathsf{root}_{\mathsf{null}}; r)$,
  - sample fresh r', r'' and compute account commitment $\mathsf{com}' \leftarrow \mathsf{Com}_{\mathsf{acct}}(b - v, \mathsf{root}_{\mathsf{null}}; r')$
  - compute on chain receipt $\rho \leftarrow \mathsf{Com}_{\mathsf{rec}}(v, \mathsf{Sen}, \mathsf{Rec}; r'')$
  - form the witness $w_{\mathsf{s}} = (b, \mathsf{root}_{\mathsf{null}}, r, r', r'', v, \mathsf{Rec})$
  - prove $\pi \leftarrow \Pi.\mathsf{Prove}(\mathcal{R}_{\mathsf{send}}, (\mathsf{Sen}, \mathsf{com}, \mathsf{com}', \rho), w_{\mathsf{s}})$
  - submit $(\mathsf{Sen}, \mathsf{com}', \rho, \pi)$.
- ledger:
  - form the statement $x_{\mathsf{s}} = (\mathsf{Sen}, \mathsf{com}, \mathsf{com}', \rho)$ using $\mathsf{com} = \mathsf{Acct}[\mathsf{Sen}]$
  - verify that $\Pi.\mathsf{Verify}(\mathcal{R}_{\mathsf{send}}, x_{\mathsf{s}}, \pi) = 1$
  - if all checks pass, set $\mathsf{Acct}[\mathsf{Sen}] \leftarrow \mathsf{com}'$. else return $\perp$
  - update the state root $(\mathsf{st}_\rho, \mathsf{root}_\rho) \leftarrow \mathsf{MMR.Insert}(\mathsf{st}_\rho, \rho)$ and record root_ρ in the root history $\mathcal{T}$
- sender reads the position pid at which the ledger inserted ρ, outputs $(\mathsf{Sent}, \mathsf{pid}, \mathsf{Rec}, v)$, and forwards $\omega = (\rho, \mathsf{pid}, v, \mathsf{Sen}, \mathsf{Rec}, r'')$ to the receiver

$\mathsf{Receive}(\mathsf{Rec}, \mathsf{pid})$:

- receiver parses $\omega = (\rho, \mathsf{pid}, v, \mathsf{Sen}, \mathsf{Rec}, r'')$ and account opening $\mathsf{com} = \mathsf{Com}_{\mathsf{acct}}(b, \mathsf{root}_{\mathsf{null}}; r)$:
  - update nullifier tree $(\mathsf{st}^{\prime\mathsf{null}}_{\mathsf{Rec}}, \mathsf{root}'_{\mathsf{null}}, \pi_{\mathsf{SMT}}) \leftarrow \mathsf{SMT.Insert}(\mathsf{st}^{\mathsf{null}}_{\mathsf{Rec}}, \mathsf{pid})$
  - sample fresh r' and compute account commitment $\mathsf{com}' \leftarrow \mathsf{Com}_{\mathsf{acct}}(b + v, \mathsf{root}'_{\mathsf{null}}; r')$
  - using the latest root root_ρ in $\mathcal{T}$ compute $\pi_{\mathsf{MMR}} \leftarrow \mathsf{MMR.Prove}(\vec{\rho}, \mathsf{pid})$
  - form the witness $w_{\mathsf{r}} = (b, \mathsf{root}_{\mathsf{null}}, \mathsf{root}'_{\mathsf{null}}, r, r', \rho, \mathsf{pid}, \pi_{\mathsf{MMR}}, \pi_{\mathsf{SMT}}, v, \mathsf{Sen}, r'')$
  - prove $\pi \leftarrow \Pi.\mathsf{Prove}(\mathcal{R}_{\mathsf{recv}}, (\mathsf{Rec}, \mathsf{com}, \mathsf{com}', \mathsf{root}_\rho), w_{\mathsf{r}})$
  - submit $(\mathsf{Rec}, \mathsf{com}', \mathsf{root}_\rho, \pi)$
- ledger:
  - form the statement $x_{\mathsf{r}} = (\mathsf{Rec}, \mathsf{com}, \mathsf{com}', \mathsf{root}_\rho)$ using $\mathsf{com} = \mathsf{Acct}[\mathsf{Rec}]$
  - verify that $\Pi.\mathsf{Verify}(\mathcal{R}_{\mathsf{recv}}, x_{\mathsf{r}}, \pi) = 1$ and $\mathsf{root}_\rho \in \mathcal{T}$
  - if all checks pass, set $\mathsf{Acct}[\mathsf{Rec}] \leftarrow \mathsf{com}'$. else return $\perp$

Figure 4: Bonsai with sender–receiver unlinkability.

$\Pi_{\mathsf{Bonsai}}[\mathcal{L}_{\mathsf{ind}}]$

$\mathsf{Send}(\mathsf{Sen}, \mathsf{Rec}, v)$:

- sender parses $\mathsf{com} = \mathsf{Acct}[\mathsf{Sen}]$ with opening $\mathsf{com} = \mathsf{Com}_{\mathsf{acct}}(b, \mathsf{root}_{\mathsf{null}}; r)$,
  - sample fresh r', r'' and compute account commitment $\mathsf{com}' \leftarrow \mathsf{Com}_{\mathsf{acct}}(b - v, \mathsf{root}_{\mathsf{null}}; r')$
  - compute on chain receipt $\rho \leftarrow \mathsf{Com}_{\mathsf{rec}}(v, \mathsf{Sen}, \mathsf{Rec}, 1; r'')$
  - form the witness $w_{\mathsf{s}} = (b, \mathsf{root}_{\mathsf{null}}, r, r', r'', v, \mathsf{Rec})$
  - prove $\pi \leftarrow \Pi.\mathsf{Prove}(\mathcal{R}_{\mathsf{op}}, (\mathsf{Sen}, \mathsf{com}, \mathsf{com}', \rho, \mathsf{root}_\rho), w_{\mathsf{s}})$ using the latest root root_ρ in $\mathcal{T}$
  - submit $(\mathsf{Sen}, \mathsf{com}', \rho, \mathsf{root}_\rho, \pi)$.
- ledger:
  - form the statement $x = (\mathsf{Sen}, \mathsf{com}, \mathsf{com}', \rho, \mathsf{root}_\rho)$ using $\mathsf{com} = \mathsf{Acct}[\mathsf{Sen}]$
  - verify that $\Pi.\mathsf{Verify}(\mathcal{R}_{\mathsf{op}}, x, \pi) = 1$ and $\mathsf{root}_\rho \in \mathcal{T}$
  - if all checks pass, set $\mathsf{Acct}[\mathsf{Sen}] \leftarrow \mathsf{com}'$. else return $\perp$
  - update the state root $(\mathsf{st}_\rho, \mathsf{root}_\rho) \leftarrow \mathsf{MMR.Insert}(\mathsf{st}_\rho, \rho)$ and record root_ρ in $\mathcal{T}$
- sender reads off the position pid at which the ledger inserted ρ, outputs $(\mathsf{Sent}, \mathsf{pid}, \mathsf{Rec}, v)$, and forwards $\omega = (\rho, \mathsf{pid}, v, \mathsf{Sen}, \mathsf{Rec}, r'')$ to the receiver over the private channel

$\mathsf{Receive}(\mathsf{Rec}, \mathsf{pid})$:

- receiver parses $\omega = (\rho_{\mathsf{in}}, \mathsf{pid}, v, \mathsf{Sen}, \mathsf{Rec}, r'')$ and account opening $\mathsf{com} = \mathsf{Com}_{\mathsf{acct}}(b, \mathsf{root}_{\mathsf{null}}; r)$:
  - update nullifier tree $(\mathsf{st}^{\prime\mathsf{null}}_{\mathsf{Rec}}, \mathsf{root}'_{\mathsf{null}}, \pi_{\mathsf{SMT}}) \leftarrow \mathsf{SMT.Insert}(\mathsf{st}^{\mathsf{null}}_{\mathsf{Rec}}, \mathsf{pid})$
  - sample fresh r' and compute account commitment $\mathsf{com}' \leftarrow \mathsf{Com}_{\mathsf{acct}}(b + v, \mathsf{root}'_{\mathsf{null}}; r')$
  - using the latest root root_ρ in $\mathcal{T}$ compute $\pi_{\mathsf{MMR}} \leftarrow \mathsf{MMR.Prove}(\vec{\rho}, \mathsf{pid})$
  - sample fresh r''' and compute dummy receipt $\rho \leftarrow \mathsf{Com}_{\mathsf{rec}}(0, \perp, \perp, 0; r''')$
  - form the witness $w_{\mathsf{r}} = (b, \mathsf{root}_{\mathsf{null}}, \mathsf{root}'_{\mathsf{null}}, r, r', r''', \rho_{\mathsf{in}}, \mathsf{pid}, \pi_{\mathsf{MMR}}, \pi_{\mathsf{SMT}}, v, \mathsf{Sen}, r'')$
  - prove $\pi \leftarrow \Pi.\mathsf{Prove}(\mathcal{R}_{\mathsf{op}}, (\mathsf{Rec}, \mathsf{com}, \mathsf{com}', \rho, \mathsf{root}_\rho), w_{\mathsf{r}})$
  - submit $(\mathsf{Rec}, \mathsf{com}', \rho, \mathsf{root}_\rho, \pi)$
- ledger:
  - form the statement $x = (\mathsf{Rec}, \mathsf{com}, \mathsf{com}', \rho, \mathsf{root}_\rho)$ using $\mathsf{com} = \mathsf{Acct}[\mathsf{Rec}]$
  - verify that $\Pi.\mathsf{Verify}(\mathcal{R}_{\mathsf{op}}, x, \pi) = 1$ and $\mathsf{root}_\rho \in \mathcal{T}$
  - if all checks pass, set $\mathsf{Acct}[\mathsf{Rec}] \leftarrow \mathsf{com}'$. else return $\perp$
  - update the state root $(\mathsf{st}_\rho, \mathsf{root}_\rho) \leftarrow \mathsf{MMR.Insert}(\mathsf{st}_\rho, \rho)$ and record root_ρ in $\mathcal{T}$

Figure 5: Bonsai with send–receive indistinguishability.

$\mathcal{T}$ by faithfully emulating the ledger party. It additionally maintains the full public transcript, including every recorded receipt. Separately, its private bookkeeping table has entries

$$\mathsf{Pay}[\mathsf{pid}] = (\rho, \mathsf{Sen}, \mathsf{Rec}, v, \mathsf{status}),$$

where $\mathsf{status} \in \{\mathsf{withheld}, \mathsf{pending}, \mathsf{completed}\}$. Certain fields not yet revealed by the ideal functionality may be empty. The ideal functionality and the simulated ledger use the same monotonically increasing receipt-log index where an identifier pid assigned to an accepted send in the ideal functionality is the MMR leaf position at which the corresponding receipt is inserted in the protocol. Sim also initializes its corrupted-ownership table with the initial accounts owned by corrupted parties and extends it after every successful corrupted Create call.

Account creation. When an honest party locally samples $(\mathsf{sk}_A, A) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$ and sends $(\mathsf{Create}, \mathsf{sk}_A, A)$ to the ideal functionality there is no public leakage so Sim does nothing. But for a corrupted caller, Sim forwards the account key pair produced by $\mathcal{A}$ to the ideal functionality via the Create interface and records the corrupted owner whenever the call succeeds.

Registration. When the ideal functionality notifies Sim of an honest party registering their account, Sim creates a dummy account commitment

$$\mathsf{com}_A \leftarrow \mathsf{Com}_{\mathsf{acct}}(0, 0, 0; r_A)$$

with fresh randomness r_A, where the $0s$ denote a canonical input to the commitment. Sim then simulates the registration proof for $\mathcal{R}_{\mathsf{reg}}$, posts it to the simulated ledger, and sets $\mathsf{Acct}[A] \leftarrow \mathsf{com}_A$.

For a registration submitted by a corrupted party, Sim first performs the ledger's duplicate account and NIZK verification checks and reproduces its rejection if either check fails. For an accepted registration it extracts (sk_A, r_A). If $A = \mathsf{VKey}(\mathsf{sk}_A)$ and $\mathsf{com}_A = \mathsf{Com}_{\mathsf{acct}}(\mathsf{Init}[A], \mathsf{root}_\emptyset; r_A)$, then Sim proceeds as follows. If A is not already recorded as belonging to a corrupted party, Sim sends $(\mathsf{Create}, \mathsf{sk}_A, A)$ on behalf of that party and records it in its corrupted owner table. Finally, Sim invokes $(\mathsf{Register}, A)$ on behalf of the recorded corrupted owner. It then sets $\mathsf{Acct}[A] \leftarrow \mathsf{com}_A$.

Send by $\mathcal{A}$. For a send submitted by a corrupted party, Sim forms the statement using the current commitment Acct[Sen] and performs all ledger checks. If the ledger would reject, Sim reproduces that rejection. Otherwise, straight-line simulation extractability yields a witness $(b, \mathsf{root}_{\mathsf{null}}, r, r', r'', v, \mathsf{Rec})$ of $\mathcal{R}_{\mathsf{send}}$. Sim sends $(\mathsf{Send}, \mathsf{Sen}, \mathsf{Rec}, v)$ to the ideal functionality on behalf of the recorded corrupted owner and records at the resulting pid

$$\mathsf{Pay}[\mathsf{pid}] = (\rho, \mathsf{Sen}, \mathsf{Rec}, v, \mathsf{withheld}).$$

It then applies the account update and receipt insertion to its simulated ledger.

Whenever a corrupted sender sends a candidate opening over a private channel to a party P, Sim stores both the opening and its channel destination. As soon as a valid send is posted on the simulated ledger and Sim observes a matching well-formed opening $\omega = (\rho, \mathsf{pid}, v, \mathsf{Sen}, \mathsf{Rec}, r'')$ delivered to the owner of Rec, Sim invokes $(\mathsf{Release}, \mathsf{pid})$, upon which the ideal functionality outputs $(\mathsf{pid}, \mathsf{Sen}, \mathsf{Rec}, v)$ to that owner, and changes the recorded status to pending.

Receive by $\mathcal{A}$. For a receive submitted by a corrupted party, Sim forms the statement using Acct[Rec], checks the revealed root against $\mathcal{T}$, and performs the NIZK verification. If the ledger would reject, Sim reproduces that rejection. For a valid receive, extraction yields a witness of $\mathcal{R}_{\mathsf{recv}}$ containing the consumed receipt, its identifier pid, and its opening proof under the revealed state root.

If the payment is still withheld (meaning the sender must have been corrupted), the accepting witness demonstrates that the corrupted receiver obtained the opening, so Sim first sends $(\mathsf{Release}, \mathsf{pid})$ to the ideal functionality and updates $\mathsf{Pay}[\mathsf{pid}].\mathsf{status} \leftarrow \mathsf{pending}$. It then sends $(\mathsf{Receive}, \mathsf{Rec}, \mathsf{pid})$ to the ideal functionality on behalf of the recorded corrupted owner and updates $\mathsf{Pay}[\mathsf{pid}].\mathsf{status} \leftarrow \mathsf{completed}$. Finally, it applies the accepted account update to its simulated ledger.

Send by Honest parties. An honest send leaks $(\mathsf{Send}, \mathsf{Sen})$ to the adversary. In either case, Sim creates a fresh dummy successor account commitment $\mathsf{com}' \leftarrow \mathsf{Com}_{\mathsf{acct}}(0, 0, 0; r')$.

- If the recipient is corrupted, the ideal functionality gives the simulator $(\mathsf{pid}, \mathsf{Sen}, \mathsf{Rec}, v)$. Sim commits honestly to the corresponding real receipt ρ and records

$$\mathsf{Pay}[\mathsf{pid}] = (\rho, \mathsf{Sen}, \mathsf{Rec}, v, \mathsf{pending}).$$

- If the recipient is honest, Sim creates a dummy receipt $\rho \leftarrow \mathsf{Com}_{\mathsf{rec}}(0, 0, 0; r)$ and records

$$\mathsf{Pay}[\mathsf{pid}] = (\rho, \mathsf{Sen}, -, -, \mathsf{pending}).$$

It simulates the proof for $(\mathsf{Sen}, \mathsf{com}, \mathsf{com}', \rho)$, inserts ρ into the receipt MMR, and records the updated root_ρ in $\mathcal{T}$, and sets $\mathsf{Acct}[\mathsf{Sen}] \leftarrow \mathsf{com}'$ exactly as the ledger does. It then adds the corresponding send to the simulated ledger and, if the receiver is corrupted, delivers the opening ω over the private channel.

Receive by Honest parties. An honest receive leaks $(\mathsf{Receive}, \mathsf{Rec})$ to the adversary. Sim creates a fresh dummy successor account commitment $\mathsf{com}' \leftarrow \mathsf{Com}_{\mathsf{acct}}(0, 0, 0; r')$ and simulates the proof for the receive statement $(\mathsf{Rec}, \mathsf{com}, \mathsf{com}', \mathsf{root}_\rho)$, where root_ρ is the latest recorded state root.

We now show that the joint distribution of the honest party outputs and simulator in the ideal world are computationally indistinguishable from the joint distribution of the honest party outputs and the adversary in the real world, through a sequence of hybrid arguments.

$\mathbf{H}_0$: This is the real execution of Π_{Bonsai}.

$\mathbf{H}_1$: Record each local account-generation event through the private ideal Create interface. Abort if two independently generated honest key pairs have the same verification key which happens with negligible probability by correctness of the signature scheme and EUF-CMA security. Thus, this hybrid is computationally indistinguishable from the previous hybrid.

$\mathbf{H}_2$: Retain the simulation and extraction trapdoors produced alongside the honestly distributed NIZK CRS instead of erasing them. Since this does not change the distribution of the CRS, this hybrid is distributed identically to the previous hybrid.

$\mathbf{H}_3$: From every registration, send, and receive proof submitted by a corrupted party that verifies, extract the witness and abort if it fails. By simulation extractability of the NIZK, this happens with negligible probability and thus this hybrid is computationally indistinguishable from the previous hybrid.

$\mathbf{H}_4$: Abort if a corrupted party supplies through Create, or in an accepting registration witness, a valid signing key sk such that $\mathsf{VKey}(\mathsf{sk}) = A$ for an honestly generated verification key A. This happens with negligible probability due to the EUF-CMA property of the signature scheme and hence this hybrid is computationally indistinguishable from the previous hybrid.

$\mathbf{H}_5$: Abort if any account or receipt commitment used by the adversary has two inconsistent openings. This changes the execution only with negligible probability by computational binding.

$\mathbf{H}_6$: Abort if an accepted receive presents an accepting opening proof under its revealed anchor root_ρ for a position–receipt pair that does not match the receipt log prefix whose insertion produced that root. The ledger only accepts anchors from its recorded history $\mathcal{T}$, each of which it computed itself over that prefix, so such a proof breaks membership binding of the Merkle Mountain Range (Section 4) and hence yields a collision in H. Since only accepted sends insert receipts, every accepted receive consumes a receipt created by one earlier accepted send, at the unique position where the ledger recorded it. This hybrid is computationally indistinguishable from the previous hybrid.

$\mathbf{H}_7$: Abort if two receives by the same account consume receipts at the same position. Binding fixes the tree root committed in each account commitment, registration commits the root of the empty tree, sends carry the committed root through unchanged, and each receive proves an insertion from the committed old root to the committed new root, so the accepted operations of an account form a single chain of verified insertions starting from the empty tree. By the previous hybrid, each receive inserts the proven position of the receipt it consumes, so two receives at the same position insert the same value twice along this chain, which breaks the insertion guarantee of the sparse Merkle tree (Section 4) and hence yields a collision in H. This hybrid is computationally indistinguishable from the previous hybrid.

$\mathbf{H}_8$: Replace every proof made by an honest party with a simulated proof for the same public statement. The zero knowledge property makes this change computationally indistinguishable.

$\mathbf{H}_9$: For each honest party, maintain an internal copy of its private protocol state, including its signing keys, account openings, balances, nullifier trees, and received receipt openings. Update this copy in lockstep with the party's actual state, but do not yet use it to determine any message or output. The copy is therefore invisible and does not change the execution distribution. It will preserve the honest party's underlying semantic state when later hybrids replace its public commitments.

$\mathbf{H}_{10}$: Abort if a corrupted party supplies a fresh accepting send/receive proof against an account owned by an honest party. To bound this event we reduce it to computational hiding and binding properties of the commitment scheme. First guess the targeted honest account commitment, losing only a polynomial factor, and embed in it the challenge com^* (in the commitment game) that is either the honest account state or the dummy state $(0, 0, 0)$. Use the NIZK simulation trapdoor to generate all affected honest proofs. If the abort occurs, simulation extractability recovers an opening $(\widetilde{m}, \widetilde{r})$ of com^* the simulator outputs its guess for what the commitment contains. By computational binding of the commitment scheme $\widetilde{m} = m_b$ (b is the challenge bit) must be either the honest account commitment or the dummy state. A non-negligible abort probability would therefore contradict computational hiding. Hence this abort occurs only with negligible probability and this hybrid is computationally indistinguishable from the previous hybrid.

$\mathbf{H}_{11}$: Replace all honest account commitments by fresh dummy commitments $\mathsf{Com}_{\mathsf{acct}}(0, 0, 0; r)$ through a sequence of subhybrids. Order the commitments by when they are generated and, in the j-th subhybrid, replace only the j-th commitment and update all affected simulated proofs. The indistinguishability of subhybrids reduces to the hiding game for $\mathsf{Com}_{\mathsf{acct}}$. The preceding hybrid guarantees that the adversary cannot produce an accepting proof that requires an opening of the challenged commitment. Since a polynomial-time execution only generates polynomially many account commitments, there are a polynomial number of subhybrids and therefore the first and last subhybrids are computationally indistinguishable.

$\mathbf{H}_{12}$: Next Sim replaces all receipt commitments created by an honest sender whose opening is delivered to an honest receiver, by the dummy receipt $\mathsf{Com}_{\mathsf{rec}}(0, 0, 0; r)$ with fresh randomness. Sim also updates every subsequent receipt root and simulates all of the affected send/receive proofs. It makes these replacements one at a time in ledger order over subhybrids and adjacent subhybrids are computationally indistinguishable. Each adjacent subhybrid reduces to the hiding game for $\mathsf{Com}_{\mathsf{rec}}$. Furthermore, there are at most a polynomial number of receipts in the protocol and hence a polynomial number of subhybrids and therefore the first and last subhybrid are computationally indistinguishable.

$\mathbf{H}_{13}$: In the final hybrid we replace the internal bookkeeping for honest parties by the corresponding calls to the ideal functionality, and let Sim generate the public execution from the resulting leakage. Everything in $\mathcal{A}$'s view is then either data provided to Sim by the ideal functionality or a dummy commitment, simulated proof, or public value derived from the simulated transcript. This corresponds to the ideal execution produced by Sim and this hybrid is distributed identically to the previous hybrid.

□

We now turn to the operation-indistinguishable variant of Bonsai, $\Pi_{\mathsf{Bonsai}}[\mathcal{L}_{\mathsf{ind}}]$ of Fig. 5.

Theorem 2 (Security of $\Pi_{\mathsf{Bonsai}}[\mathcal{L}_{\mathsf{ind}}]$)**.** *Under the assumptions of Theorem 1, with* Π *a secure NIZK for the* $\mathcal{R}_{\mathsf{reg}}$ *and* $\mathcal{R}_{\mathsf{op}}$ *relations (Definition 4),* $\Pi_{\mathit{Bonsai}}[\mathcal{L}_{\mathsf{ind}}]$ *securely realizes* $\mathcal{F}_{\mathsf{PrivPay}}[\mathcal{L}_{\mathsf{ind}}]$ *against static malicious corruption of any set of users.*

Proof sketch. The simulator and hybrid sequence are those of Theorem 1, adapted to the two main differences between the constructions:

- Both send and receive operations have the common format $(A, \mathsf{com}', \rho, \mathsf{root}_\rho, \pi)$ and their proofs prove the same relation $\mathcal{R}_{\mathsf{op}}$,

and the leakage reveals only the acting account, so Sim is never told whether an honest operation is a send or a receive. We describe the changes these differences require.

Receipt-log positions. Every accepted operation, whether a send or a receive, inserts exactly one receipt into the MMR. Correspondingly, $\mathcal{F}_{\mathsf{PrivPay}}[\mathcal{L}_{\mathsf{ind}}]$ increments its position counter on every accepted receive (Fig. 1), so the receipt-log index advances at every accepted operation in both worlds. The identifier pid of an accepted send is still the leaf position of its receipt, while positions skipped by receives hold dummy receipts in the protocol, are unassigned in Log, and are never payment identifiers.

Operations by $\mathcal{A}$. Sim performs the ledger checks for the common statement $x = (A, \mathsf{com}, \mathsf{com}', \rho, \mathsf{root}_\rho)$ and, for an accepted record, extracts $(\mathsf{op}, w_{\mathsf{op}})$ from the proof for $\mathcal{R}_{\mathsf{op}}$. The branch selector tells Sim which operation the corrupted party performed, after which it proceeds exactly as in the "Send by $\mathcal{A}$" or "Receive by $\mathcal{A}$" case of the $\mathcal{L}_{\mathsf{unl}}$ simulator, additionally inserting the published ρ into the MMR in both cases.

Operations by honest parties. Sim learns only the acting account A. It always publishes a fresh dummy account commitment $\mathsf{com}' \leftarrow \mathsf{Com}_{\mathsf{acct}}(0, 0, 0; r')$, the latest recorded root root_ρ, and a simulated proof. For the receipt there are two cases. If the operation is a send to a corrupted recipient, the payment data $(\mathsf{pid}, \mathsf{Sen}, \mathsf{Rec}, v)$ destined for that recipient is delivered to Sim, from which it learns that the operation is a send and publishes the real receipt $\mathsf{Com}_{\mathsf{rec}}(v, \mathsf{Sen}, \mathsf{Rec}, 1; r'')$ as before. Otherwise, whether the operation is a send to an honest recipient or a receive, Sim publishes the canonical dummy receipt $\mathsf{Com}_{\mathsf{rec}}(0, \bot, \bot, 0; r)$.

Hybrids. The hybrid sequence carries over with the following modifications.

- The Merkle Mountain Range binding hybrid now guarantees that every receive branch consumes the receipt published by one earlier accepted operation, which may itself be a receive. We add one hybrid that aborts if the consumed receipt was published by a receive. Such a receipt has a known opening as $\mathsf{Com}_{\mathsf{rec}}(0, \bot, \bot, 0; r''')$, either held by the honest publisher or extracted from the corrupted one, whereas the receive branch opens it with type 1. Two distinct openings contradict computational binding of $\mathsf{Com}_{\mathsf{rec}}$, so every consumed receipt was created by the send branch of an earlier operation and carries a payment identifier in the ideal functionality.
- The abort on corrupted proofs against honest accounts now covers both branches of $\mathcal{R}_{\mathsf{op}}$. Since the receive branch requires the acting account to equal the receipt's recipient, a corrupted party can never consume a receipt addressed to an honest account.

After the receipt-replacement hybrid, a send from an honest party to an honest receiver and a receive from an honest party publish identically distributed records: a dummy account commitment, a canonical dummy receipt $\mathsf{Com}_{\mathsf{rec}}(0, \bot, \bot, 0; r)$, the latest recorded root, and a simulated proof. The final hybrid can therefore generate the public execution from the $\mathcal{L}_{\mathsf{ind}}$ leakage alone, and is computationally indistinguishable from a real-world execution. □

7 ZK-PARI: A zkSNARK with Batch Verification

We construct a zero-knowledge SNARK for Square R1CS starting from PARI [DMS24] and its improvement [Eag25]. The latter optimizes the proof size to $2\mathbb{G}_1 + \mathbb{F}$, but neither work provides zero knowledge, which is crucial to protecting user privacy. Concurrent work adds zero knowledge to the original $2\mathbb{G}_1 + 2\mathbb{F}$ version of PARI [KPS26]. In contrast, we add zero knowledge to the optimized $2\mathbb{G}_1+\mathbb{F}$ version with no change to its proof size, negligible prover overhead, and preserving the batch-verification strategy from [Pol26].

7.1 ZK-PARI

We extend PARI [DMS24, Eag25] to support zero-knowledge using vanishing polynomials. Importantly, the masks do not add to the proof size which remains at $2\mathbb{G}_1 + \mathbb{F}$ as in [Eag25]. We write the Square R1CS index as $\mathtt{i} = (\mathbb{F}, n, k, m, H, A, B)$ and partition the assignment as $z = (\mathbb{x} \| w)$, where $\mathbb{x} \in \mathbb{F}^n$ is the public statement and $w \in \mathbb{F}^{k-n}$ is the witness, so that

$$\mathcal{R}^{\mathsf{sq}}_{\mathtt{i}}(\mathbb{x}, w) = 1 \quad \iff \quad (Az) \circ (Az) = Bz.$$

The set $H = \{h_1, \ldots, h_m\} \subset \mathbb{F}$ indexes the constraint rows, with row t assigned to the point h_t, and $Z_H(X) := \prod_{\omega \in H}(X - \omega)$ is its vanishing polynomial.

The underlying IOP. Before presenting our construction we first recall the IOP underlying the Pari proof system. In a polynomial IOP, the prover sends oracles to polynomials, written $\boxed{p(X)}$, and the verifier queries them at points of its choice. Let $\bar{z}_A$ and $\bar{z}_B$ denote the interpolations of Az and Bz over H, so that z satisfies the instance exactly when $\bar{z}_A^2 - \bar{z}_B$ vanishes on H. To accept, the verifier must be convinced of two separate facts: (1) that the above polynomial evaluates to 0 on H, and (2) that the polynomials behind the oracles really are interpolations of Az and Bz for some z whose first n coordinates equal $\mathbb{x}$. The second condition is a linear-consistency check, commonly called a lincheck.

- The lincheck is enforced by the oracle model rather than by queries where the prover is restricted to sending oracle pairs of the form

  $$\bar{z}_A(X) = \sum_{i=1}^{k} z_i \, a_i(X), \qquad \bar{z}_B(X) = \sum_{i=1}^{k} z_i \, b_i(X)$$

  for a common vector $z \in \mathbb{F}^k$. The verifier can compute the contribution from public coordinates themselves from $A'\mathbb{x}$ and $B'\mathbb{x}$. Looking ahead, this will be enforced in the instantiation by embedding entries of A and B in the CRS.

- The prover computes the quotient $\tilde{q} := (\bar{z}_A^2 - \bar{z}_B)/Z_H$, which is a low-degree polynomial if and only if $\bar{z}_A^2 - \bar{z}_B$ vanishes on H, and sends the oracles $\boxed{\bar{z}_A(X)}$, $\boxed{\bar{z}_B(X)}$, and $\boxed{\tilde{q}(X)}$.

- The verifier samples a challenge $\zeta \leftarrow \mathbb{F} \setminus H$, queries the three oracles at ζ, and accepts if and only if

  $$\bar{z}_A(\zeta)^2 - \bar{z}_B(\zeta) = Z_H(\zeta)\, \tilde{q}(\zeta).$$

  Suppose $\bar{z}_A^2 - \bar{z}_B - Z_H\tilde{q}$ does not vanish on H, but since it is a nonzero polynomial of degree at most $2m - 2$, the check passes with probability at most $(2m-2)/(q-m)$ by the Schwartz–Zippel lemma.

The above IOP is not zero knowledge, and its compilation in [DMS24] exposes the committed polynomials at two points: the verifier queries at ζ, and the KZG-style commitments are themselves evaluations at the CRS point τ in the exponent. To ensure the proof is simulatable, the prover masks $\bar{z}_A$ as $\hat{z}_A := \bar{z}_A + (\eta_1 + \eta_2 X)\, Z_H$ for uniform $\eta_1, \eta_2 \leftarrow \mathbb{F}$.

ZK-PARI

$\mathsf{Setup}_{\mathsf{Pari}}(1^\lambda, \mathbb{i}) \to (\mathsf{crs}_{\mathbb{i}}, \mathsf{td})$.

1. Parse the Square R1CS index $\mathbb{i}$ as $(\mathbb{F}, n, k, m, H, A, B)$, where $A, B \in \mathbb{F}^{m\times k}$. Set $\mathsf{id}_{\mathbb{i}} := \mathsf{HashIdx}(\mathbb{i})$ and form the succinct index $\mathbb{i}_{\mathsf{s}} := (\mathbb{F}, n, k, m, H, A', B', \mathsf{id}_{\mathbb{i}})$, where $A', B' \in \mathbb{F}^{m\times n}$ are the public-input columns of A and B.
2. Sample a bilinear group $\langle \mathsf{group}\rangle := \langle \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G, H, e\rangle \leftarrow \mathsf{SampleGrp}(1^\lambda, \mathbb{F})$.
3. Interpolate the columns of A and B over H to obtain $\{a_i(X)\}_{i=1}^k$ and $\{b_i(X)\}_{i=1}^k$. Extend these bases by $a_{k+1} := Z_H(X)$, $a_{k+2} := XZ_H(X)$, and $b_{k+1} = b_{k+2} := 0$.
4. Sample $\alpha, \beta, \delta, \tau \leftarrow \mathbb{F}$.
5. Compute the opening keys $\Sigma_A := \big[\alpha\tau^i G\big]_{i=0}^{m}$ and $\Sigma_R := \big[\beta\tau^i G\big]_{i=0}^{2m+1}$.
6. Let $W := \{n+1, \ldots, k\} \cup \{k+1, k+2\}$, and compute the commitment keys
$$\Sigma_W := \left[\frac{\alpha\, a_i(\tau) + \beta\, b_i(\tau)}{\delta}\, G\right]_{i\in W}, \qquad \Sigma_Q := \left[\frac{\beta\, Z_H(\tau)\tau^i}{\delta}\, G\right]_{i=0}^{m+2}.$$
7. Set $\mathsf{ipk} := (\langle \mathsf{group}\rangle, \mathbb{i}, \Sigma_W, \Sigma_Q, \Sigma_A, \Sigma_R)$ and $\mathsf{ivk}_0 := (\langle \mathsf{group}\rangle, \mathbb{i}_{\mathsf{s}}, \alpha G, \beta G, \delta H, \tau H)$. Set $\mathsf{id}_{\mathsf{vk}} := \mathsf{HashVK}(\mathsf{ivk}_0)$ and $\mathsf{ivk} := (\mathsf{ivk}_0, \mathsf{id}_{\mathsf{vk}})$. Set $\mathsf{crs}_{\mathbb{i}} := (\mathsf{ipk}, \mathsf{ivk})$ and $\mathsf{td} := (\alpha, \beta, \delta, \tau)$. Publish $\mathsf{crs}_{\mathbb{i}}$ and erase td.

$\mathsf{Prove}^{\rho}(\mathsf{ipk}, \mathbb{x}, w) \to \pi_{\mathsf{pari}}$.

1. Parse ipk as $(\langle \mathsf{group}\rangle, \mathbb{i}, \Sigma_W, \Sigma_Q, \Sigma_A, \Sigma_R)$, and write $z := (\mathbb{x} \| w)$.
2. Interpolate the assignment and the statement: $\bar{z}_A(X) := \sum_{i=1}^k z_i\, a_i(X)$, $\bar{z}_B(X) := \sum_{i=1}^k z_i\, b_i(X)$, $\hat{x}_A(X) := \sum_{i=1}^n \mathbb{x}_i\, a_i(X)$, and $\hat{x}_B(X) := \sum_{i=1}^n \mathbb{x}_i\, b_i(X)$.
3. Sample $\eta_1, \eta_2 \leftarrow \mathbb{F}$.
4. Mask the A-side assignment polynomial: $\hat{z}_A(X) := \bar{z}_A(X) + (\eta_1 + \eta_2 X)\, Z_H(X)$. Compute the quotient $\tilde{q}(X) := (\hat{z}_A^2(X) - \bar{z}_B(X))/Z_H(X)$, of degree at most $m+2$.
5. Compute
$$T := \sum_{j=1}^{k-n} w_j \cdot \Sigma_W[n+j] + \eta_1 \cdot \Sigma_W[k+1] + \eta_2 \cdot \Sigma_W[k+2] + \sum_{i=0}^{m+2} \tilde{q}[i] \cdot \Sigma_Q[i].$$
6. Derive the challenge $\zeta := \rho(\mathsf{transcript})$ and evaluate $v_a := \hat{z}_A(\zeta) - \hat{x}_A(\zeta)$.
7. Compute the KZG witness polynomials, of degrees at most m and $2m+1$:
$$\hat{W}_A(X) := \frac{\hat{z}_A(X) - \hat{x}_A(X) - v_a}{X - \zeta}, \quad \hat{W}_R(X) := \frac{\bar{z}_B(X) - \hat{x}_B(X) + Z_H(X)\tilde{q}(X) - v_R}{X - \zeta},$$
where $v_R := (v_a + \hat{x}_A(\zeta))^2 - \hat{x}_B(\zeta)$.
8. Compute the batched opening proof $U := \sum_{i=0}^m \hat{W}_A[i] \cdot \Sigma_A[i] + \sum_{i=0}^{2m+1} \hat{W}_R[i] \cdot \Sigma_R[i]$. Output $\pi := (T, U, v_a)$.

$\mathsf{Verify}^{\rho}(\mathsf{ivk}, \mathbb{x}, \pi) \to b$.

1. Parse $\pi = (T, U, v_a)$ and $\mathsf{ivk} = (\mathsf{ivk}_0, \mathsf{id}_{\mathsf{vk}})$, where $\mathsf{ivk}_0 = (\langle \mathsf{group}\rangle, \mathbb{i}_{\mathsf{s}}, \alpha G, \beta G, \delta H, \tau H)$. Recompute the challenge $\zeta := \rho(\mathsf{transcript})$.
2. Compute the vectors $x_A := A'\mathbb{x}$ and $x_B := B'\mathbb{x}$, and interpolate them over H to obtain $\hat{x}_A$ and $\hat{x}_B$. Compute $v_R := (v_a + \hat{x}_A(\zeta))^2 - \hat{x}_B(\zeta)$.
3. Accept if and only if
$$e(T,\ \delta H) \ =\ e(U,\ \tau H - \zeta \cdot H) \ \cdot\ e\big(v_a \cdot \alpha G + v_R \cdot \beta G,\ H\big).$$

Figure 6: ZK-PARI, with the zero-knowledge modifications highlighted.

7.2 Security

Theorem 3 (Perfect correctness)**.** *ZK-*PARI *(Fig. 6) satisfies perfect completeness (Definition 4).*

Proof. The mask vanishes on H, so it preserves the Square R1CS constraints. Consequently, $\hat{z}_A^2 - \bar{z}_B$ is divisible by Z_H, and $\tilde{q} = (\hat{z}_A^2 - \bar{z}_B)/Z_H$ has degree at most $m+2$. Weighting the first-message commitment gives

$$\delta\, T = \alpha\big(\hat{z}_A(\tau) - \hat{x}_A(\tau)\big)G + \beta\big(\bar{z}_B(\tau) - \hat{x}_B(\tau) + Z_H(\tau)\tilde{q}(\tau)\big)G.$$

The left side commits to

$$\alpha(\hat{z}_A - \hat{x}_A) + \beta(\bar{z}_B - \hat{x}_B + Z_H\tilde{q}).$$

The verification equation is the KZG opening identity for this commitment at ζ, so it holds for every $\zeta \in \mathbb{F} \setminus H$. □

Computational statements hold in the algebraic group model, in which the prover supplies a representation of every group element it outputs as a linear combination of the group elements it has received. The only computational assumption is the following q-type discrete-logarithm assumption.

Definition 6 ((q_1, q_2)-DLOG)**.** *The (q_1, q_2)-discrete-logarithm assumption holds if for every PPT adversary $\mathcal{A}$, the probability*

$$\Pr\left[\langle \mathsf{group}\rangle \leftarrow \mathsf{SampleGrp}(1^\lambda, \mathbb{F}),\ x \leftarrow \mathbb{F} : \mathcal{A}\big(\langle \mathsf{group}\rangle, \{x^i G\}_{i=1}^{q_1}, \{x^i H\}_{i=1}^{q_2}\big) = x\right] \le \epsilon_{\mathsf{dlog}}(\lambda).$$

We now prove knowledge soundness.

Theorem 4 (Knowledge soundness)**.** *For every index $\mathbb{i}$, the public-coin variant of ZK-*PARI *is a knowledge argument in the algebraic group model under the $(2m+3, 1)$-DLOG assumption: from any accepting algebraic prover, the extractor obtains w with $\mathcal{R}_{\mathbb{i}}^{\mathsf{sq}}(\mathbb{x}, w) = 1$. Its knowledge error is at most*

$$\epsilon_{\mathsf{dlog}} + \frac{4m+6}{q-m}.$$

Proof. The algebraic prover supplies representations of T and U as linear combinations of the $\mathbb{G}_1$ elements of $\mathsf{crs}_{\mathbb{i}}$, namely G, αG, βG, Σ_A, Σ_R, Σ_W, and Σ_Q. We treat $\alpha, \beta, \delta, \tau$ as formal variables, so that every exponent becomes a rational function whose denominator divides δ. The verification equation asserts that the rational function

$$\delta\langle T\rangle - (\tau - \zeta)\langle U\rangle - v_a\,\alpha - v_R\,\beta$$

evaluates to zero at the sampled trapdoor point. To simplify our analysis, we convert this rational function into a polynomial by multiplying it by δ:

$$P := \delta\big(\delta\langle T\rangle - (\tau - \zeta)\langle U\rangle - v_a\,\alpha - v_R\,\beta\big). \tag{2}$$

An accepting transcript therefore implies that P also evaluates to zero at the sampled trapdoor point. Observe that P is a polynomial of total degree at most $2m+4$ in the formal variables $\alpha, \beta, \delta, \tau$, and its coefficients are known from the representations. The largest CRS exponents are $\beta\tau^{2m+1}$ from Σ_R and $\beta Z_H(\tau)\tau^{m+2}/\delta$ from Σ_Q, of numerator degree $2m+2$ and $2m+3$, and each term of P multiplies one such exponent by a prefactor of degree at most two, namely δ^2 or $(\tau - \zeta)\delta$, minus any cancelled denominator. There are now two cases:

Case 1: $P \neq 0$. We reduce to $(2m+3, 1)$-DLOG. Given $\{x^i G\}_{i \le 2m+3}$ and xH, sample independent uniform $a_t, b_t \leftarrow \mathbb{F}$ for each trapdoor $t \in \{\alpha, \beta, \delta, \tau\}$ and set $t := a_t x + b_t$, which is uniform and independent of the other trapdoors for any fixed x. Rescale the $\mathbb{G}_1$ generator by δ, which preserves the CRS distribution. Every $\mathbb{G}_1$ element of $\mathsf{crs}_{\mathbb{i}}$ then has an exponent that is a polynomial of degree at most $2m+3$ in x, and the $\mathbb{G}_2$ elements $\delta H, \tau H$ are affine in x, so the reduction computes the entire CRS from the instance.

Let $d \le 2m+4$ be the total degree of P and let P_d be its degree-d homogeneous component, which is nonzero by the definition of d. Substituting $t := a_t x + b_t$ into P yields a univariate polynomial P_x of degree

at most d whose coefficient on x^d equals $P_d(a_\alpha, a_\beta, a_\delta, a_\tau)$. Since the a_t's are uniform and independent of P, the Schwartz–Zippel lemma guarantees that

$$\Pr\big[P_d(a_\alpha, a_\beta, a_\delta, a_\tau) = 0\big] \le \frac{d}{q} \le \frac{2m+4}{q}.$$

Thus P_x is not identically zero, except with probability $\le (2m+4)/q$. When P_x is not zero, an accepting transcript implies that $P_x(x) = P(\alpha, \beta, \delta, \tau) = 0$, so x is one of the at most $2m+4$ roots of P_x. The reduction factors P_x, tests each root x' against the instance by checking $x'G = xG$, and outputs the match, thereby solving the $(2m+3, 1)$-DLOG problem. An accepting transcript with $P \neq 0$ therefore occurs with probability at most $\epsilon_{\mathsf{dlog}} + (2m+4)/q$.

Case 2: $P = 0$. The representation of an element $E \in \{T, U\}$ is a linear combination of the $\mathbb{G}_1$ elements of $\mathsf{crs}_{\mathbb{i}}$, written in the exponent as:

$$\langle E\rangle = \underbrace{c}_{G} + \underbrace{c_\alpha\,\alpha}_{\alpha G} + \underbrace{c_\beta\,\beta}_{\beta G} + \underbrace{\sum_{i=0}^{m} A_i\,\alpha\tau^i}_{\Sigma_A} + \underbrace{\sum_{i=0}^{2m+1} R_i\,\beta\tau^i}_{\Sigma_R} + \underbrace{\sum_{i\in W} w_i\,\frac{\alpha\,a_i(\tau) + \beta\,b_i(\tau)}{\delta}}_{\Sigma_W} + \underbrace{\sum_{i=0}^{m+2} q_i\,\frac{\beta\,Z_H(\tau)\,\tau^i}{\delta}}_{\Sigma_Q}.$$

Collecting terms, this can be written as

$$\langle E\rangle = c + \alpha A(\tau) + \beta R(\tau) + \frac{\alpha\,\hat{w}_A(\tau) + \beta\big(\hat{w}_B(\tau) + Z_H(\tau)\,q(\tau)\big)}{\delta},$$

where $A(X) := c_\alpha + \sum_i A_i X^i$ and $R(X) := c_\beta + \sum_i R_i X^i$ absorb $\alpha G = \Sigma_A[0]$ and $\beta G = \Sigma_R[0]$, $q(X) := \sum_i q_i X^i$, and

$$\hat{w}_A(X) := \sum_{i\in W} w_i\,a_i(X), \qquad \hat{w}_B(X) := \sum_{i\in W} w_i\,b_i(X).$$

Each of the two elements (T, U) carries its own set of coefficients, and we superscript them with the element when a specific one is meant, as in A^U or $\hat{w}_A^T$. Substituting these representations into Eq. (2) and sorting by the powers of δ yields three monomials, each of which must vanish as a polynomial in (α, β, τ) since $P = 0$. Two of them receive contributions from a single element:

$$\begin{aligned}
\delta^2: &\quad c^T + \alpha A^T(\tau) + \beta R^T(\tau) = 0,\\
\delta^0: &\quad -(\tau-\zeta)\big(\alpha\,\hat{w}_A^U(\tau) + \beta(\hat{w}_B^U(\tau) + Z_H(\tau)\,q^U(\tau))\big) = 0.
\end{aligned}$$

Each displayed identity holds in the polynomial ring $\mathbb{F}[\alpha, \beta, \tau]$, and its left side is linear in α and β over $\mathbb{F}[\tau]$, so the coefficient of each of the monomials 1, α, and β is the zero polynomial in $\mathbb{F}[\tau]$.

- From δ^2: $c^T = 0$ and $A^T = R^T = 0$, so T receives no contribution outside Σ_W and Σ_Q.
- From δ^0: $\hat{w}_A^U = 0$ and $\hat{w}_B^U + Z_H\,q^U = 0$, so U receives no contribution outside G, Σ_A, and Σ_R.

The third monomial δ^1 mixes contributions from both elements and is the only one containing the evaluation terms $v_a\,\alpha$ and $v_R\,\beta$. Comparing coefficients of 1, α, and β as before gives

$$\begin{aligned}
1: &\quad -(\tau-\zeta)\,c^U = 0,\\
\alpha: &\quad \hat{w}_A^T(\tau) - (\tau-\zeta)A^U(\tau) - v_a = 0,\\
\beta: &\quad \hat{w}_B^T(\tau) + Z_H(\tau)\,q^T(\tau) - (\tau-\zeta)R^U(\tau) - v_R = 0,
\end{aligned}$$

so $c^U = 0$ as well. Define the candidate assignment polynomials

$$Z_A(X) := \hat{x}_A(X) + \hat{w}_A^T(X), \qquad Z_B(X) := \hat{x}_B(X) + \hat{w}_B^T(X).$$

In this notation, the α and β identities above imply that:

$$Z_A(X) - \hat{x}_A(X) - v_a = (X-\zeta)A^U(X), \qquad Z_B(X) - \hat{x}_B(X) + Z_H(X)\,q^T(X) - v_R = (X-\zeta)R^U(X),$$

hence $Z_A(\zeta) - \hat{x}_A(\zeta) = v_a$ and $Z_B(\zeta) - \hat{x}_B(\zeta) + Z_H(\zeta)\,q^T(\zeta) = v_R$. Combining them with $v_R = (v_a + \hat{x}_A(\zeta))^2 - \hat{x}_B(\zeta)$ yields

$$Z_A(\zeta)^2 - Z_B(\zeta) - Z_H(\zeta)\,q^T(\zeta) = 0. \tag{3}$$

The extractor outputs the witness $w := (w_i^T)_{i=n+1}^{k}$, the restriction of T's coefficients to the witness columns without the masking coefficients w_{k+1}^T and w_{k+2}^T. It remains to bound the probability that $(\mathbb{x}, w)$ violates the Square R1CS instance even though the transcript is accepting. To this end, define the polynomial

$$F(X) := Z_A(X)^2 - Z_B(X) - Z_H(X)\,q^T(X),$$

of degree at most $2m+2$: the interpolated columns have degree at most $m-1$ and the largest masking column is $a_{k+2} = XZ_H$ of degree $m+1$, so $\deg Z_A^2 \le 2m+2$, while $\deg Z_B \le m-1$ and $\deg\big(Z_H\,q^T\big) \le m + (m+2) = 2m+2$ because Σ_Q supports quotients only up to degree $m+2$. The polynomial F has the following two properties:

- $F(\zeta) = 0$, by Eq. (3).
- $F(h_t) = (Az)_t^2 - (Bz)_t$ for every constraint row $t \in [m]$, where A and B are the constraint matrices of the index $\mathbb{i}$, the vector $z := (\mathbb{x} \| w) \in \mathbb{F}^k$ is the extracted assignment, and $h_t \in H$ is the interpolation point of row t. Indeed, every masking contribution is a multiple of Z_H, namely $a_{k+1} = Z_H$, $a_{k+2} = XZ_H$, and $Z_H\,q^T$, and $Z_H(h_t) = 0$, while $b_{k+1} = b_{k+2} = 0$. The surviving columns interpolate the matrix entries, $a_i(h_t) = A_{t,i}$ and $b_i(h_t) = B_{t,i}$, so $Z_A(h_t) = (Az)_t$ and $Z_B(h_t) = (Bz)_t$.

If $(\mathbb{x}, w)$ violated the instance, then by the second property $F \neq 0$, while by the first the challenge ζ, uniform over $\mathbb{F} \setminus H$ and sampled after F is determined, would land on one of its at most $2m+2$ roots, an event of probability at most $(2m+2)/(q-m)$. Except with this probability, the extracted witness w satisfies $\mathcal{R}_{\mathbb{i}}^{\mathsf{sq}}(\mathbb{x}, w) = 1$. The extractor is straight line: it reads w off the representation of T in the final proof and never rewinds the prover. Together with Case 1, this gives the stated knowledge error. □

Theorem 5 (Honest-verifier zero knowledge)**.** *For honestly generated parameters, the public-coin variant of ZK-*PARI *is statistical honest-verifier zero knowledge with distance at most*

$$\frac{1}{q-m} + \frac{m}{q}.$$

Proof. Fix a satisfiable $\mathbb{x}$ and a witness w. Given $(\alpha, \beta, \delta, \tau)$, the simulator:

- Sample ζ uniformly from $\mathbb{F} \setminus (H \cup \{\tau\})$.
- Compute $\hat{x}_A(\zeta), \hat{x}_B(\zeta)$ and $\hat{x}_A(\tau), \hat{x}_B(\tau)$ by interpolating $A'\mathbb{x}$ and $B'\mathbb{x}$ over H.
- Sample $v_a, y \leftarrow \mathbb{F}$ independently and uniformly, where y represents $\hat{z}_A(\tau)$.
- Define
$$v_R := (v_a + \hat{x}_A(\zeta))^2 - \hat{x}_B(\zeta).$$
- Set
$$T := \frac{\alpha\,(y - \hat{x}_A(\tau)) + \beta\,(y^2 - \hat{x}_B(\tau))}{\delta} \cdot G.$$
- Compute the unique accepting opening proof
$$U := \frac{\delta}{\tau - \zeta} \cdot T - \frac{\alpha\, v_a + \beta\, v_R}{\tau - \zeta} \cdot G.$$

- Output

$$\big(T,\ \zeta,\ (U, v_a)\big).$$

Consider the following hybrids.

$\mathbf{H}_1$: Start with the real proof distribution.

$\mathbf{H}_2$: Sample ζ from $\mathbb{F} \setminus (H \cup \{\tau\})$. Conditioning a uniform $\zeta \in \mathbb{F} \setminus H$ in this way changes the distribution by exactly $1/(q-m)$.

$\mathbf{H}_3$: Replace

$$\big(\hat{z}_A(\zeta), \hat{z}_A(\tau)\big)$$

by independent uniform elements (z_ζ, y), and set

$$v_a := z_\zeta - \hat{x}_A(\zeta).$$

The map

$$(\eta_1, \eta_2) \longmapsto \big(\eta_1 + \eta_2\zeta,\ \eta_1 + \eta_2\tau\big)$$

is linear with matrix

$$\begin{pmatrix} 1 & \zeta \\ 1 & \tau \end{pmatrix}$$

and determinant $\tau - \zeta$. Since $\zeta \neq \tau$, this map is invertible, so $(\eta_1 + \eta_2\zeta,\ \eta_1 + \eta_2\tau)$ is uniform over $\mathbb{F}^2$, as is

$$\big(\hat{z}_A(\zeta), \hat{z}_A(\tau)\big) = \big(\bar{z}_A(\zeta) + (\eta_1 + \eta_2\zeta)Z_H(\zeta),\ \bar{z}_A(\tau) + (\eta_1 + \eta_2\tau)Z_H(\tau)\big),$$

because $Z_H(\zeta), Z_H(\tau) \neq 0$.

$\mathbf{H}_4$: Compute T from $y = \hat{z}_A(\tau)$:

$$T := \frac{\alpha\,(y - \hat{x}_A(\tau)) + \beta\,(y^2 - \hat{x}_B(\tau))}{\delta} \cdot G.$$

This preserves the distribution because

$$\delta\, T = \alpha\,(\hat{z}_A(\tau) - \hat{x}_A(\tau))G + \beta\big(\bar{z}_B(\tau) - \hat{x}_B(\tau) + Z_H(\tau)\tilde{q}(\tau)\big)G.$$

Moreover,

$$\bar{z}_B(\tau) + Z_H(\tau)\tilde{q}(\tau) = \hat{z}_A(\tau)^2,$$

so the displayed expression is the real T as a function of y.

$\mathbf{H}_5$: Compute U from the trapdoor:

$$U := \frac{\delta}{\tau - \zeta} \cdot T - \frac{\alpha\, v_a + \beta\, v_R}{\tau - \zeta} \cdot G.$$

For $\zeta \neq \tau$, the verifier equation

$$e(T, \delta H) = e(U, (\tau - \zeta)H) \cdot e((\alpha v_a + \beta v_R)G, H)$$

has a unique accepting U, so this is the real opening proof.

$\mathbf{H}_6$: The resulting distribution is the simulator's output.

All transitions are perfect except the conditioning step, whose distance is $1/(q-m)$, and the third step, which requires $Z_H(\tau) \neq 0$. Since $\mathsf{Setup}_{\mathsf{Pari}}$ samples τ uniformly from $\mathbb{F}$, this fails only when $\tau \in H$, an event of probability m/q. □

Remark 2 (Simulation extractability)**.** *Theorems 4 and 5 establish knowledge soundness and zero knowledge but the payment protocol of Section 6 additionally relies on simulation extractability (Definition 4). We expect that ZK-*PARI *is simulation extractable, but we defer a formal proof to future work.*

8 Implementation and Evaluation

We implement the ZK-PARI proof system in Rust using `arkworks` [ac22] for finite-field, elliptic-curve, and pairing operations. Our implementation is publicly available.[1] All experiments use BLS12-381 and were run on an M5 MacBook Pro with 48 GB of RAM. To isolate proof-system costs, all timings operate on uncompressed, already subgroup-checked points in memory and exclude serialization, deserialization, and subgroup-membership checks.

Proof Generation and Verification We benchmark a Square R1CS squaring chain with exactly one public input. In Table 1 we benchmark the prover time by varying the number of constraints. We note that when benchmarked against the author's implementation of Pari[2] that does not have zero-knowledge there was negligible overhead for the prover due to the masking polynomials.

We also benchmark the verifier performance in Table 2 where we vary the number of proofs being verified (N) for a fixed circuit with 2^{12} constraints. We use the batch verification strategy as outlined in [Pol26] and report the *amortized* cost of verifying each proof together with the (factor×) speedup over verifying proofs individually.

Constraints	Prove
2^{10}	47.5
2^{12}	142.5
2^{14}	490.0
2^{16}	1,711.3
2^{18}	6,354.7
2^{20}	22,748.1

Table 1: Prover benchmarks. Timings in ms.

N	Time/N	Speedup
1	722.91	—
256	29.84	25.0×
4,096	16.10	46.3×
65,536	11.66	64.0×

Table 2: Batch-verifier benchmarks. Amortized time per operation in μs.

Timing Breakdown. We now recall the batch verification of Pari proofs [Pol26] and in Table 3, we provide a breakdown of where time is spent during batch verification. The k-th proof is $\left(T^{(k)}, U^{(k)}, v_a^{(k)}\right)$. Given a batch of N proofs, the verifier carries out the following steps:

1. *Challenge:* recompute each proof's challenge ζ_k and sample the random weights $r_k \leftarrow \mathbb{F}$.
2. *Evaluate:* evaluate the interpolated statement polynomials at ζ_k to obtain $v_R^{(k)}$.
3. *MSMs:* set $V^{(k)} := v_a^{(k)}\alpha G + v_R^{(k)}\beta G - \zeta_k U^{(k)}$ and compute
$$\widetilde{T} := \sum_k r_k\, T^{(k)}, \qquad \widetilde{U} := \sum_k r_k\, U^{(k)}, \qquad \widetilde{V} := \sum_k r_k\, V^{(k)},$$
one N-term MSM each, reported in one column each.
4. *Pairing:* accept if and only if
$$e\big(\widetilde{T},\, \delta H\big) \;=\; e\big(\widetilde{U},\, \tau H\big) \cdot e\big(\widetilde{V},\, H\big),$$
a product of three pairings, the batched form of the verification equation of Fig. 6.

In the implementation, we make two additional optimizations:

[1] https://github.com/guruvamsi-policharla/zk-pari/pull/2
[2] https://github.com/alireza-shirzad/garuda-pari

- First, challenge derivation is independent of the key size: the verifying key is absorbed into the Fiat-Shamir transcript once, when the key is built, so each of the N derivations clones that fixed-size transcript state and absorbs only its own proof's statement and commitment $T^{(k)}$.
- Second, the Evaluate phase batches its field inversions *across* proofs. Evaluating the statement polynomial at ζ_k requires inverting one Lagrange evaluation per statement element. Here, we use Montgomery's batch-inversion trick [Mon87].

Multi-Threaded Verification Throughput We next measure batch verification performance by increasing the number of cores. We naively shard the proofs across T threads, each batch-verifying its own chunk of $80{,}000$ proofs in an isolated single-threaded pool. Because the benchmarks were run on an M5 chip which has a mix of performance and efficiency cores, the scaling is not truly linear in Table 4, but nevertheless exceeds one million transactions per second on 18 threads, where a transaction refers to either a send or a receive.

Phase	Time (ms)
Challenge	35.6
Evaluate	24.9
$\widetilde{T}$ MSM	188.2
$\widetilde{U}$ MSM	177.1
$\widetilde{V}$ MSM	344.1
Pairing	0.59
Total	770.6

Table 3: Batch-verifier breakdown at $N = 65{,}536$.

Threads	Operations/s
1	86,929
4	324,857
8	559,399
16	992,728
18	1,051,487

Table 4: Multi-threaded verification throughput in operations (send or receive) per second.

Payment Circuits Next we benchmark Bonsai's relations $\mathcal{R}_{\mathsf{send}}$ and $\mathcal{R}_{\mathsf{recv}}$ of Section 6, along with the operation-hiding relation $\mathcal{R}_{\mathsf{op}}$ of Section 6.1. Table 5 reports constraint counts, prover timings on 1 and 8 threads, and single-threaded verification timings. We benchmark two instantiations of the hash function: Pedersen hashes over the Jubjub curve, whose security rests only on the discrete logarithm assumption, and Poseidon over the BLS12-381 scalar field with width 3, $\alpha = 5$, and 8 full and 57 partial rounds. Both the receipt MMR opening and the nullifier tree have depth 40.

$\mathcal{R}_{\mathsf{send}}$ is just three commitment openings and range checks: $16{,}109$ R1CS constraints under Pedersen and $1{,}647$ under Poseidon, proving in 0.84 s and 0.14 s respectively on a single thread. In $\mathcal{R}_{\mathsf{recv}}$, the depth-$40$ receipt opening (40 hashes) and the sparse Merkle tree insertion ($2\ell = 80$ hashes) account for 120 of its 123 hash evaluations, giving $398{,}111$ R1CS constraints under Pedersen and $30{,}729$ under Poseidon, which prove in 20.1 s and 1.6 s on a single thread, and in 3.8 s and 0.29 s on 8 threads. Across all six circuits the prover scales by 5.2–$5.8\times$ from 1 to 8 threads. The prover timings include witness generation but not constraint synthesis, which is performed once per circuit. $\mathcal{R}_{\mathsf{op}}$ folds both operations into a single circuit whose branch is selected by a witness bit, and costs only $\sim$8,300 (Pedersen) or $\sim$1,000 (Poseidon) R1CS constraints more than $\mathcal{R}_{\mathsf{recv}}$.

Notably this is far fewer constraints than the sum of the two circuits. A naive disjunction would synthesize both relations side by side. Instead, we observe that $\mathcal{R}_{\mathsf{send}}$ is structurally a sub-relation of $\mathcal{R}_{\mathsf{recv}}$ where both open two account commitments, hash one receipt, and perform the same range checks. The circuit therefore instantiates each of these shared gadgets *once* and lets the selector bit multiplex only the values fed into them. Since the proof system works over Square R1CS constraints, we use a standard compiler from R1CS $\rightarrow$ SR1CS at the cost of doubling the number of constraints. Proofs are $2\mathbb{G}_1 + \mathbb{F}$ (128 bytes) and verification requires three pairings, which takes roughly 0.8 milliseconds for every circuit in Table 5.

Circuit	Hash	R1CS	SR1CS	Prove (1 thread)	Prove (8 threads)	Verify (μs)	$\lvert\pi\rvert$
$\mathcal{R}_{\mathsf{send}}$	Pedersen	16,109	32,227	837.0	143.6	747.3	128
$\mathcal{R}_{\mathsf{recv}}$	Pedersen	398,111	796,231	20,072.1	3,822.5	753.7	128
$\mathcal{R}_{\mathsf{op}}$	Pedersen	406,460	812,931	20,395.8	3,844.3	805.1	128
$\mathcal{R}_{\mathsf{send}}$	Poseidon	1,647	3,303	136.7	26.2	805.5	128
$\mathcal{R}_{\mathsf{recv}}$	Poseidon	30,729	61,467	1,632.8	293.9	768.0	128
$\mathcal{R}_{\mathsf{op}}$	Poseidon	31,706	63,423	1,646.2	291.5	761.2	128

Table 5: Payment-circuit benchmarks. Prover timings in ms, verification is single-threaded. Proof sizes in bytes. Receipt MMR and nullifier tree both have depth 40.

References

[ac22] arkworks contributors. `arkworks` zksnark ecosystem. https://arkworks.rs, 2022. 29

[BAZB20] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In Joseph Bonneau and Nadia Heninger, editors, *FC 2020*, volume 12059 of *LNCS*, pages 423–443. Springer, Cham, February 2020. 1, 5

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014. 1, 2, 5

[BCG+20] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. ZEXE: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy*, pages 947–964. IEEE Computer Society Press, May 2020. 1

[BCMR24] Foteini Baldimtsi, Kostas Kryptos Chalkias, Varun Madathil, and Arnab Roy. SoK: Privacy-preserving transactions in blockchains. Cryptology ePrint Archive, Report 2024/1959, 2024. 1

[BLKZ25] Maxence Brugeres, Victor Languille, Petr Kuznetsov, and Hamza Zarfaoui. Fast, private and regulated payments in asynchronous networks. In *Advances in Financial Technologies*, volume 354 of *LIPIcs*, pages 3:1–3:24, 2025. 1

[BLMG21] Gabrielle Beck, Julia Len, Ian Miers, and Matthew Green. Fuzzy message detection. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1507–1528. ACM Press, November 2021. 2

[BM25] Sean Bowe and Ian Miers. A note on notes: Towards scalable anonymous payments via evolving nullifiers and oblivious synchronization. Cryptology ePrint Archive, Report 2025/2031, 2025. 6

[BPS19] Florian Bourse, David Pointcheval, and Olivier Sanders. Divisible E-cash from constrained pseudo-random functions. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 679–708. Springer, Cham, December 2019. 4

[Bra94] Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 302–318. Springer, Berlin, Heidelberg, August 1994. 4

[BSKD22] Mathieu Baudet, Alberto Sonnino, Mahimna Kelkar, and George Danezis. Zef: Low-latency, scalable, private payments. Cryptology ePrint Archive, Report 2022/083, 2022. 5

[CFN90] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 319–327. Springer, New York, August 1990. 4

[CFT98] Agnes Hui Chan, Yair Frankel, and Yiannis Tsiounis. Easy come - easy go divisible cash. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 561–575. Springer, Berlin, Heidelberg, May / June 1998. 4

[CGG+26] Arka Rai Choudhuri, Sanjam Garg, Matthew Gregoire, Keewoo Lee, Mike Lodder, Hart Montgomery, Guru Vamsi Policharla, and Jim Zhang. Qurrency: a quantum-secure, private, and auditable platform for digital assets. Cryptology ePrint Archive, Paper 2026/015, 2026. 5

[Cha82] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982. 1, 3, 4

[Cha83] David Chaum. Blind signature system. In David Chaum, editor, *CRYPTO'83*, page 153. Plenum Press, New York, USA, 1983. 4

[Cha90] David Chaum. Online cash checks. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 288–293. Springer, Berlin, Heidelberg, April 1990. 4

[CHA22] Matteo Campanelli and Mathias Hall-Andersen. Veksel: Simple, efficient, anonymous payments with large anonymity sets from well-studied assumptions. In Yuji Suga, Kouichi Sakurai, Xuhua Ding, and Kazue Sako, editors, *ASIACCS 22*, pages 652–666. ACM Press, May / June 2022. 6

[CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 302–321. Springer, Berlin, Heidelberg, May 2005. 4

[CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, Berlin, Heidelberg, August 1993. 4

[CPST15] Sébastien Canard, David Pointcheval, Olivier Sanders, and Jacques Traoré. Divisible E-cash made practical. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 77–100. Springer, Berlin, Heidelberg, March / April 2015. 4

[DGS+18] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *PoPETs*, 2018(3):164–180, July 2018. 6

[DMS24] Michel Dellepere, Pratyush Mishra, and Alireza Shirzad. Garuda and pari: Faster and smaller SNARKs via equifficient polynomial commitments. Cryptology ePrint Archive, Report 2024/1245, 2024. 4, 23

[DPP16] Rasmus Dahlberg, Tobias Pulls, and Roel Peeters. Efficient sparse Merkle trees: Caching strategies and secure (non-)membership proofs. In Billy Bob Brumley and Juha Röning, editors, *Secure IT Systems, NordSec 2016*, volume 10014 of *Lecture Notes in Computer Science*, pages 199–215. Springer, 2016. 3, 7, 8

[Eag25] Liam Eagen. Glock: Garbled locks for Bitcoin. Cryptology ePrint Archive, Report 2025/1485, 2025. 4, 23

[FMMO19] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 649–678. Springer, Cham, December 2019. 6

[GK15] Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 253–280. Springer, Berlin, Heidelberg, April 2015. 1, 5

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004. 6, 9

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Berlin, Heidelberg, May 2016. 4

[GWY+19] Zhangshuang Guan, Zhiguo Wan, Yang Yang, Yan Zhou, and Butian Huang. BlockMaze: An efficient privacy-preserving account-model blockchain based on zk-SNARKs. Cryptology ePrint Archive, Report 2019/1354, 2019. 6

[HBHW26] Daira-Emma Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. Version v2026.7.0-115-gad30e5, Network Upgrade 6.2, 2026. 1, 2, 4, 13

[KKS22] Aggelos Kiayias, Markulf Kohlweiss, and Amirreza Sarencheh. PEReDi: Privacy-enhanced, regulated and distributed central bank digital currencies. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1739–1752. ACM Press, November 2022. 5

[KPS26] Shubham Khurana, Sahadeo Padhye, and Rajeev Anand Sahu. Zero-knowledge extension of PARI. *IACR Communications in Cryptology*, 3(2), 2026. 23

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Berlin, Heidelberg, December 2010. 4

[LRR+19] Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. Omniring: Scaling private payments without trusted setup. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 31–48. ACM Press, November 2019. 1, 5

[LT22] Zeyu Liu and Eran Tromer. Oblivious message retrieval. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 753–783. Springer, Cham, August 2022. 2

[MGGR13] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed E-cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411. IEEE Computer Society Press, May 2013. 5

[Mon87] Peter L Montgomery. Speeding the pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987. 30

[Noe15] Shen Noether. Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098, 2015. 1, 5

[OO90] Tatsuaki Okamoto and Kazuo Ohta. Disposable zero-knowledge authentications and their applications to untraceable electronic cash. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 481–496. Springer, New York, August 1990. 4

[Pal24] Santiago Palladino. Global state epochs. Aztec forum post (design proposal), January 2024. https://forum.aztec.network/t/global-state-epochs/2704. 6

[Pen24] Penumbra Labs. The Penumbra protocol specification. Penumbra protocol documentation, 2024. https://protocol.penumbra.zone. 13

[Pol23] Polygon Miden. Miden state model. https://miden.xyz/blog/state, 2023. 6

[Pol26] Guru Vamsi Policharla. The proof is in the pairing. https://commonware.xyz/blogs/batch-pari, March 2026. Accessed August 13, 2026. 4, 23, 29

[PS16] David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 111–126. Springer, Cham, February / March 2016. 5

[PSS19] Alexey Pertsev, Roman Semenov, and Roman Storm. Tornado cash privacy solution. Whitepaper, version 1.4, 2019. https://github.com/tornadocash/tornado-core. 13

[SAB+19] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. In *NDSS 2019*. The Internet Society, February 2019. 5

[Sch98] Berry Schoenmakers. Basic security of the ecash payment system. In *State of the Art in Applied Cryptography, COSIC'97 Course*, volume 1528 of *Lecture Notes in Computer Science*. Springer, 1998. 6

[SKK24] Amirreza Sarencheh, Aggelos Kiayias, and Markulf Kohlweiss. PARScoin: A privacy-preserving, auditable, and regulation-friendly stablecoin. In Markulf Kohlweiss, Roberto Di Pietro, and Alastair R. Beresford, editors, *CANS 2024, Part I*, volume 14905 of *LNCS*, pages 289–313. Springer, Singapore, September 2024. 5

[SKKK25] Amirreza Sarencheh, Hamidreza Khoshakhlagh, Alireza Kavousi, and Aggelos Kiayias. DART: Decentralized, anonymous, and regulation-friendly tokenization. Cryptology ePrint Archive, Report 2025/239, 2025. 6

[ST99] Tomas Sander and Amnon Ta-Shma. Auditable, anonymous electronic cash. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 555–572. Springer, Berlin, Heidelberg, August 1999. 5

[TBA+22] Alin Tomescu, Adithya Bhat, Benny Applebaum, Ittai Abraham, Guy Gueta, Benny Pinkas, and Avishay Yanai. UTT: Decentralized ecash with accountable privacy. Cryptology ePrint Archive, Report 2022/452, 2022. 1, 5

[Tod12] Peter Todd. Merkle mountain ranges. OpenTimestamps documentation, 2012. https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md. 3, 8

[WKDC22] Karl Wüst, Kari Kostiainen, Noah Delius, and Srdjan Capkun. Platypus: A central bank digital currency with unlinkable transactions and privacy-preserving regulation. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2947–2960. ACM Press, November 2022. 5