Privacy with native AA

Private ownership needs a private path to paying for execution.

State

Authorization

Computation

Fees

From the account model to native AA

Four parts. The native-AA section uses the specifications' own frame and field names and ends on the 8130 / 8141 comparison.

An account is not its token balance

State, authorization and computation are distinct parts of the EVM model.

STATE

What the account holds

- `nonce`, ETH `balance`, `codeHash`, `storageRoot`
- USDC balances live in the USDC contract's storage

AUTHORIZATION

Who may act

- A signature authenticates the sender
- Contract rules authorize token spending: `transfer`, `allowance`

COMPUTATION

What gets executed

- The EVM runs the contract call
- Execution reads and updates state

> An ERC-20 balance is application state. ETH for gas is native account state, checked by the protocol before any contract code runs.

How a privacy pool works: Tornado Cash

Deposit and withdrawal are public. The proof says only: I control one unspent note in this tree.

ORIGINAL TORNADO CASH · ONE FIXED DENOMINATION PER POOL

1 · Alice deposits

Send fixed amount D to the pool
Publish C: a sealed note receipt
Alice, D and C are public

→

2 · Join many deposits

Public commitments: C_1 C_2 C_3 C_4 …
A Merkle tree summarizes them in root R
The contract holds the pooled funds

→

3 · Withdraw to Bob

Prove control of one deposited note
Bob receives D minus any relayer fee
Bob and the exit amount are public

UNLINKABLE: which deposit funded Bob's withdrawal?

The proof hides the chosen commitment — no deposit-to-withdrawal match is published.

Keep the witness private

Note secrets + selected Merkle path
These identify the deposit to the prover

→

Publish a ZK proof

"I know a note in this tree."
Root R + nullifier hash N are public

→

Check + record + pay

Verify proof; reject if N was used
Record N as spent; release the funds

Timing, address reuse and gas funding can still narrow the set of plausible deposits.

Same primitives. Three different proof statements.

Commitments, a Merkle tree, nullifiers and a ZK proof in all three. What each proof asserts is what each one hides.

	TORNADO CASH	0XBOW · PRIVACY POOLS	RAILGUN
The proof says	"I control an unspent deposit in this tree."	"…and its label is in the approved set."	"This private state transition is valid."
Privacy set	Every deposit in the pool	The ASP-approved subset of deposits	Every shielded balance and transfer
A spend leaves	A public withdrawal to a chosen address	A public withdrawal plus a change commitment	New private notes, or a public unshield
Who can exclude you	Nobody: the pool holds no policy	The ASP revokes the label; `ragequit` exits publicly	A Broadcaster may decline; self-signing stays open

All three hide which input funded the spend. Only RAILGUN also hides the output; only 0xbow lets a third party decide whose input qualifies.

Tornado withdrawal circuit · 0xbow ZK withdrawal · 0xbow ASP · 0xbow ragequit · RAILGUN privacy system · RAILGUN transactions · Privacy Pools origin spec

RAILGUN: private balances, public execution

Private balance → Relay Adapt → ordinary EVM calls → private balance. Ownership is hidden; the calls are not.

RAILGUN CROSS-CONTRACT CALL · ONE TRANSACTION, ONE BLOCK · ANY FAILED CALL REVERTS EVERYTHING

1 · Unshield

Out of the private balance:
`relayAdaptUnshieldERC20Amounts`
Unshielding costs a 0.25% fee
The tokens sit in Relay Adapt

2 · Multicall the public EVM

Ordered `{to, data, value}` calls
Approve first, then swap, deposit, mint
The called contracts see Relay Adapt
as the caller, not the `0zk` wallet

3 · Shield the result

Only `relayAdaptShieldERC20Addresses`
returns as a new private balance
Anything unlisted is unrecoverable
Private ownership resumes here

WHAT ONE SUCH TRANSACTION PUBLISHES

Stays private

Which `0zk` wallet spent, and its balance before and after
The private UTXO ownership graph
Sender, recipient, token and amount of a private transfer

Public on chain

Every contract called, its calldata and the route taken
The amounts crossing the boundary, in and out
A Broadcaster pays the gas and is the visible submitter

RAILGUN is not a private EVM: step 2 is ordinary public execution, wrapped in private ownership.

FHE hides values. Stealth addresses hide the recipient link.

The two techniques protect different parts of the same transfer.

ZAMA / ERC-7984 · FHE

Alice → Bob: encrypted amount

- Public token contract and public addresses
- Encrypted balances and arithmetic
- Decryption follows an access-control policy

ERC-5564 · STEALTH ADDRESS

Alice → fresh address B′

- Bob derives B′′s spending key from a stealth meta-address
- Public amount and token flow
- Funding B′ with ETH for gas can reveal Bob again

> Combine mechanisms when both the value and the identity link must be hidden.

The same account model applies at both levels

Account / state → auth → EVM. The pool's contract logic runs inside the protocol's execution step.

PROTOCOL LEVEL Ordinary EOA transaction · sender pays gas

STATE

Account

- Sender's `nonce` and ETH `balance`
- `codeHash` and `storageRoot`

→

AUTHORIZATION

Auth

- **Authorization** · sender signature
- **Nonce** · replay protection
- **Fee payer** · enough ETH for gas

→

COMPUTATION

EVM

- Run the pool's contract call
- Read and update public state

CONTRACT LEVEL Privacy pool · inside the EVM above

STATE

Pool account / storage

- Commitments and accepted roots
- Spent nullifier hashes

→

AUTHORIZATION

Pool auth

- Valid ZK proof against a known root
- Nullifier hash not already spent

→

COMPUTATION

EVM contract logic

- Mark the nullifier hash as spent
- Release funds to the recipient

A pool proof authorizes the withdrawal. Protocol auth still requires a valid nonce and an ETH-funded fee payer.

Zcash: the fee is a value-balance residual

The fee is the value a spend does not hand to an output. `ZIP-317` prices it by structure, not computation.

ALICE SPENDS ONE 10 ZEC NOTE AND SENDS BOB 3 ZEC · NO GAS METER · NO FEE FIELD

Where the fee comes from

Spent note	10.0000 ZEC	hidden
Bob's note	− 3.0000 ZEC	hidden
Change note	− 6.9999 ZEC	hidden
valueBalanceOrchard	**= 0.0001 ZEC**	public · the fee

Value commitments hide the amounts; only the balance is public.

How much · ZIP-317

```
conventional_fee =
5000 zatoshi × max(2, logical_actions)
```

One logical action per Orchard action

Two-action floor: 0.0001 ZEC

Not a consensus rule: wallet and node policy

The fee leaves the same shielded value the spend authorizes: no fee field, no gas limit, no funded fee account.

The fee amount is public. The note values behind it are not.

Aztec: many users, one public fee payer

Each user's credit is a private note inside the same contract. The chain records the contract paying, not the user.

PRIVATE CREDIT NOTES

One anonymity set

Derive the address from the class hash and a shared deployment salt, by convention `Fr.ZERO`

A private FPC of your own tags your users instead

WHAT AN OBSERVER SEES

SharedPrivateFPC paid

One address, shared by every app that uses it

WHAT STAYS HIDDEN

Whose note paid

Each balance

Which call ran

Private calls are told apart only by their footprint

Aztec: proving and spending that credit

`mint` proves the bridge claim against the nullifier tree. `pay_fee()` deducts from the note during setup.

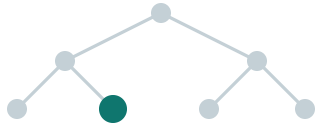
FUND · ONCE PER L1 DEPOSIT

L1 deposit

`to` = the FPC
`secretHash`
from the bridge salt

→

Claim nullifier



the claim

→

`mint`

Rebuild it
Assert it is in the tree
Credit the note

→

Alice's balance note

Only she can spend it

SPEND · EVERY TRANSACTION

Alice's balance note

Held in the FPC

→

`pay_fee()`

Runs in setup
– `max_gas_cost`
No refund

→

`set_as_fee_payer()`

The FPC is the payer

→

Protocol charges

the FPC, not Alice

Setup is non-revertible, so the fee is committed before the app phase runs and a reverting application still pays.

Starknet: a paymaster decouples payer from sender

AVNU's paymaster submits `apply_action` through its forwarder and pays the STRK gas. That hop is also the censorship point.

REVIEWED AVNU PRIVATE FLOW · `SPONSORED_PRIVATE` FEE MODE · API MARKED PREVIEW

Wallet proves the pool action client-side → Paymaster forwarder submits `apply_action` and pays gas → Pool fee reimburses the paymaster

Two costs: STRK gas to the network, paid by the paymaster; the pool fee, paid from the private balance in `poolFeeToken`.

THE DECOUPLING

No public account is linked to the private balance

- The fee payer is the paymaster; the actor is a shielded note
- No personal STRK top-up, and `apply_action` carries no user signature

THE COST

Inclusion runs through one permissioned service

- `x-paymaster-api-key` is required for `sponsored_private`
- Only a whitelisted pool; a blacklisted call returns `UNKNOWN_ERROR`
- No documented unsponsored path: `PRIVACY_REQUIRES_SPONSORING`

Three solution families

Who converts private value into a fee the protocol can collect?

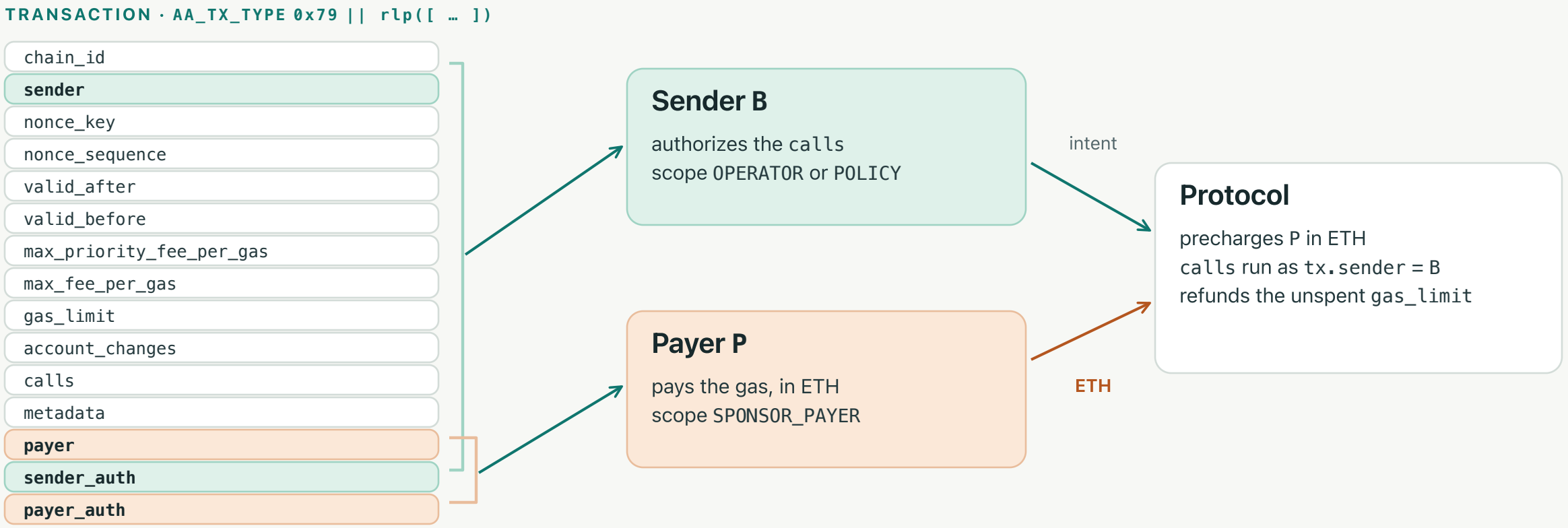
ARCHITECTURE	HOW IT PAYS	WHAT IT REQUIRES
Service fronts gas	User emits a private fee output; a relayer pays native gas	A submitting service and a reimbursement policy
Native contract vault	A proof authorizes a shared native reserve; the user settles privately	Admissible validation and a durable fee entitlement
Native shielded fee validity	The protocol accepts a proof of conserved private value that includes the fee	A shielded value / fee state transition in the protocol

> For Ethereum native AA, the contract vault is the incremental design target.

RAILGUN broadcasters · AVNU execute private tx · EIP-8141 · Zcash protocol

8130: one transaction, two actors

The same RLP list names the account that authorizes the calls and the account that pays the ETH.

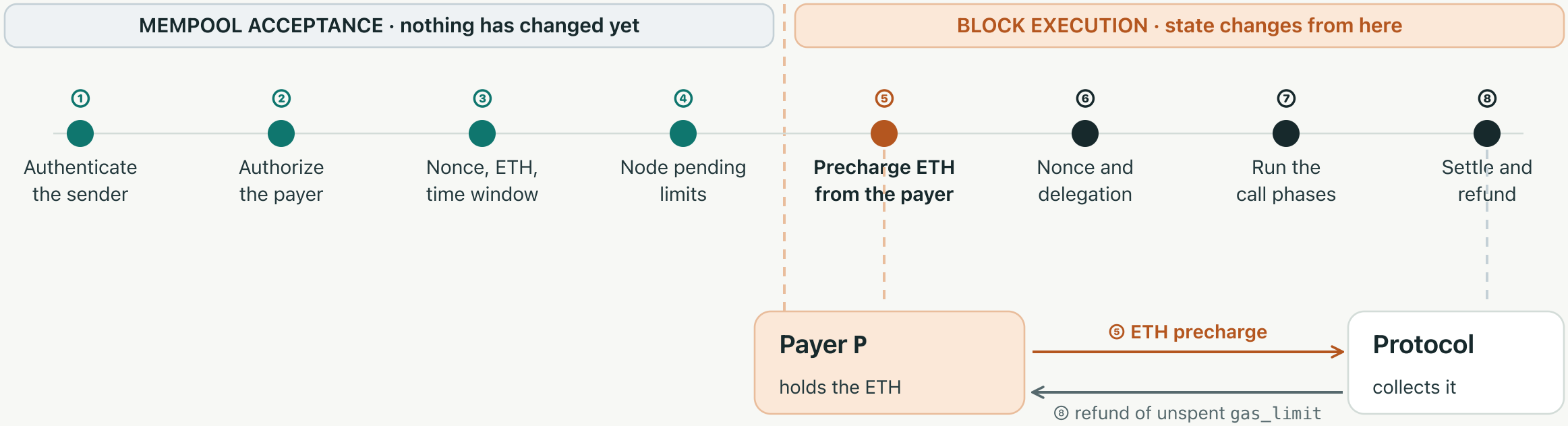


Self-pay leaves `payer` empty. A sponsor is a second named account: `payer` is the sender's commitment, `payer_auth` its authorization.

8130: nothing changes until the payer is charged

Four mempool checks, then four block steps. The ETH precharge opens execution, before any `calls` run.

ONE TRANSACTION · THE ORDER A NODE PROCESSES IT



At ⑤ the payer is charged `effective_gas_limit × max_fee_per_gas`, in ETH, before any call runs.

The `calls` that could release private value do not run until ⑦.

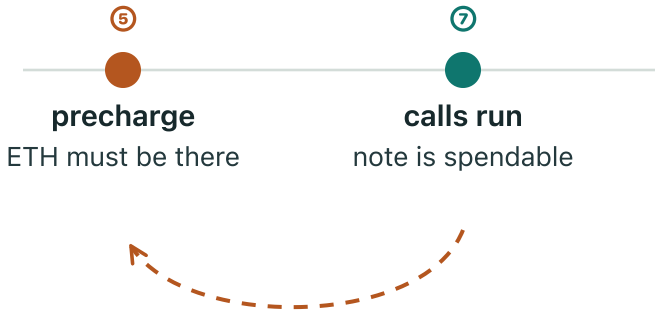
8130: where a private fee falls out

Three obstacles, each one a rule of the draft rather than an implementation gap.

THREE OBSTACLES · EACH ONE FROM THE DRAFT'S OWN RULES

Charged too early

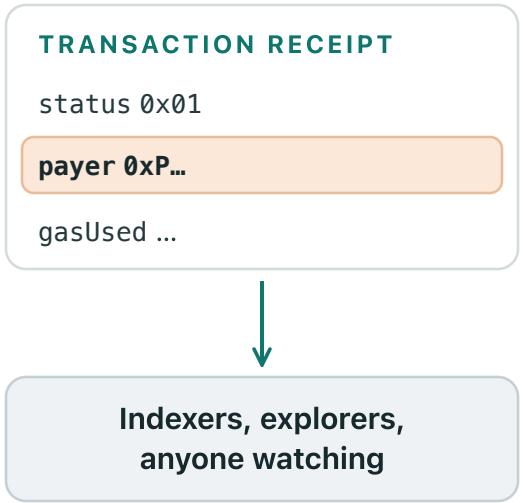
the fee is ETH, taken first



A shielded note is not a balance the protocol can debit

Charged in public

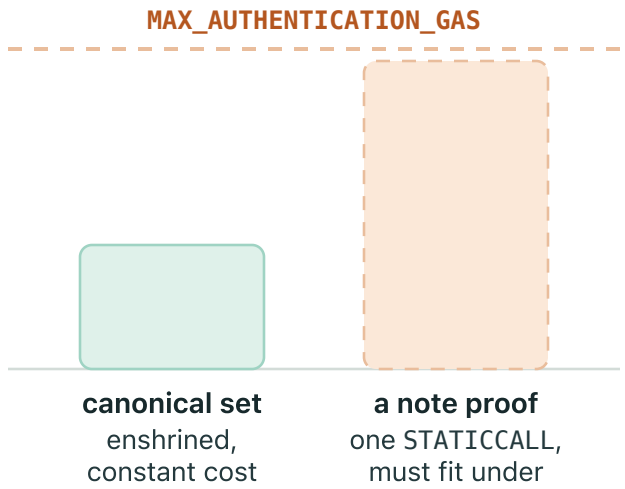
payer is a named account



One named payer per private spend, signed by the sender and logged

Proved off the path

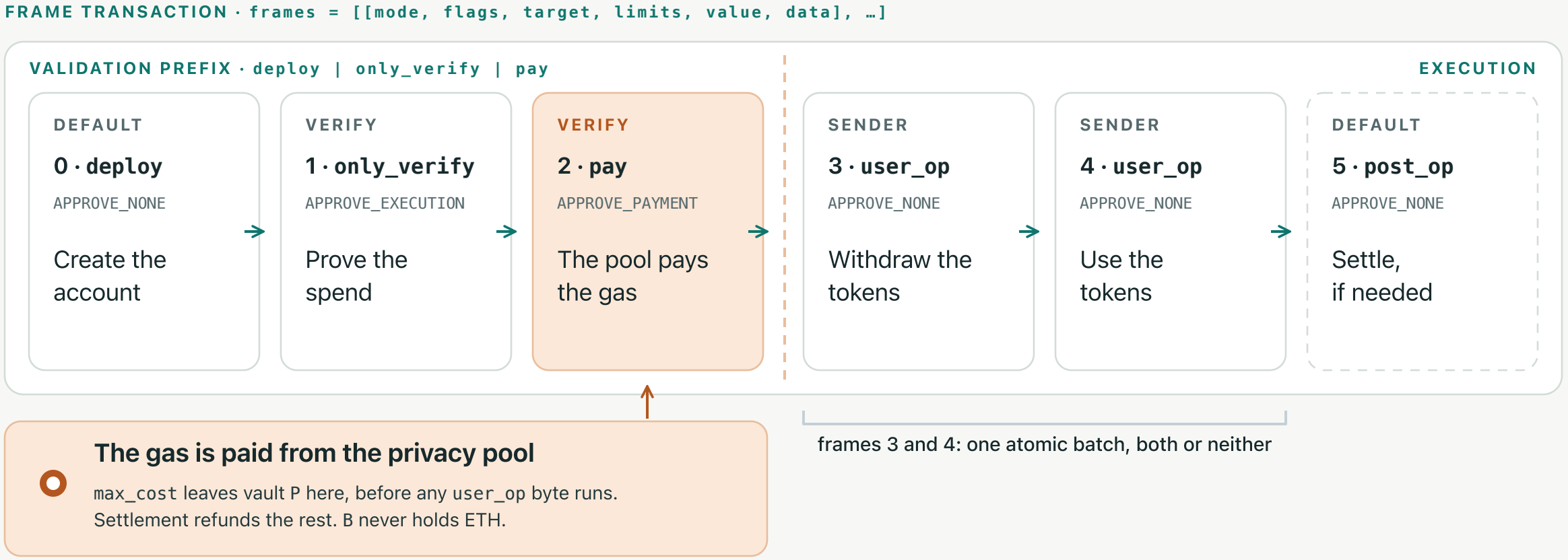
a note proof is not canonical



✕ canonical-only chains do not accept it on the 8130 path at all

8141: a brand-new account spends from the pool

Six frames, three modes. Frame 2 is where the privacy pool pays the gas, before anything executes.



An optional `expiry_verify` and 8272's `recent_root_verify` lead the prefix, before `deploy`.

8141: three mempool rules, two proposed fixes

Each rule blocks the private spend for a different reason. Two have draft EIPs, the third an unmerged PR — and none of them makes the note spendable exactly once.

PUBLIC-MEMPOOL POLICY, NOT CONSENSUS · EVERY RULE APPLIES ONLY UNTIL payer IS SET		PROPOSED FIX
1 **Storage outside `tx.sender`** `SLOAD` only at the sender, even through `CALL*` or `DELEGATECALL`	The pay frame cannot read P's own root or nullifiers — the state it exists to check. →	EIP-8272 · DRAFT ✓ **A canonical root ring** One narrow frame verifies it; the proof binds that tuple.
2 **Validation gas budget** `MAX_VERIFY_GAS` 100,000 gas `MAX_VERIFY_STATE_GAS` 500,000	A spend verifier must fit inside it, with the signature checks and 8272's own frame. →	EIP-8288 · DRAFT ✓ **The proof leaves the EVM** A leanSTARK dependency costs a fixed 30,000 gas.
3 **One pending spend per payer** `MAX_PENDING_TXS_USING_NON_CANONICAL_PAYMASTER = 1`	A proof-checking vault backs one pending spend pool-wide, and the canonical exception instead wants an `ecrecover` signer — a sponsor again. →	NOTHING MERGED ✗ **Still open** PR 12328 would admit payers by a decidable code property.

EVEN WITH ALL THREE FIXED · THE POOL STILL HAS TO SPEND ITS NOTE ONCE

A verified tuple proves only that a root was stored and is recent, never that the note is unspent.

`VERIFY` is a `STATICCALL`, so the nullifier `SSTORE` lands in a `SENDER` frame — after pay collected `max_cost`.

Rejection is not invalidity: private delivery changes the propagation and service-availability assumptions.

8272: the root arrives as frame data

Mempool rule 1 blocks the read. 8272 removes the need for it.

EIP-8272 · RECENT_ROOT_ADDRESS 0x…8272 · RECENT_ROOT_LENGTH 8192 · REQUIRES 7843 AND 8141

WITHOUT 8272

pay at P

checks the proof

×

P's storage

`roots[i]`

It has to read storage outside `tx.sender`

WITH 8272 · THE ROOT ARRIVES AS FRAME DATA

The pool publishes its root

CALL `0x…8272` with `salt || root`, kept for 8192 slots

↓

One canonical frame verifies it

`recent_root_verify`, with exactly two extra permissions

↓

The proof binds the verified tuple

handed forward as frame data · P's storage is never read

8288: make proof-heavy validation affordable

Aggregation is a cost mechanism; zero knowledge comes from the underlying proof.

PROPOSED AGGREGATION PATH

`DEP_VERIFY_FRAME_MODE` frame (mode 3) declares `(scheme, data_hash, verification_key_hash)` triples → Mempool wrapper per `AGGREGATION_INTERVAL`: direct proofs or one recursive STARK → Block validity: one recursive STARK proves every dependency

PRIVACY RELEVANCE

Amortize repeated verification

- Schemes: `LEANSPHINCS_SCHEME` signatures and `LEANSTARK_SCHEME` proofs against a shared verifier
- Aggregation lowers the per-transaction cost, not the disclosure

REMAINING WORK

Fit the whole validation prefix

- Root checks, proof checks, calldata and state must fit `MAX_VERIFY_GAS` = 100,000 and `MAX_VERIFY_STATE_GAS` = 500,000
- A cheap Merkle-proof estimate is not a ZK-verifier benchmark

EIP-8288 · EIP-8141 · Open PR 12309 · Open PR 12322

These drafts are a design space, not a ready stack

Resolve the conflicts before claiming a working public-mempool protocol.

ISSUE	WHY IT MATTERS	REQUIRED RESOLUTION
Frame mode 3 claimed three times	8288 `DEP_VERIFY_FRAME_MODE`, 8209 `COMMIT` and 7906 `POST_TX` cannot all be mode 3	Agree a registry / mode allocation (PR 12253)
Stale introspection fields	Proof and intent adapters may inspect different data	Align payloads and frame accessors (PRs 12309, 12322)
Failure semantics	7906 prose conflicts; repayment preservation is pending	Reconcile PR 12304 exemptions and PR 12266 approval effects
Unfinished admission details	8272 runtime (`RECENT_ROOT_CODE` TBD), sealed payer and proof budget unproven	Implement, measure and test the composed rules

8130 vs 8141: the privacy-relevant differences

Both separate sender and payer. The validation and payment interfaces differ.

CAPABILITY	8130 · KEYSTORE ACCOUNTS	8141 · FRAME TRANSACTION
Authenticate / authorize	`authenticate(hash, data) → actorId`; Keystore `scope` grants `OPERATOR` / `POLICY`	`only_verify` → `APPROVE(APPROVE_EXECUTION)`
Authorize gas payment	`payer` + `payer_auth`; `SPONSOR_PAYER`; ETH precharge	`pay` → `APPROVE(APPROVE_PAYMENT)`; collect `max_cost`
Execute the operation	`calls`: sequentially-committed atomic phases	`user_op` in `SENDER` mode; optional atomic batches
Custom private fee proof	Permissive acceptance policy: any authenticator via bounded `STATICCALL`; canonical-only path: no	Programmable `VERIFY`; public-mempool trace and payer rules still apply
Shared-root validation	No spec mechanism; a custom dependency policy is needed	8272 `recent_root_verify`: an explicit, narrow exception

8130: depends on the chain's authenticator acceptance policy (the L1 profile pairs with permissive). 8141: explicit frames and a clearer extension path, with fee safety and public-mempool admission unresolved.