Topi

Post-quantum private payments on Ethereum frame transactions

Whitepaper, draft • November 1, 2026

Abstract

Private payment systems on Ethereum rest on elliptic curves almost everywhere: pairing-based zero-knowledge proofs, elliptic-curve Diffie–Hellman in the packet formats of mixnets, and ECDSA for every account and relayer. A large quantum computer breaks all three, and the confidentiality of traffic recorded today can be broken after the fact. Topi is a private payment system in which no step on the path from deposit to settlement depends on discrete logarithms. A wallet proves a withdrawal from a shielded pool with a hash-based zero-knowledge proof and sends it through a three-hop mixnet whose packets are sealed per hop with X-Wing, a hybrid of ML-KEM-768 and X25519. Admission to the mixnet is rate limited by anonymous, hash-based epoch tickets. An exit node pairs spends and folds their proofs by recursion into a single proof that an on-chain WHIR verifier checks; the pool settles both spends in one EIP-8141 frame transaction authorized by leanSPHINCS, a hash-based signature scheme, so no ECDSA key touches the pool. A Lean 4 model of the pool's validation and settlement carries machine-checked theorems and is tested transaction by transaction against the deployed contracts. Topi runs end to end on a public test network: settles of two spends use about 16.3M gas, and a 1.3 MB proof crosses four cloud regions in about a minute over a mobile connection.

1 Introduction

Privacy on a public ledger is a long-lived promise. A payment that is private today must remain private for as long as anyone cares who paid whom, and the ledger keeps everything forever. This sets privacy systems apart from most other cryptographic deployments: an adversary who records data now and breaks the underlying assumption later still wins. For systems built on the hardness of the discrete logarithm, that later adversary is a cryptographically relevant quantum computer running Shor's algorithm [1].

Today's private payment designs for Ethereum depend on elliptic curves in three places:

- **Proofs.** Shielded pools typically use Groth16 [2] over the BN254 pairing curve. A quantum adversary who recovers the discrete logarithms in its structured reference string can forge proofs and drain the pool.
- **Transport.** Mixnets in the Sphinx [3] family derive per-hop keys by elliptic-curve Diffie–Hellman. Traffic recorded today can be unwrapped once those keys fall, revealing who sent which message.

- **Authorization.** Every deposit, relayer and fee payment is signed with ECDSA, whose keys a quantum adversary recovers from public keys exposed on chain.

Topi replaces each of these. Its contributions are:

1. a shielded pool that runs as an EIP-8141 [4] account, settles two spends per transaction with one hash-based recursive proof, and binds that proof to the spends' public fields on chain;
2. a KEM-based Sphinx mixnet with an X-Wing [5] encapsulation per hop, fixed 8 KB packets, batching, cover traffic and single-use reply blocks;
3. anonymous per-epoch rate limiting from Binius64 [6] zero-knowledge proofs over SHA-256, with no pairings and no trusted setup;
4. post-quantum authorization of every transaction on the pool path with leanSPHINCS signatures in EIP-8141 signature entries, for user accounts and for the paymaster that pays for settles;
5. a Lean 4 [7] model of the pool's validation and settlement with machine-checked safety theorems, checked against the chain by differential testing;
6. an implementation running end to end on a public test network, with measurements.

2 Background

Frame transactions. EIP-8141 introduces a transaction type that is a list of *frames* executed in order, each with a mode, a target, gas limits and data, plus a list of signature entries. A VERIFY frame calls an account's validation code, which approves the transaction with a scope: PAYMENT makes the account pay for gas, EXECUTION lets later SENDER frames act as the sender. Validation reads the transaction through new opcodes, so any account can define its own authorization. A signature entry of scheme ARBITRARY carries opaque bytes that the validating account interprets; the signature hash covers the whole transaction except the bytes of entries whose message is empty, so a witness in such an entry can sign the entire transaction, including the frame that verifies it.

Privacy Pools. The withdrawal relation of Privacy Pools [8] proves that a withdrawal spends a note committed in a pool tree, whose deposit label lies in an association tree, and reveals a nullifier that prevents a second spend. Topi uses this relation with a change note, so a withdrawal can take part of a note's value.

Hash-based proofs. Spartan [9] reduces circuit satisfiability to sum-checks; WHIR [10] is a hash-based polynomial commitment and proximity test with fast verification. Both rest only on collision-resistant hashing and are plausibly post-quantum. leanVM is a minimal zkVM built for recursively verifying such proofs.

Post-quantum primitives. ML-KEM [11] is the lattice-based key encapsulation standardized by NIST; X-Wing [5] combines ML-KEM-768 with X25519 so that the result is secure if either component is. SPHINCS+ [12, 13] is a stateless hash-based signature scheme; leanSPHINCS is a variant over Keccak with an on-chain verifier.

3 System overview

Topi has four parts: a wallet (the SDK and a client prover), a post-quantum mixnet of four roles, an exit prover that aggregates spends, and contracts on chain. Figure 1 shows the path of one spend. The mixnet carries only proofs and their public fields; the exit learns which spends it pairs, but not who sent them. The lifecycle of a private spend is:

1. **Deposit.** A post-quantum account calls `deposit` with value and a precommitment only it can open; the pool inserts the note.

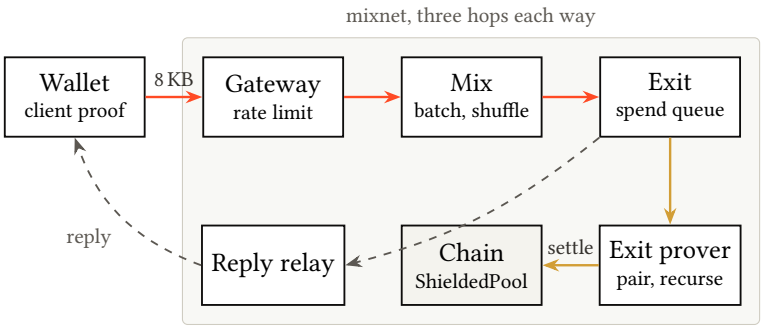


Figure 1. The path of a spend. Red: the forward route, on which a payment can still be traced to its sender's network address until it has crossed the mix. Amber: the exit's aggregation and the settle. Dashed: the reply on a single-use reply block.

2. **Prove.** The wallet rebuilds the pool's trees from its events and proves a withdrawal in zero knowledge: about a second and 450 MB.
3. **Send.** The 1.3 MB proof and its fields leave as about 280 packets of 8,192 bytes through gateway, mix and exit, admitted on one anonymous rate-limit token.
4. **Aggregate.** The exit checks each queued spend against the chain, pairs two, and folds both proofs by recursion into one terminal proof: 12 to 20 minutes and about 12 GB.
5. **Settle.** One frame transaction with the pool as sender verifies the terminal proof, binds it to both spends, consumes both nullifiers, pays the recipients and inserts the change notes.

4 The shielded pool

4.1 Fields and digests

All hashing is over the KoalaBear prime field, $p = 2^{31} - 2^{24} + 1 = 2{,}130{,}706{,}433$. A digest is eight field elements; on chain it is packed into one 256-bit word. Every element entering the contract is checked to be below p. We write $\mathsf{H}_D(x_1 \| \ldots \| x_n)$ for the Poseidon2 [14] sponge over KoalaBear with domain tag D.

4.2 Deposits

A deposit carries value $v < 2^{128}$ and a precommitment $pre = \mathsf{H}_{\text{PRECOMMIT}}(nullifier \| secret)$. The pool assigns the next index and computes

$$label = \mathsf{H}_{\text{LABEL}}(scope \| index),$$
$$note = \mathsf{H}_{\text{NOTE}}(pool \| label \| v \| pre),$$

inserting *note* (as a pair, so later paired inserts stay aligned) into the pool tree and *label* into the association tree. Both are depth-32 incremental Merkle trees; every root either tree has had is recorded, and a spend may prove against any recorded root. The deposit event carries every leaf, so a wallet can rebuild both trees from logs alone.

4.3 Spends

A spend is described by 82 public fields: pool and scope identity, a *context* that commits to the fee recipient, the recipient, the withdrawn amount, the fee, the pool and association roots, the nullifier hash and the change-note commitment. The client proof shows that these fields describe a valid withdrawal; its only public input is the transaction digest $\mathsf{H}_{\text{TX}}(fields)$.

Frame	Mode	Target	Scope	Effect
0	`VERIFY`	pool	execution	the pool's policy approves exactly this shape
1	`VERIFY`	paymaster	payment	checks the exit operator's leanSPHINCS signature, pays gas
2	`SENDER`	pool		`settle(proofCall, a, b)`: verification and state change

Table 1. The settle transaction. The pool never pays gas, and only `settle` ever runs as the pool.

4.4 Settlement

The exit prover submits one frame transaction whose sender is the pool (Table 1).
`settle` requires that it runs as the pool; forwards the proof to the terminal verifier and requires acceptance; recomputes the application root from both spends' fields and requires it to equal the root inside the proof calldata; checks that each spend names this pool and known roots and an unspent nullifier, and marks it spent; inserts both change notes; pays each recipient and the committed fee recipient; and emits `Settled`. Any failure reverts the `SENDER` frame, so no nullifier is consumed and no value moves.

4.5 Design decisions

- **Nullifiers in storage, not in nonce keys.** Were a nonce key required to equal a nullifier, an attacker could copy a pending spend's nullifier into a transaction with a bogus proof: validation would pass and consume the key, `settle` would revert, and the victim's note would become unspendable. With storage nullifiers a failed settle consumes nothing.
- **The fee goes to the committed address.** A front-runner who copies a valid proof can only execute the victim's own withdrawal; it cannot redirect the fee.
- **Fresh random nonce keys.** Each settle uses a random 128-bit nonce key, so spends are order-free and can be delayed by the mixnet without a shared nonce queue.
- **Heavy work in the `SENDER` frame.** Validation is capped at 500,000 gas, far below the cost of verifying a proof.

5 Proofs and recursion

Client proof. Each spend is proven with Spartan-WHIR in full zero knowledge over a circuit of 63,952 constraints for the Privacy Pools relation with change. Proving takes about a second and 450 MB on a laptop; the proof is about 1.3 MB.

Recursion. The exit runs leanVM over two client proofs: it verifies each inside the VM, lifts them, and runs a grouped root recursion. The output is a grouped Keccak WHIR terminal proof of about 100 KB, whose *application root* commits to both spends' statements.

Binding on chain. The generated terminal verifier returns only a boolean: some two proofs with statements hashing to the application root were accepted by the fixed programs. It does not see public inputs. The pool therefore recomputes, for each spend,

$$d = \mathsf{H}_{\mathrm{TX}}(\mathit{fields}_{0..81}),$$
$$s = \mathrm{P1}\big(\texttt{"leanvm-spartan-whir-statement-v1"},\, d\big),$$

where H is Poseidon2 and P1 a length-prefixed Poseidon1 digest, and the root $\mathrm{P1}(\textsc{nod}1, 2, s_A, s_B)$, and requires it to equal the proven application root. This binding is what ties an otherwise generic proof to these two spends. Terminal verification costs about 10.2M gas with its calldata; a full settle about 16.3M, of the 2^{24} per-transaction cap.

6 The mixnet

6.1 Packet format

Topi's packets are a KEM-based variant of Sphinx. Instead of one group element blinded per hop, each hop receives its own X-Wing ciphertext, so a packet stays confidential if either ML-KEM-768 or X25519 holds. From each hop's shared secret, BLAKE3 derives four keys: header MAC, header stream, payload stream and replay tag. The header uses the Sphinx filler construction, so its length is the same at every hop, and every node keeps the replay tags it has seen and drops repeats.

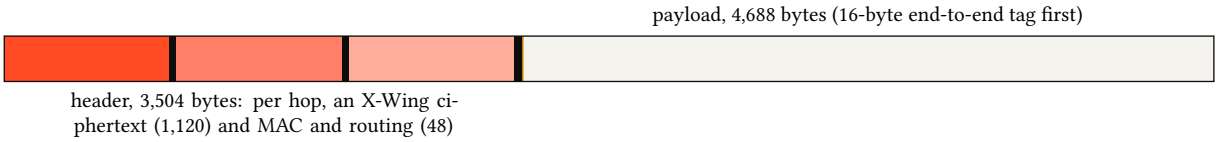


Figure 2. Every packet is 8,192 bytes at every hop.

6.2 Roles and routes

The forward route is always gateway, mix, exit. The gateway delays each packet by an exponentially distributed time (mean 100 ms), as in Loopix [15]; the mix ignores per-packet delay and instead collects packets for a batch interval (1.5 s by default) and releases each batch shuffled. The gateway also emits cover packets at exponential intervals (mean 700 ms), indistinguishable from real packets until the exit removes the last layer.

6.3 Messages and replies

A message larger than one payload is split into fragments; the first also carries a single-use reply block (SURB) for the route reply relay, mix, gateway, whose last routing entry names a random reply slot. The exit reassembles the message, queues the spend, and answers on the SURB; the reply waits in the gateway's reply store until the client polls for it. Clients never accept inbound connections, so they work behind NAT and on mobile networks.

7 Anonymous rate limiting

A gateway must refuse floods without learning who sends what. Each registered member may send $L = 16$ messages per epoch of 600 s. With member secret sk:

$$C = \mathrm{SHA256}(\texttt{"TPID"} \,\|\, sk), \qquad M = \mathrm{MerkleRoot}(\{C\}),$$
$$t_i = \mathrm{SHA256}(\texttt{"TPTK"} \,\|\, sk \,\|\, epoch \,\|\, i), \qquad R = \mathrm{MerkleRoot}(t_0, \ldots, t_{L-1}),$$

for $0 \le i < L$, where M is published. Once per epoch the client proves in zero knowledge that it knows sk and a path placing C under M, and that R is the token root for $(sk, epoch)$; only $(M, epoch, R)$ are public. The proof is a Binius64 proof wrapped in an outer zero-knowledge Spartan proof: 79 SHA-256 compressions, about 316 KiB, proven in about 0.2 s and verified in about 0.19 s. Each message then carries a token $(t_i, i,$ path to $R)$ of 164 bytes, and the gateway accepts each t_i once. Because R is a deterministic function of $(sk, epoch)$ and has only L leaves, over-use is impossible rather than punishable.

A spend is about 280 packets, so a message spends one token on its first packet and receives a budget of 512 continuation packets that carry only the 32-byte token value.

8 Post-quantum authorization

Every transaction on the pool path is authorized by leanSPHINCS: signatures of 5,684 bytes under a public key of two 32-byte words, checked by a shared verifier contract.

- **Accounts.** A user account is a two-frame transaction: a VERIFY frame on the account approving both execution and payment, and a SENDER frame with the call. One empty-message ARBITRARY entry holds the witness over the signature hash. Wallets deposit from such accounts and receive withdrawals to them.
- **The paymaster.** The post-quantum paymaster approves payment only for transactions of its pool that carry exactly one ARBITRARY entry accepted under the exit operator's key. Its VERIFY frame costs about 61,000 gas.

Account and paymaster validation must run opcodes that Solidity cannot emit, so both are written in Yul.

9 Formal verification

A Lean 4 model transcribes the pool account's and both paymasters' validation, and the settlement state machine. Among its machine-checked theorems:

- an approved pool transaction has exactly three frames, and only `settle` runs as the pool (`only_settle_runs_as_pool`);
- the pool never pays for its own transactions (`pool_never_pays`);
- no nullifier is spent twice over any history of transactions (`run_distinct`);
- a settle that fails changes nothing (`failed_settle_changes_nothing`);
- the pool's balance always equals the value of its live notes (`settle_preserves_inv`);
- with the post-quantum paymaster, no signature on the pool path is ECDSA (`pq_pool_tx_has_no_ecdsa`).

All proofs depend only on Lean's `propext` and `Quot.sound`; continuous integration fails if any depends on `sorryAx` or `Classical.choice`. The model abstracts the cryptography and assumes proof-system soundness; its correspondence to deployed bytecode is checked by differential tests against the chain: 528 of 528 cases for the pool policy with the classic paymaster, 90 of 90 with the post-quantum paymaster, 56 of 56 regression cases and 42 of 42 for the post-quantum account.

10 Security

Property	Against	Provided by
Deposit–withdrawal unlinkability	chain observers	zero-knowledge client proof; nullifiers reveal nothing about the note
Sender's network identity	the exit, the RPC, partial network observers	three hops, fixed-size packets, batching, cover traffic
Gas-payment unlinkability	chain observers	the pool is the sender; a paymaster pays
Pool funds	anyone submitting transactions	validation policy, statement binding, storage nullifiers, the Lean model
Long-term confidentiality and authorization	a future quantum adversary	X-Wing, hash-based proofs, leanSPHINCS

Table 2. What Topi protects, and how.

Trust. The exit operator is trusted for liveness, not safety: it can refuse or delay spends but cannot redirect funds, because recipients and the fee recipient are fixed by the proof and the binding check. The mixnet gives anonymity if at least one hop is honest and there is enough concurrent traffic for batches to mix. The paymaster risks only gas.

Limitations. Topi is research software on a test network and has not been audited.

- With one node per role and a fixed route, a single compromised mix or a global observer of all links defeats mixing; with little traffic, batches are small.
- The payload uses a per-hop stream layer rather than a wide-block cipher, so payload tagging is detected only end to end.
- A member's tokens within one epoch are linkable at the gateway, and continuation token values travel in the clear to it.
- A settle needs exactly two spends; the exit sees the public fields of the spends it pairs.
- Upstream reports that the recursion's full composition security is not yet established; the proof systems and leanSPHINCS are young implementations.
- Funding a post-quantum account, the fee recipient and the paymaster's admin withdrawal still use classical accounts, and Ethereum's consensus is not post-quantum.

11 Evaluation

Measurements come from a 16-thread laptop and from transactions on the chain 8141 test network.

Measurement	Value
Packet construction / processing per hop	0.4 ms / 0.18 ms
Ping round trip across four AWS regions	3.8 to 10 s
1.3 MB spend over a mobile uplink, queued at the exit	68 to 78 s
Epoch ticket: prove / verify	0.2 s / 0.19 s
Client proof: prove / peak memory	about 1 s / 450 MB
Exit recursion of two proofs	721 to 1,129 s, 12.4 GB
Poseidon2-24 permutation on chain	68,600 gas
Terminal proof verification with calldata	10.2M gas
Deposit from a post-quantum account	7.5M gas
Settle of two spends	16.3M gas

Table 3. Costs of the main operations.

Two spends sent through the four-region mixnet, signed with leanSPHINCS and paid by the post-quantum paymaster, settled in blocks 557,893 and 558,272 at 16.32M and 16.33M gas; a further settle followed in block 612,907.

12 Related work

Topi's pool follows Privacy Pools [8], replacing its pairing-based proofs with Spartan-WHIR and leanVM recursion. HOPR is a production mixnet using elliptic-curve Sphinx packets and proof-of-relay incentives; Topi's per-hop X-Wing encapsulation makes the transport post-quantum, and it admits traffic by zero-knowledge rate limiting instead of payment. Loopix [15] introduced the Poisson mixing and cover traffic Topi's gateway and mix follow. EthLib builds verified Lean implementations of Ethereum primitives, including a stateful hash-based signature; Topi's Lean work instead models a deployed contract system's validation and state machine. PQ1 signs with SPHINCS+ and validates through an ERC-4337 account; Topi takes the same approach on frame transactions, where the signature covers the whole transaction, and applies it to the paymaster as well as to user accounts.

13 Future work

- **More spends per settle.** Moving per-spend checks into the recursive proof and folding spends through an aggregation tree would let one terminal proof cover many spends, amortizing its 10M-

gas verification. EIP-8288, under discussion, could let a transaction depend on a proof verified once by the protocol, removing most of that cost.

- **Private transfers.** Note-to-note transfers to another person, not only withdrawals.
- **A wider network.** Several nodes per role with route selection, incentives, link encryption between nodes and key rotation.
- **Association sets.** An association-set provider, so pools can exclude deposits by policy.
- **Verified bytecode.** A proof, rather than tests, that the deployed bytecode refines the Lean model.

14 Conclusion

Topi shows that a private payment system on Ethereum can avoid discrete logarithms on every step from deposit to settlement today: hash-based proofs and recursion for the pool, lattice-based encapsulation for the transport, hash-based tickets for admission and hash-based signatures for authorization, with the pool's rules checked by machine. Its costs (a minute in the mixnet, minutes of aggregation, 16.3M gas per two spends) are high but already practical on a test network, and the paths to lowering them are clear.

References

[1] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. *FOCS*, 1994.

[2] J. Groth. On the size of pairing-based non-interactive arguments. *EUROCRYPT*, 2016.

[3] G. Danezis and I. Goldberg. Sphinx: a compact and provably secure mix format. *IEEE S&P*, 2009.

[4] EIP-8141: Frame transactions. https://eips.ethereum.org/EIPS/eip-8141.

[5] M. Barbosa, D. Connolly, J. D. Duarte, A. Kaiser, P. Schwabe, K. Varner and B. Westerbaan. X-Wing: the hybrid KEM you've been looking for. *IACR Communications in Cryptology*, 2024.

[6] B. E. Diamond and J. Posen. Succinct arguments over towers of binary fields. *EUROCRYPT*, 2025.

[7] L. de Moura and S. Ullrich. The Lean 4 theorem prover and programming language. *CADE*, 2021.

[8] V. Buterin, J. Illum, M. Nadler, F. Schär and A. Soleimani. Blockchain privacy and regulatory compliance: towards a practical equilibrium. *Blockchain: Research and Applications*, 2023.

[9] S. Setty. Spartan: efficient and general-purpose zkSNARKs without trusted setup. *CRYPTO*, 2020.

[10] G. Arnon, A. Chiesa, G. Fenzi and E. Yogev. WHIR: Reed–Solomon proximity testing with super-fast verification. *EUROCRYPT*, 2025.

[11] NIST. FIPS 203: Module-lattice-based key-encapsulation mechanism standard. 2024.

[12] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld and P. Schwabe. The SPHINCS+ signature framework. *ACM CCS*, 2019.

[13] NIST. FIPS 205: Stateless hash-based digital signature standard. 2024.

[14] L. Grassi, D. Khovratovich and M. Schofnegger. Poseidon2: a faster version of the Poseidon hash function. *AFRICACRYPT*, 2023.

[15] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser and G. Danezis. The Loopix anonymity system. *USENIX Security*, 2017.