Privashh

A Compliance-Compatible Privacy L3 on Base:
Shielded UTXO Transfers and Unlockable Privacy Pools over a Shared zk Core

Privashh

@privashhgg · shh.gg

Whitepaper v1.0 · May 2026

Abstract

Public blockchains expose the full transaction graph: every transfer is linkable, and balances are perpetual. Privacy protocols that hide this graph have historically faced a regulatory dead-end, because indiscriminate anonymity shields illicit and legitimate flows alike. **Privashh** resolves this tension at the chain layer. It is a privacy *Layer-3* settling to Base on the OP Stack, built around a single zero-knowledge core — Poseidon commitments in a fixed-depth Merkle tree, proved with Groth16 over the BN254 curve — exposed through two interchangeable profiles. *Profile A* is a full privacy chain in which every balance is a shielded UTXO note and every transfer is a join-split proof. *Profile B* is a transparent EVM L3 with an opt-in, fixed-denomination *Privacy Pool* whose withdrawals additionally prove membership in an Association Set: a deposit can exit privately only if an Association Set Provider includes it, making compliance an *unlockable* property rather than a backdoor. We give the exact commitment, nullifier, and circuit constructions; the on-chain double-spend, root-history, and anti-front-running mechanisms; a bidirectional shielded bridge that delivers bridged funds directly as spendable notes; and a threat model with mitigations. The cryptographic core is implemented and test-verified end to end; the surrounding rollup and application stack are turn-key for local deployment and target Base Sepolia and mainnet.

Keywords: zero-knowledge proofs, zk-SNARK, Groth16, Poseidon, Privacy Pools, association sets, shielded UTXO, OP Stack, Layer-3, compliance.

1. Introduction

Every transaction on a public ledger is, by construction, a public record. Amounts, counterparties, and timing are visible to anyone, and they remain visible forever. For individuals this erodes financial privacy; for businesses it leaks payroll, supplier relationships, and treasury strategy. The cryptographic tools to fix this — commitment schemes, nullifiers, and succinct zero-knowledge proofs — have been production-ready for years, yet the deployments that used them most aggressively collapsed under regulatory pressure, because an anonymity set that admits everyone unconditionally also launders everyone's proceeds unconditionally.

Privashh starts from the position that privacy and compliance are not opposites but a design problem. The relevant question is not *whether* a protocol hides the transaction graph, but *who*, if anyone, can scope the set of funds that are allowed to move through the shield. We adopt the Privacy Pools equilibrium of Buterin et al. [5]: withdrawals prove membership not only in the set of all deposits, but also in an *Association Set* published by an Association Set Provider (ASP). Honest users prove they are inside a set that excludes known-tainted deposits; the protocol never learns *which* deposit is being spent. Exclusion from the association set does not seize funds — it simply makes them non-withdrawable through the private path. Privacy is the default; compliance is unlockable.

We deliver this as a Layer-3 rather than a single contract for three reasons. A dedicated rollup gives Privashh its own block space and fee market, so privacy traffic does not compete with unrelated L2 congestion. Settling to Base on the OP Stack inherits Base's security and, transitively, Ethereum's finality, without re-implementing a consensus or data-availability layer. And owning the execution layer lets us ship privacy-native predeploys and genesis configuration that a tenant contract on a shared chain cannot.

1.1 Contributions

- **A dual-profile architecture over one zk core.** A full-privacy UTXO chain (Profile A) and a transparent L3 with an opt-in, compliance-compatible Privacy Pool (Profile B) reuse the same field, hash, Merkle tree, circuits, and verifiers; a profile is selected by a single deployment flag.
- **A precise, implementation-faithful specification.** We fix every primitive — field element, Poseidon arity, Merkle depth, nothing-up-my-sleeve zero leaf, and the exact ordering of every public signal — so that circuits, contracts, and client SDK are provably in agreement.
- **A bidirectional shielded bridge.** Funds locked on Base arrive on Privashh directly as a spendable shielded note, not a transparent balance; withdrawals burn a note under proof and unlock on Base through the canonical OP Stack bridge.
- **A concrete threat model and verification gate.** Each protection — double-spend, forged membership, value inflation, front-running, non-compliant exit, bridge spoofing — maps to a specific mechanism and a passing test.

1.2 Document structure

Section 2 reviews the cryptographic preliminaries. Section 3 presents the system architecture and the rationale for an L3 on Base. Section 4 specifies the shared cryptographic core, the two profiles, and the shielded bridge. Section 5 describes the on-chain protocol mechanics. Section 6 analyses the compliance equilibrium. Section 7 is the threat model. Section 8 covers implementation and verification, Section 9 the roadmap, Section 10 related work, and Section 11 concludes.

2. Preliminaries

2.1 Field and hash function

All commitments, nullifiers, and Merkle nodes are elements of the BN254 scalar field F_p, where

$$p = 21888242871839275222246405745257275088548364400416034343698204186575808495617.$$

We hash with **Poseidon** [3], an algebraic hash purpose-built for arithmetic circuits: it costs a small number of constraints per permutation, which makes Merkle proofs and commitment openings cheap to prove. We write $H(a, b, \ldots)$ for Poseidon over the listed inputs, with the arity fixed per call site. The on-chain hasher is the EVM contract emitted by circomlib's Poseidon generator, and it is byte-for-byte equal to the hash used inside the circuits and the client SDK — a cross-component invariant we treat as security-critical (§4.6).

2.2 Incremental Merkle tree

State is accumulated in a binary Merkle tree of fixed depth $LEVELS = 20$, holding up to $2^{20} = 1\,048\,576$ leaves. Each internal node is $H(\text{left}, \text{right})$ over its two children, ordered by the corresponding path-index bit. The empty-subtree leaf is a nothing-up-my-sleeve constant,

$$ZERO = keccak256(\text{“shh”}) \bmod p,$$

so no party can claim a structurally special pre-image for the empty tree. The on-chain tree is *incremental*: it stores only the current frontier and a ring buffer of recent roots, so insertions are O(LEVELS) and a proof may be verified against any recent root, not merely the latest one (§5.2).

2.3 Groth16 succinct proofs

Circuits are written in Circom and proved with **Groth16** [2] over BN254. Groth16 yields a constant-size proof (three group elements) and constant-time verification — a fixed, inexpensive pairing check independent of circuit size — which is exactly what an on-chain verifier needs. The cost is a per-circuit trusted setup: a Powers-of-Tau phase shared across circuits, then a circuit-specific phase that emits a proving key, a verifying key, and a generated Solidity verifier. Setup secrecy is a soundness assumption; we address it with a multi-party ceremony for production (§9) and flag the development setup as non-production throughout.

3. System Architecture

Privashh is an OP Stack rollup that settles to Base, which in turn settles to Ethereum. Figure 1 shows the layering. Running as an L3 lets Privashh inherit Base's security and Ethereum's finality while keeping its own block space, gas policy, and privacy-native predeploys. Locally the chain runs as a single-sequencer devnet under Docker Compose; the same images deploy against Base Sepolia or mainnet by swapping the settlement endpoint and keys.

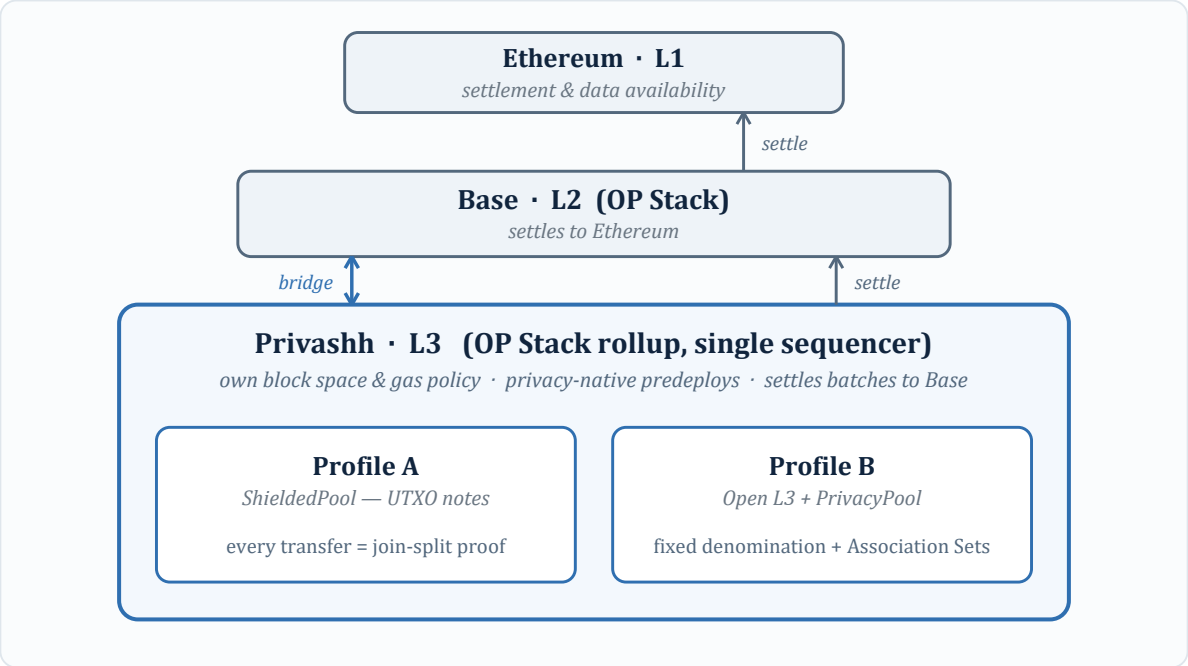

Figure 1. *Privashh as a Layer-3 on Base. The same OP Stack base supports both privacy profiles; they differ only in which contract is the default value path and in the genesis predeploys.*

3.1 Two profiles, one core

Both profiles share the field, Poseidon hasher, Merkle tree, circuits, and verifier infrastructure. They differ in which contract is the canonical value layer.

- **Profile A — full privacy chain.** The *ShieldedPool* is the canonical value layer. Balances exist only as UTXO note commitments in the Poseidon tree, and every transfer is a join-split proof. There are no transparent transfers in the default user experience.
- **Profile B — open L3 + Privacy Pool.** A normal transparent EVM L3 on which privacy is opt-in through the fixed-denomination *PrivacyPool*. Withdrawals require an Association Set membership proof, so an ASP can scope which deposits may exit — compliance-compatible privacy.

3.2 Component map

Layer	Component	Technology
Proofs	UTXO join-split & Privacy-Pool withdraw circuits	Circom + Groth16 (snarkjs)
Contracts	Pools, Poseidon Merkle tree, bridges, verifiers	Solidity (Hardhat)
Client	Notes, Merkle tree, witness & proof generation	TypeScript (isomorphic SDK)
Chain	op-geth / op-node / op-batcher / op-proposer	OP Stack (Docker Compose)
Indexer	Block explorer with privacy-aware views	Blockscout
App	Deposit / shielded transfer / withdraw	Next.js + Web-Worker proving

Table 1. *The Privashh stack. The proof, contract, and client layers are bound by the shared cryptographic specification of §4.*

4. Cryptographic Construction

This section is the cryptographic source of truth. Where circuits, contracts, and SDK touch the same value, the definition here is canonical and the three implementations must agree.

4.1 Profile B: the fixed-denomination Privacy Pool

A pool is parameterised by one *denomination* (e.g. 0.1 ETH); every deposit is exactly that amount, which concentrates the anonymity set on a single value. A depositor samples two secrets, a *nullifier* and a *secret*, both uniform in F_p, and publishes the commitment

$$commitment = H(nullifier, secret).$$

On withdrawal the spender reveals only nullifierHash = H(nullifier), which the contract records to prevent a second spend, and proves — in zero knowledge — that the hidden commitment lies in two trees at once:

- the **state tree**, whose leaves are all deposited commitments in insertion order (the deposit exists); and
- the **association tree**, whose leaves are the subset of commitments an ASP has marked compliant (the deposit is approved to exit).

Both trees use the same leaf value, so a single pair of secrets proves membership in both. The withdraw circuit `pool/poolWithdraw.circom` takes the following public signals, in this exact order:

```
[ stateRoot, associationRoot, nullifierHash,
  recipient, relayer, fee, refund ]
```

together with the private witness (nullifier, secret) and a Merkle authentication path into each tree. It enforces five constraints:

- **C1.** commitment = H(nullifier, secret).
- **C2.** nullifierHash ≡ H(nullifier).
- **C3.** the state-tree Merkle proof of commitment reproduces *stateRoot*.
- **C4.** the association-tree Merkle proof of commitment reproduces *associationRoot*.
- **C5.** recipient, relayer, fee, and refund are bound into the statement (each squared), so a captured proof cannot be re-aimed at a different recipient or fee — the standard anti-front-running technique.

Figure 2 traces the flow: dual-tree membership of one hidden leaf produces a proof whose public signals the contract checks before paying out.

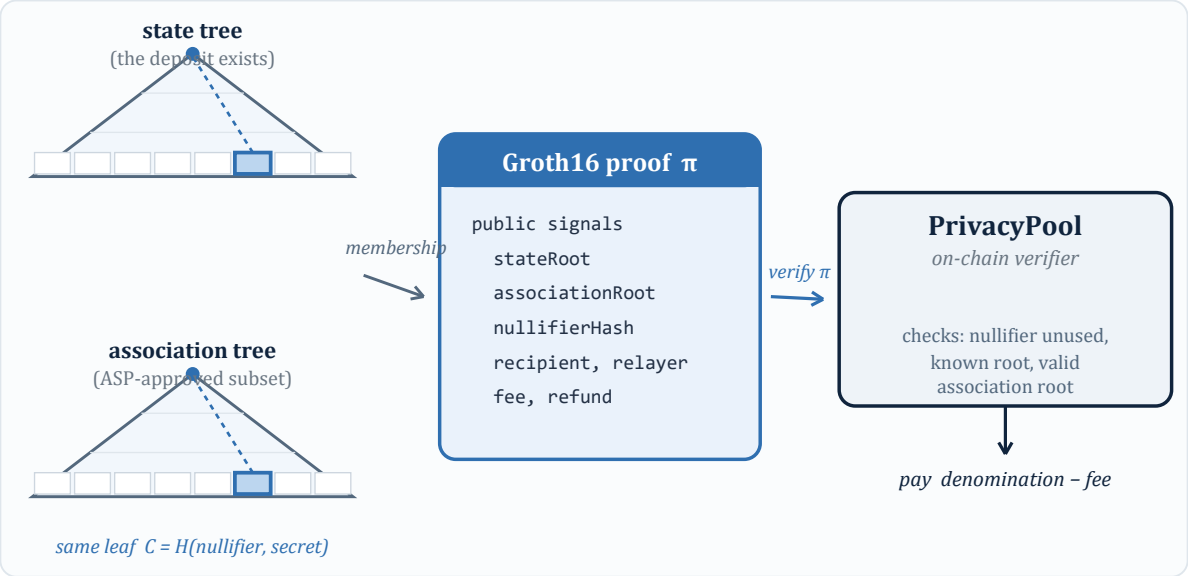

Figure 2. *Privacy Pool withdrawal. The same commitment leaf is proved present in both the state and association trees; the Groth16 proof reveals only the public signals, and the contract pays out only after checking the nullifier is unused and both roots are valid.*

4.2 Profile A: the shielded UTXO pool

Profile A follows the Tornado Nova [4] UTXO model, supporting arbitrary amounts and join-split transfers rather than a single fixed denomination. A spending key is a random privKey $\in F_p$ with public key pubKey = H(privKey). A note binds an amount, an owner, and a per-note blinding factor:

$$note = H(amount, pubKey, blinding).$$

To spend a note the owner derives, over its Merkle path indices,

$$signature = H(privKey, note, pathIndices), \quad nullifier = H(note, pathIndices, signature),$$

so the nullifier is unforgeable without the spending key and unique per note position. Each amount is range-checked to 248 bits to prevent field overflow when amounts are summed. The transaction circuit `shielded/transaction.circom`, instantiated as 2-input / 2-output, exposes

```
[ root, publicAmount, extDataHash,
  inputNullifier[nIns], outputCommitment[nOuts] ]
```

and enforces, for each input: recomputation of note, signature, and nullifier with nullifier ≡ inputNullifier[i]; a Merkle proof to *root* for every non-zero input (zero-value inputs are dummies that pad to the fixed arity and skip the root check); for each output, recomputation of its commitment and the 248-bit range check; pairwise-distinct input nullifiers (no in-transaction double spend); and, centrally, **value conservation**:

$$\Sigma_i\, inAmount_i + publicAmount \equiv \Sigma_j\, outAmount_j \pmod p.$$

The signed quantity *publicAmount* is computed on-chain as (extAmount − fee) mod p and selects the transaction's external meaning:

- publicAmount > 0 — a **deposit** (the caller sends value in);
- publicAmount < 0 — a **withdrawal** (the contract pays value out);
- publicAmount = 0 — an **internal private transfer**.

Finally *extDataHash* is bound into the proof; it commits to the recipient, relayer, fee, and the encrypted output ciphertexts the contract emits for note discovery, so none of these can be altered after proving.

4.3 Note and value summary

Object	Definition	Profile
commitment	`H(nullifier, secret)`	B
nullifierHash	`H(nullifier)`	B
pubKey	`H(privKey)`	A
note	`H(amount, pubKey, blinding)`	A
signature	`H(privKey, note, pathIndices)`	A
nullifier	`H(note, pathIndices, signature)`	A

Table 2. *Commitment and nullifier constructions. All inputs and outputs are elements of F_p; H is Poseidon with the arity implied by its argument count.*

4.4 The shielded bridge (Base ↔ Privashh)

Bridging in is designed so that value arrives *already shielded*. Locking the denomination or amount on Base emits a message through the OP Stack messenger; on Privashh the `L2ShieldedBridge` inserts the resulting commitment straight into the shielded set, so bridged funds appear as a spendable note rather than a transparent balance and never touch a public address on the L3. Bridging out reverses this: a shielded burn proof on Privashh authorises an unlock on Base, settled through the canonical L2 standard bridge for the authenticated L2→L1 transfer.

4.5 Cross-component invariants

Because the same values are recomputed in Circom, in Solidity, and in TypeScript, a divergence in any of them is a soundness or liveness bug. We therefore treat the following as a fixed checklist enforced across the three implementations:

- Poseidon arity and constants are identical in the circuit and the deployed hasher.
- *LEVELS*, the *ZERO* leaf, and node ordering are identical in circuit, on-chain tree, and SDK.
- Public-signal ordering is identical in circuit, exported verifier, and contract call sites.
- The 248-bit amount width is enforced in-circuit and respected by the SDK.

5. On-chain Protocol Mechanics

5.1 Deposit and withdrawal lifecycle

A Privacy Pool deposit sends exactly the denomination and inserts the caller's commitment into the state tree, emitting its leaf index. A withdrawal submits a Groth16 proof with the public signals of §4.1. Before paying out, the contract checks that the nullifier hash is unused, that the supplied state root is a known recent root, that the association root is one the ASP currently endorses, and that the attached refund matches; only then does it verify the proof, mark the nullifier spent, and pay denomination − fee to the recipient and fee to the relayer. Effects precede interactions — the nullifier is recorded before any value moves — and all payouts use checked calls, which together close the reentrancy surface.

5.2 Nullifier set and root history

Double-spend protection is a persistent on-chain set of spent nullifier hashes; any reuse reverts. Because proofs are generated against a Merkle root that may be one or more blocks old by the time they land, the tree keeps a bounded ring buffer of historical roots and accepts a proof against any root still in the buffer. This decouples proving from the exact head of the chain without ever accepting a root the contract did not itself produce.

5.3 Relayers

A fresh shielded recipient holds no gas, so withdrawals may be submitted by a relayer that is paid the in-proof fee. The relayer is untrusted for safety: the recipient, relayer, and fee are bound into the proven statement, so a relayer can neither redirect funds nor inflate its fee — it can only decline to serve, and a user can always self-submit. The reference wallet backend exposes a gasless withdrawal endpoint that performs exactly this relay.

6. The Compliance Equilibrium

Profile B's defining property is that withdrawal *requires* membership in an ASP-published association root (constraint C4). The ASP never learns which deposit a user is spending — it only curates the public set of deposits it is willing to vouch for. This yields the practical equilibrium of Buterin et al. [5]:

- Honest users prove membership in an association set that *excludes* known-illicit deposits, credibly dissociating themselves from them while revealing nothing about their own transaction.
- A tainted deposit excluded from every association root remains in the pool but is *non-withdrawable through the private path* — it is *unlockable* rather than confiscated.
- There is no global de-anonymisation key and no privileged party that can link a deposit to a withdrawal. Compliance is enforced at the boundary of who may exit, not by breaking privacy.

Multiple ASPs with different inclusion policies can coexist over the same pool; users choose which association root to prove against, and verifiers choose which roots to honour. The protocol fixes the mechanism, not the policy.

7. Threat Model

7.1 Assets, goals, and trust assumptions

Privashh protects three assets: **unlinkability** between a deposit and its withdrawal (and between note owners and transfers); **fund integrity** — no inflation, theft, or double spend; and **compliance optionality** — an ASP can scope exits without learning more than the public deposit set. We assume Groth16 soundness under a non-backdoored trusted setup (single contributor in development, a multi-party ceremony in production); the security of Poseidon and BN254, with the on-chain hasher equal to the circuit's; and that the Base settlement layer and its OP Stack bridge behave to specification.

7.2 Threats and mitigations

Threat	Mitigation
Double spend	Per-spend nullifier hash recorded on-chain; any reuse reverts.
Forged membership	Withdrawals verify a Groth16 Merkle-inclusion proof against a known root; only current/recent roots in the ring buffer are accepted.
Value inflation (shielded)	Circuit enforces Σ in + publicAmount ≡ Σ out (mod p); outputs range-checked to 248 bits.
Front-running / proof theft	Circuit binds recipient, relayer, fee, refund (Pool) and extDataHash (Shielded); a stolen proof cannot be re-aimed.
Non-compliant exit	Withdrawal requires membership in an ASP association root; excluded deposits are non-withdrawable through the private path.
Bridge spoofing (in)	The L2 shielded bridge finalises only messages from the aliased L1 bridge address (OP deposit aliasing).
Bridge spoofing (out)	Withdrawal spends via a proof bound to the bridge as recipient, then uses the canonical L2 standard bridge for the authenticated L2→L1 transfer.
Reentrancy on payout	Effects-before-interactions: the nullifier is marked spent before transfers; payouts use checked calls.
Relayer censorship	Relayer is untrusted for safety (proof binds recipient/fee); it can refuse service but not steal. Users can self-submit.
Anonymity-set erosion	Fixed denominations and shared trees concentrate the set; the client discourages amount/timing fingerprints.

Table 3. *Threats and their mitigations. Each row corresponds to a mechanism specified in §4–§5 and exercised by the verification gate of §8.*

7.3 Known limitations

We state the boundaries plainly. The development trusted setup is single-contributor and **must not** secure real funds; production requires the ceremony of §9. The circuits and contracts are **unaudited**, and fuzz and invariant testing are still to be added. Encrypted-note discovery via ECIES ciphertexts is currently stubbed pending the frontend. And **metadata privacy** — IP addresses, transaction timing — is out of scope at the protocol layer; it must be addressed by relayers and client behaviour. Privashh hides the on-chain graph, not the network layer beneath it.

8. Implementation and Verification

Privashh is a pnpm monorepo. The cryptographic core — circuits, contracts, and the isomorphic SDK — is implemented and verified end to end; the surrounding rollup, explorer, and application stack are scaffolded and turn-key for local deployment. The SDK is split into a browser-safe entry (Poseidon via poseidon-lite, randomness via Web Crypto) and a Node proving entry, so the same note and proof code runs in a browser Web Worker and on a server.

Correctness is gated by an automated suite that runs in continuous integration on every change: it installs Circom, compiles the circuits, runs the Groth16 trusted setup, and then executes the three test layers below before building the web app.

Suite	Coverage	Gate
SDK	Poseidon / Merkle / notes, value conservation, zero leaf	9 / 9
Circuits	Valid witness proves & verifies; tampered witness fails	4 / 4
Contracts	Deposit→prove→withdraw (both pools), double-spend, ASP gating, bidirectional bridge, front-running	10 / 10

Table 4. *The verification gate. Every threat mitigation in Table 3 has a corresponding passing test.*

The local stack is a single command: it boots a node, deploys the privacy core, copies the circuit artifacts into the web app, and runs the wallet backend, after which a deposit can be made and indexed against the running chain. The OP Stack devnet (op-geth, op-node, op-batcher, op-proposer) and a Blockscout explorer are wired under Docker Compose and validated; booting L3 blocks and binding the live OP portal address are the remaining integration steps.

9. Roadmap to Mainnet

The path to production is deliberately gated; each step has a concrete exit criterion.

- **Testnet.** Deploy verifiers, pools, and the bridge to Base Sepolia behind a public single-sequencer devnet with a hosted explorer; an external wallet completes the full deposit–transfer–withdraw flow.
- **Trusted-setup ceremony.** Replace the development keys with a multi-party Powers-of-Tau contribution and publish the transcript, so no single party knows the setup secret.
- **Audit and hardening.** External audit of circuits and contracts, plus fuzz and invariant tests, with all findings resolved before mainnet.
- **Mainnet.** Deploy behind a timelock-governed proxy with an emergency pause; verifiers and Merkle parameters are immutable per deployment.

10. Related Work

Fixed-denomination mixers. Tornado Cash [6] established the commitment/nullifier withdrawal pattern over a Merkle tree that Profile B builds on, but offered unconditional anonymity with no compliance affordance. **Shielded UTXO pools.** Tornado Nova [4] generalised that pattern to arbitrary amounts and join-split transfers; Profile A adopts its note, signature, and nullifier structure. **Compliance equilibria.** Privacy Pools [5] introduced association sets as a way to reconcile privacy with regulation; Privashh implements this as constraint C4 and makes it a first-class profile.

Anonymous payment systems. Zerocash and Zcash [7] pioneered note commitments and nullifiers for a full shielded currency; our Profile A occupies the same design space with circuit-friendly Poseidon hashing and an OP Stack settlement model. **Succinct proofs and circuit hashing.** Groth16 [2] and Poseidon [3] are the proving and hashing primitives throughout. Relative to all of these, Privashh's contribution is integrative: one zk core, two deployable privacy profiles, an unlockable compliance path, and a shielded bridge, delivered as a Base-settled L3.

11. Conclusion

Privashh treats privacy and compliance as a single design problem rather than a trade-off. By building two privacy profiles — a full shielded-UTXO chain and an opt-in Privacy Pool with association sets — over one Poseidon/Groth16 core, and settling the whole thing to Base as an OP Stack L3, it offers strong on-chain unlinkability while leaving exit-eligibility scopable by an Association Set Provider that never sees who is spending. The cryptographic core is specified exactly and verified end to end; the remaining work is the rollup integration, a multi-party trusted-setup ceremony, and an external audit before any funds are at stake. Privacy should be the default, and compliance should be unlockable — not the other way around.

12. References

[1] Privashh. *Privashh — Privacy Design, Architecture, and Threat Model.* Project documentation, 2026. shh.gg.

[2] J. Groth. “On the Size of Pairing-Based Non-Interactive Arguments.” *EUROCRYPT*, 2016.

[3] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger. “Poseidon: A New Hash Function for Zero-Knowledge Proof Systems.” *USENIX Security*, 2021.

[4] Tornado Cash. “Tornado Cash Nova: Arbitrary-Amount Shielded Transfers.” Protocol documentation, 2021.

[5] V. Buterin, J. Illum, M. Nadler, F. Schär, and A. Soleimani. “Blockchain Privacy and Regulatory Compliance: Towards a Practical Equilibrium.” 2023.

[6] A. Pertsev, R. Semenov, and R. Storm. “Tornado Cash Privacy Solution.” Whitepaper, 2019.

[7] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. “Zerocash: Decentralized Anonymous Payments from Bitcoin.” *IEEE S&P*, 2014. See also the Zcash Protocol Specification (Sapling).

[8] Optimism Collective. “OP Stack Specification.” Technical documentation, 2023–2025.

Disclaimer. This document describes a research and engineering effort under active development. The development trusted setup is single-contributor and must not secure real funds; the protocol is unaudited. Nothing herein is financial, legal, or investment advice.