Haze Whitepaper

A Lightweight Mimblewimble-Based Layer 1 for Confidential Value Transfer

Version 2.0
July 2026

Pranav Jha
pranav@hazeprotocol.xyz

Abstract

We present Haze, a Layer 1 blockchain implementing the Mimblewimble protocol, providing confidential transaction amounts and a bounded, non-growing chain state without reliance on a virtual machine or smart contract execution layer. Haze combines Pedersen commitments, Bulletproofs range proofs, horizon-based cut-through compaction, and an aggregate, kernel-offset validation scheme that allows a freshly-syncing node to safely verify chain history through a peer that has already pruned. A dynamic, size-based fee model, a Dandelion++ transaction relay layer, and an on-chain naming registry with deterministic key recovery round out a wallet-to-network stack intended for real use, not only demonstration. This document describes Haze's cryptographic foundations, consensus and validation model, fee mechanism, naming layer, token issuance schedule, and current status, including the limitations that remain as of this writing — principally around infrastructure maturity and independent security review, rather than protocol design. Haze remains scoped narrowly and deliberately as a settlement layer for private value transfer.

1 Introduction

Public blockchains built on the unspent transaction output (UTXO) model, beginning with Bitcoin, expose the amount transferred in every transaction to any observer of the chain. This transparency is a deliberate design choice that enables trivial auditability but forecloses on financial privacy as a default property. Individuals seeking to transact without exposing their full financial history to counterparties, employers, or anyone capable of reading a public ledger currently have limited options: heavyweight privacy chains with growing storage requirements, or custodial intermediaries that reintroduce the trust assumptions blockchains were designed to remove.

Several systems have addressed transactional privacy through different mechanisms. Monero combines ring signatures with stealth addresses. Zcash uses zero-knowledge succinct non-interactive arguments of knowledge. Mimblewimble, first described pseudonymously in 2016 and implemented by Grin and Beam, combines Pedersen commitments with an interactive transaction-construction protocol. Each carries distinct tradeoffs. Ring-signature schemes grow transaction size with anonymity set size. Historical zero-knowledge proof systems required trusted setup procedures. Mimblewimble's tradeoff is structural: because transactions are constructed interactively between sender and receiver, and because the protocol has no persistent address concept, the chain can aggressively prune historical transaction data through a process called cut-through, without weakening the validity guarantees of the remaining state.

What is needed is a payments-focused chain that takes Mimblewimble's pruning property seriously enough to keep home-node operation viable indefinitely, paired with a usability layer that addresses Mimblewimble's historical weakness: the absence of a persistent, human-readable way to receive funds. We propose Haze, a Mimblewimble Layer 1 with an on-chain naming registry, a functioning multi-platform wallet stack, and tested cut-through compaction.

2 Architecture Overview

Haze separates concerns into a small number of components: a validation and consensus core, a storage layer built on an embedded key-value store, a naming registry committed into block state, and a wallet layer that constructs and signs transactions client-side. There is no virtual machine and no contract execution layer; the protocol surface is deliberately limited to transaction validation, block production, chain-state compaction, and name registration.

Layer	Component	Function
Core	Validation & Consensus	Transaction and block validation, proof-of-stake block proposal, chain-state application, aggregate pruned-history validation
Storage	Chain State Store	Persistent embedded storage (sled) for blocks, UTXO set, and compaction metadata
Registry	Naming Registry	On-chain name-to-key mapping, first-come-first-served, permanent
Network	P2P Sync & Relay	Peer discovery, block/chain synchronization, and Dandelion++ transaction relay between nodes
Wallet	CLI, WASM, Mobile Core	Key management, deterministic recovery, name-based send/receive, transaction construction, client-side

Table 1: Haze's protocol layers

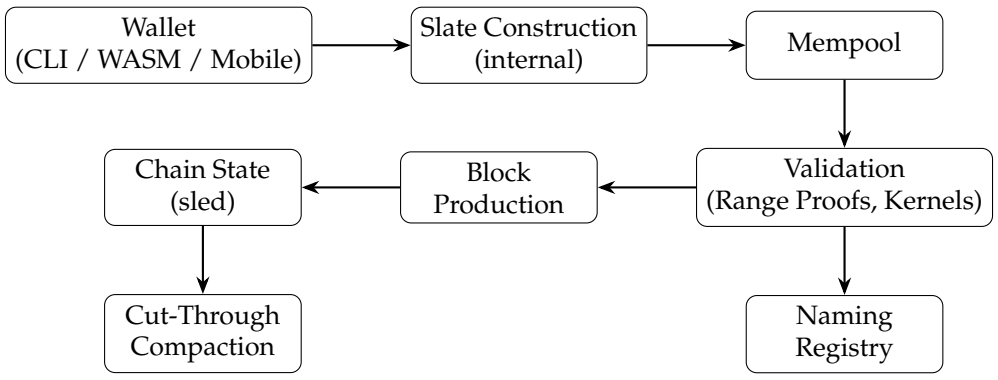

Figure 1: Haze system architecture

3 Cryptographic Foundations

3.1 Pedersen Commitments

Every output on Haze commits to a value v using a Pedersen commitment

$$C = vH + rG \tag{1}$$

where G and H are independent generator points on the Ristretto group (built on curve25519), v is the output's value, and r is a blinding factor known only to the output's owner. The

commitment is binding, hiding, and additively homomorphic: commitments can be summed, allowing the protocol to verify that total inputs equal total outputs plus fees without revealing any individual value.

3.2 Kernel Signatures

Each transaction produces a kernel: a Schnorr-style signature over the excess blinding factor, proving that the transaction's blinding factors sum to a value the signer knows, and therefore that the transaction is well-formed. Kernels are the only transaction artifact Haze retains permanently regardless of pruning; they constitute the durable proof that a given amount of value was validly transferred, independent of whether the specific inputs and outputs involved remain in storage.

3.3 Range Proofs

A Pedersen commitment alone does not prevent a party from constructing a transaction using a value that, modulo the underlying field, appears to balance the transaction equation while allowing value to be created from nothing. Haze addresses this using Bulletproofs, a non-interactive zero-knowledge range proof system requiring no trusted setup. Every output carries a range proof attesting that its committed value lies within a valid range. Verification is unconditional: it is the first validation step applied to every output on every block application, including coinbase outputs, with no code path in the current implementation permitting an output to be accepted without a passing check.

4 Transaction Construction

Because Mimblewimble transactions have no persistent address concept, constructing a transaction fundamentally requires interaction between sender and receiver: both parties must contribute a blinding factor before a transaction is valid. Haze resolves this at the protocol level using the slate construction described below, and resolves it at the usability level by exposing that interaction to users exclusively through the naming registry (Section 6), rather than through manual data exchange.

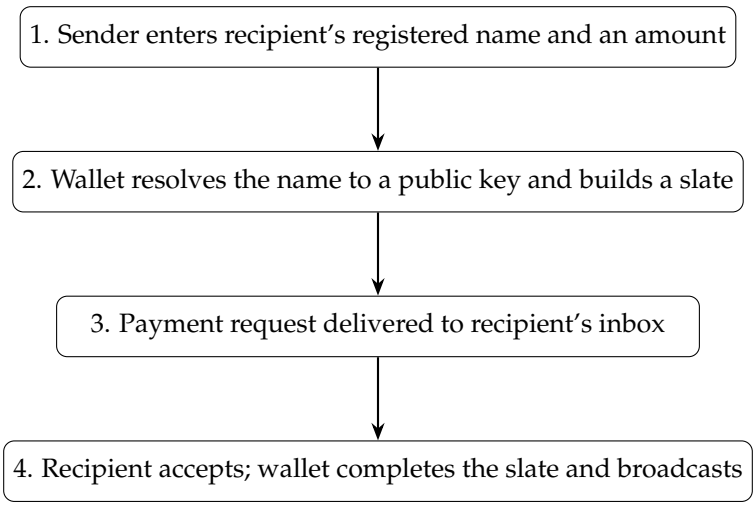

Figure 2: The name-to-name send flow

From a user's perspective, sending funds resembles sending to an account on a conventional chain: a name is entered, an amount is specified, and the transaction completes once the recipient accepts. The interactive slate protocol that makes this cryptographically valid — each party independently contributing a blinding factor such that the transaction's balance equation holds without either party learning the other's raw values — is handled entirely by the wallet client and is never exposed to the user as a manual step; there is no slate file, QR code, or other artifact for a user to copy or paste at any point in the send or receive flow. A user's receive screen displays their registered name directly, and that name is the only information anyone needs in order to pay them.

5 Cut-Through and Bounded Chain State

Cut-through is the mechanism by which Haze avoids unbounded chain growth. If an output is created in one block and later fully spent by an input in a subsequent block, and both events lie beyond a defined horizon from the current chain tip, the create/spend pair can be removed from a node's permanent storage without altering the validity of any block. This is possible because the kernel already constitutes permanent, sufficient proof that the transaction was valid.

Formally, for a chain of height h and a configured horizon H, any output created at height i and spent at height j, where $j \leq h - H$, is eligible for pruning. The current implementation applies this per-node, as a periodic background compaction process, and preserves two invariants unconditionally: the current unspent output set, and all kernels, regardless of age.

5.1 Safety Property

Block header hashing in Haze is defined over a fixed set of fields — height, previous block hash, kernel offset, timestamp, validator commitment and signature, and the name registry root — and does not include the transaction body's input or output vectors. Consequently, removing specific input/output entries from an already-applied block's stored body cannot alter that block's hash, any descendant block's hash, the recorded chain tip, or the current UTXO set. Compaction is defined as a pure function operating on chain state, which allows its correctness to be verified directly: a compacted chain and an uncompacted copy of the same chain must produce an identical tip hash, identical UTXO set, and identical kernel count. This property has been verified by direct test.

5.2 Aggregate Validation for Pruned History

A node that prunes its own storage presents a problem for any new peer attempting to sync from genesis through it: per-block balance validation re-checks each block's input/output sum at apply time, and a pruned block is, by construction, missing the specific values that check requires. Haze resolves this the same way Grin does: rather than re-verifying each pruned block individually, a syncing node instead verifies an aggregate balance equation over the pruned range, relying only on data that survives pruning unconditionally — kernels and the current UTXO set.

Concretely, for any range of blocks, the sum of currently-unspent output commitments plus the sum of kernel excess values over that range must equal the sum of all coinbase and reward outputs issued over that same range. Because Pedersen commitments are additively homomorphic, this equation can be verified without access to any individual pruned input or

output. Blocks within the cut-through horizon, which retain full bodies, continue to be validated per-block as before; blocks beyond the horizon are validated via this aggregate check. The peer-to-peer sync protocol serves the data this check requires — the full kernel list and current UTXO set — rather than declining to serve pruned ranges, as an earlier implementation did. This has been verified by test: a freshly-instantiated node syncing entirely through an already-pruned peer arrives at an identical tip hash and UTXO set to a node that synced before pruning occurred.

6 Naming Registry

Mimblewimble's lack of a persistent address concept is a meaningful usability cost. Haze addresses this with an on-chain naming registry, structurally closer to NEAR Protocol's account model than to a resolver contract such as ENS, adapted for a UTXO-based chain with no smart contract layer.

6.1 Registration Rules

- Registration is first-come, first-served; no auction mechanism is used.
- Ownership is permanent; there is no expiry or renewal requirement.
- A registration fee, denominated in HAZE, deters mass pre-registration.
- Names are restricted to lowercase alphanumeric characters plus hyphen and underscore, subject to length constraints and a reserved-word denylist enforced at the protocol level.

A registered name resolves to a public key, used by the sending wallet to construct the underlying transaction described in Section 4. The registry's state is committed into each block via a dedicated root, separate from the transaction commitment structure, since name ownership is intentionally a public record rather than a confidential one.

6.2 Onboarding

A new user's first interaction with Haze proceeds as a single linear sequence, rather than as separately-discovered steps:

1. A wallet keystore is generated client-side, and the corresponding BIP39 recovery phrase is displayed to the user.
2. The user chooses and registers a name before any funds exist in the wallet.
3. Because a brand-new wallet holds no balance, the registration fee for this initial name is sponsored by the network, drawn from the treasury allocation described in Section 9.2, rather than paid by the user. This is the only case in which registration is fee-free; subsequent name changes or additional registrations follow the standard fee schedule.
4. The user is given full access to the wallet, with a registered, receivable name already in place.

This sequence is intended to remove the two failure points most likely to strand a new user on an interactive, addressless chain: leaving a wallet without any way to receive funds, and requiring a funded balance before a receiving identity can exist. Registration outside of onboarding, including changing a name or a second wallet registering an additional name, requires the standard fee to be paid from an existing balance.

6.3 Receiving

A wallet's registered name is displayed directly on its receive screen. No further action, key exchange, or address construction is required of the recipient; sharing the name is sufficient for a sender to initiate payment. This is the mechanism by which Haze delivers the name-to-name send flow described in Section 4 without exposing any interactive protocol detail to either party.

6.4 Key Recovery

A wallet's blinding factors are derived deterministically from its BIP39 seed rather than generated independently per output, so that a wallet holding only its recovery phrase — with no other local state — can regenerate every key it has ever used, rescan the chain, and reconstruct its correct balance. Recovery additionally covers ownership of registered names: rather than requiring name ownership to be independently re-derivable, a restored wallet performs an owner-keyed lookup against the naming registry to recover which names it holds, closing what would otherwise be a gap between key recovery and full account recovery. This has been verified by test: a wallet's full balance, together with its registered name, is reconstructed identically in a completely fresh wallet instance restored from phrase alone.

7 Fee Model

Transaction fees on Haze are not gas in the Ethereum sense — there is no computation to meter, since Haze has no virtual machine. Instead, fees follow Monero's model: a fee proportional to transaction size in bytes, reflecting the block space a transaction consumes rather than any computation it performs. A minimum fee floor prevents trivially small transactions from being priced near zero, which would otherwise permit low-cost mempool spam. This replaced an earlier flat per-transaction fee used during initial development, and validation logic across all transaction types, including name registry operations, was updated uniformly to the size-based calculation.

8 Consensus and Network

Haze uses a proof-of-stake model in which a node registers as a validator by staking a confirmed output: revealing that output's blinding factor to the node constitutes proof of ownership without spending the output. Block proposal rights are tied to possession of the staked output's private key, held by the node actually producing blocks, distinct from simply appearing in the validator registry.

At the network layer, submitted transactions are relayed using a Dandelion++-style stem-and-fluff propagation scheme before broadcast, obscuring the network-layer origin of a transaction from simple graph analysis of message propagation. Multi-node peer-to-peer synchronization

has been verified across independently-run node instances, including recovery after a temporary disconnection and a late-joining node's full sync from genesis, both prior to and following cut-through compaction having occurred on the peer it synced through.

9 Token Issuance

9.1 Supply and Emission

Haze targets a maximum supply of 21,000,000,000 HAZE. This figure, rather than a Bitcoin-parity 21,000,000, reflects the protocol's lack of decimal subdivision: at Haze's target block interval, a literal 21,000,000 cap combined with a multi-year integer halving schedule would drive the per-block reward to zero well before the schedule's intended horizon.

Block rewards follow an integer halving schedule: an initial reward of 540 HAZE per block, halving every 12,600,000 blocks, corresponding to an approximately four-year halving period at Haze's target block interval. The reward at halving epoch n is

$$R_n = \frac{540}{2^n} \quad \text{HAZE per block} \tag{2}$$

9.2 Genesis Allocation

Category	Allocation
Block rewards (emitted over time)	65%
Team and future advisors	13%
Investors (pre-seed / seed)	13%
Community airdrop	6%
Treasury reserve	3%

Table 2: Genesis and long-run supply allocation

The treasury allocation funds the network's development-stage faucet in addition to future ecosystem needs.

9.3 Vesting

Team, advisor, and investor genesis allocations are locked by a protocol-level unlock-height mechanism, enforced during output validation, rather than relying on convention or off-chain trust alone. The cryptographically random blinding secrets backing these locked outputs were generated independently, out-of-band, and are not present in the project's source repository or its history; only the resulting public commitments, range proofs, and kernel data are embedded in genesis. An earlier development-stage version of this mechanism used small, hard-coded blinding values checked into source, which would have allowed anyone reading the repository to construct a valid spend regardless of the unlock height; this has been corrected, and the fix is treated as a disclosed part of the project's history rather than omitted.

Disclosed limitation: public disclosure of the exact unlock heights and per-tranche schedule for team and investor allocations is planned prior to any public distribution event or fundraising close, rather than published in this document.

10 Security Model

10.1 Threat Model

Haze's security model considers the following:

- **Invalid value creation:** An adversary attempts to construct a transaction encoding a negative value via modular wraparound that satisfies the balance equation without a valid range proof. This is rejected unconditionally at the range-proof verification step, which has been tested directly against a constructed attack transaction, at both the isolated transaction level and the full block-application level against a chain seeded from genesis.
- **Chain-state corruption via pruning:** An adversary or software defect attempts to alter validated chain state through the compaction process. Compaction is a pure function verified to leave tip hash, UTXO set, and kernel count identical between compacted and uncompacted copies of the same chain.
- **Cross-network replay:** A block from one Haze network (for example, a development network) is presented to a node on a distinct network. Block headers include a chain identifier field, verified during block application, preventing cross-acceptance between networks with otherwise compatible serialization formats.

10.2 Current Guarantees and Gaps

The following properties are implemented and tested: confidential transaction amounts via Pedersen commitments; unconditional range proof enforcement preventing invalid value creation; kernel-based permanent proof of transaction validity; horizon-based cut-through with verified compaction safety; aggregate validation allowing safe sync through pruned peers; Dandelion++ transaction relay; deterministic wallet key recovery covering balances, names, and registry-held assets; a dynamic, size-based fee model; and a protocol-level vesting lock on genesis-allocated outputs backed by secrets held outside the source repository. The following remain open: publication of exact vesting unlock heights and per-tranche schedule prior to any public distribution event; independent, third-party security review of the cryptographic implementation; and validation of the network's behavior under a validator set and infrastructure footprint broader than the current development configuration.

11 Regulatory Considerations

Haze is a protocol, not a company, and does not custody user funds at any point — all keys and blinding factors are generated and held client-side. The regulatory environment for privacy-preserving cryptocurrencies has shifted meaningfully over the course of this protocol's development, with several jurisdictions restricting exchange listing and custody of confidentiality-by-default assets. This document does not take a position on the merits of that policy trajectory; it is noted here as a material fact relevant to the network's likely adoption path, particularly with respect to centralized exchange access. Haze's design does not include any built-in compliance or selective-disclosure mechanism at this time.

12 Roadmap

Phase	Milestones
Devnet (current)	Core protocol, wallet stack, naming registry, cut-through, aggregate pruned-sync validation, Dandelion++, dynamic fees, and key recovery implemented and tested. Running on hosted infrastructure without a persistent disk guarantee.
Testnet	Migration to dedicated, persistent infrastructure. Publication of exact vesting unlock heights and per-tranche schedule. Validator set and independent node participation broader than the current development configuration.
Pre-mainnet	Independent, third-party security review of the cryptographic implementation.
Mainnet	Genesis lock finalized. Public network launch.

Table 3: Haze development roadmap

13 Conclusion

We have described Haze, a working Mimblewimble-based Layer 1 combining confidential transaction amounts, bounded chain growth through tested cut-through compaction and aggregate pruned-history validation, network-layer transaction relay, a dynamic fee model, and an on-chain naming registry with deterministic key recovery — a wallet-to-network stack intended for real use. What remains before a public, non-development network is not, at present, protocol design: it is infrastructure durability, disclosure of finalized vesting terms, and independent security review, and this document has attempted to state that plainly rather than conflate development progress with production readiness. The project's scope remains deliberately limited to private payments; it does not, and is not intended to, support smart contracts, wrapped assets, or multi-asset issuance.

References

[1] Tom Elvis Jedusor. Mimblewimble. `https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.txt`, 2016.

[2] Grin Developers. Grin: A Minimal Implementation of Mimblewimble. `https://github.com/mimblewimble/grin`, 2019.

[3] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short Proofs for Confidential Transactions and More. *IEEE Symposium on Security and Privacy*, 2018.

[4] Nicolas van Saberhagen. CryptoNote v2.0. `https://cryptonote.org/whitepaper.pdf`, 2013.

[5] Henry de Valence, Isis Lovecruft, and Tony Arcieri. The Ristretto Group. `https://ristretto.group/`, 2019.

[6] NEAR Protocol. Human-Readable Accounts. `https://docs.near.org/concepts/protocol/account-model`, 2021.

[7] Giulia Fanti, Shaileshh Bojja Venkatakrishnan, Surya Bakshi, Bradley Denby, Shruti Bhargava, Andrew Miller, and Pramod Viswanath. Dandelion++: Lightweight Cryptocurrency Networking with Formal Anonymity Guarantees. *ACM SIGMETRICS*, 2018.