

Whitepaper · v1.0 · 15 October 2026

Shroud.

A shielded pool with post-quantum notes.

Pre-launch: the contracts are not deployed or audited, and the trusted-setup ceremony is still to come.

Abstract

Shroud is a shielded pool for ETH, ERC-20 tokens and tokenized stocks on Ethereum and Robinhood Chain. Funds go in publicly, then move, trade and come out privately: every private action is a Groth16 zero-knowledge proof built in the user's browser and checked by an immutable contract. Ownership is defined by Poseidon hashes, not elliptic-curve keys. Notes are encrypted with X-Wing (ML-KEM-768 + X25519), so ciphertexts stored on-chain today stay sealed against a future quantum computer. Because the proof system itself is not post-quantum, a one-way quantum switch and a hash-only exit keep the money recoverable if its curve is ever broken. This paper describes the design, the cryptography, the contracts, the threat model, and what is not done yet.

1 · Motivation

Why payments need privacy.

A public blockchain records every balance and every payment permanently. Pay a supplier and they can read your treasury. Get paid and the payer can read your savings. Salaries, donations and trading positions all become public record. For individuals that is a safety problem; for businesses it rules out most real use.

Zero-knowledge proofs solve this without a trusted party: a transaction can prove that it is valid (the coins exist, belong to the sender and are not spent twice) while revealing nothing else. Zcash and the privacy pools that followed it showed that this works on a public chain.

Shroud builds on that line of work with four goals in mind: many assets in one anonymity set, including tokenized stocks; private trading against public Uniswap liquidity; compliance at the edges instead of a backdoor; and readiness for **Q-day**, the day a quantum computer breaks the elliptic-curve cryptography most of crypto runs on. A chain is a permanent archive. Anything encrypted with classical curves alone and posted today can be read on that day.

2 · Goals

What it does, and what it doesn't.

Goals

- **Unlinkability inside the pool.** A private send or swap reveals no sender, recipient, amount or balance.
- **Self-custody.** Nobody can move a note without its owner's secret. No admin function can freeze or seize a note, and the contract can not be upgraded.
- **Gasless private actions.** A relayer can submit any proof and pay the gas, but can not change anything the proof commits to.
- **Private trading.** Notes can be swapped against public Uniswap v4 liquidity without the trade pointing back at the trader.
- **Quantum readiness.** Privacy that survives Q-day, and money that can still get out if the proof system's curve falls.
- **Compliance at the edges.** Sanctions screening where an oracle exists, viewing keys for selective disclosure, and deposit caps.

Non-goals

- Hiding deposits and withdrawals. Both are public transfers: the address, token and amount are visible.
 - Hiding the token of a private transaction. Each proof is for one token, and that token is public.
 - Defeating timing and amount correlation by itself. Depositing 1.37 ETH and withdrawing 1.37 ETH a minute later is easy to match; users have to vary amounts and wait.
 - Protecting the user's normal Ethereum wallet, which is outside Shroud.
-

3 · Overview

How the parts fit together.

Each chain runs one `ShieldedPool` contract. It holds the assets and two public data structures: a Merkle tree of **note commitments** (who owns what, hidden) and a set of spent **nullifiers** (which notes are gone, unlinkable to the notes).

```

wallet —shield (public)→ ShieldedPool —withdraw (public)→ any address
                        | commitment tree (223 leaves)
                        | nullifier set
relayer —proof→ | Groth16 verifier
                | Uniswap v4 PoolManager (private swaps)

```

A user holds **notes**. A private action spends up to four notes and creates two, proving in zero knowledge that the inputs exist in the tree, belong to the prover, are unspent, and that the amounts balance. The chain sees the proof, four nullifiers, two new commitments and two encrypted notes, and nothing that links them to earlier activity.

The client (the web app and its SDK) holds the keys, scans the chain for the user's notes, builds proofs in a Web Worker, and sends them to a relayer or straight to the contract. No server ever sees a key.

4 · Keys

How keys are derived.

A wallet is a BIP-39 phrase of 24 words (256 bits of entropy) generated in the browser. Everything else is derived from it:

```

spendSecret    = keccak256("SHIELDP00L/v1/spend"    || entropy) mod p
nullifierSecret = keccak256("SHIELDP00L/v1/nullifier" || entropy) mod p
kemSeed        = keccak256("SHIELDP00L/v1/kem"       || entropy)          (32 bytes)

authKey = Poseidon(spendSecret)
owner   = Poseidon(authKey, nullifierSecret)

```

`p` is the BN254 scalar field order. The encryption key pair comes from `kemSeed` exactly as X-Wing expands it: SHAKE256 gives a 64-byte ML-KEM-768 seed and a 32-byte X25519 secret. The public key is 1,216 bytes (ML-KEM-768's 1,184 plus X25519's 32).

Addresses

A private address is bech32m with the prefix `zk`, carrying a version byte, `owner` (32 bytes) and the X-Wing public key: 1,249 bytes, about 2,000 characters. It is long because it carries a quantum-safe key. It never appears on-chain; people share it as a Shroud Pay link or QR code. The same address works on every chain, while notes live in each chain's pool.

Viewing keys

A viewing key (`zkview`, 97 bytes) holds `authKey`, `nullifierSecret` and `kemSeed`. It decrypts every note and recognises every spend, so it shows a full balance and history, but it can not spend: spending requires `spendSecret`, the preimage of `authKey`.

5 · Notes

Commitments and nullifiers.

A note is a tuple (token, amount, owner, blinding). The chain only ever sees hashes of it:

```

npk          = Poseidon(owner, blinding)           one-time owner tag
commitment = Poseidon(npk, token, amount)         the leaf in the tree
nullifier   = Poseidon(commitment, leafIndex, nullifierSecret)

```

- **Hiding.** The blinding is a fresh random field element, so a commitment says nothing about its owner or amount, even to unlimited computing power.
- **Binding.** Opening a commitment to a different note would need a Poseidon collision.
- **One nullifier per note.** The nullifier depends on the leaf index and a secret only the owner has. Publishing it marks the note spent without saying which leaf it was.
- **Range.** Every amount is proven below 2^{248} , so sums can not wrap around the field.

The **empty note** (owner 0, blinding 0, amount 0) pads unused outputs; zero-amount inputs are dummies that skip the tree check.

6 · Encryption

How notes are encrypted.

Whoever creates a note encrypts its opening (token, amount, blinding) to the receiver, so the receiver can find and spend it:

```
(ct, ss) = X-Wing.Encapsulate(receiverPublicKey)          ML-KEM-768 + X25519
key      = HKDF-SHA256(ss, salt = ct, info = "SHROUD/v2/note")
sealed   = XChaCha20-Poly1305(key, nonce = 0, version || token || amount || blinding)
note ciphertext = 0x02 || ct (1,120 bytes) || sealed (101 bytes)    = 1,222 bytes
```

Every note gets a fresh encapsulation and therefore a fresh key, so the fixed nonce is safe. X-Wing combines both shared secrets: the note stays sealed if **either** ML-KEM-768 (NIST FIPS 203) or X25519 holds. A quantum computer that breaks X25519 still faces ML-KEM; a flaw found in the young ML-KEM still faces X25519.

Finding your notes

The wallet trial-decrypts every note ciphertext in the pool. In a browser that costs about 0.44 ms per note (WebCrypto X25519 plus ML-KEM decapsulation): roughly 4.4 seconds per 10,000 notes, done in a Web Worker and cached. Shroud deliberately has no “view tag” shortcut: a tag computed from the classical key alone would let a future quantum adversary tell which notes belong to a known address.

Deposits are the exception: their token and amount are public anyway, so the pool computes the commitment on-chain from the depositor’s `nPK` and publishes the note’s token and amount alongside the ciphertext.

7 · The tree

One tree for every asset.

Commitments go into an incremental Poseidon Merkle tree of depth 23: 8,388,608 leaves, shared by every token. Each action appends exactly two leaves (the two outputs, or a deposit and the empty leaf), so the contract updates one path per action.

- Empty slots hold a fixed zero leaf; the roots of empty subtrees are precomputed constants.
- The contract accepts any of the last **200 roots**, so a proof built a few blocks ago still verifies after other actions land.

- A full withdrawal whose two outputs are both the empty note appends nothing, so a full tree can never stop anyone from taking everything out.

8 · The circuit

Transaction(23, 4, 2).

One Circom circuit covers sends, withdrawals and swaps. Public inputs: `root`, `publicAmount`, `extDataHash`, `token`, four input nullifiers and two output commitments. Private inputs: the two secrets, each input's amount, blinding, leaf index and Merkle path, and each output's amount, owner and blinding.

1. **Ownership.** `owner = Poseidon(Poseidon(spendSecret), nullifierSecret)`; every input note is rebuilt from it, so only the holder of `spendSecret` can spend.
2. **Membership.** Each input commitment with a non-zero amount has a valid path to `root`.
3. **Nullifiers.** Each published nullifier is the input's true nullifier, and no two are equal.
4. **Outputs.** Each output commitment is `Poseidon(Poseidon(owner, blinding), token, amount)` for the same token.
5. **Range.** Every input and output amount is below 2^{24^8} .
6. **Balance.** `$\sum \text{inputs} + \text{publicAmount} = \sum \text{outputs}$` in the field, where `publicAmount` is `p - leaving` for value leaving the pool.
7. **Binding.** `extDataHash` is constrained into the proof, tying it to the recipient, fees, swap terms and note ciphertexts.

The circuit has about 28,000 constraints. Proofs are generated in the browser with snarkjs in a few seconds and verified on-chain by a generated Groth16 verifier.

9 · Operations

Shield, send, swap, withdraw.

Shield

`shield(token, amount, npk, ciphertext)` moves funds in from any wallet. The depositor is screened (on Ethereum), the pool's holdings of that token must stay under its cap, the 0.5% fee is set aside, and the net note is committed on-chain.

Transact: send and withdraw

`transact(proof, extData)` carries `ExtData = (token, recipient, withdrawAmount, relayer, relayerFee, ciphertext1, ciphertext2)`. Its hash is domain-separated by a tag, the chain id and the pool address, so a proof can not be replayed elsewhere:

```
extDataHash = keccak256(abi.encode("ShieldedPool.transact.v1", chainid, pool, extData)) mod p
```

Value leaving the pool is `withdrawAmount + relayerFee + fee`, with the 0.5% fee charged on both. A private send has `withdrawAmount = 0`: nothing is charged on the amount sent, only the relayer's gas fee and 0.5% of that fee. The recipient and the relayer are screened like any exit.

Fees

Action	Protocol fee
Shield	0.5% of the deposit
Private send	0% of the amount (the relayer fee is charged 0.5%)
Swap	0.5%, on the ETH side when the output is ETH, else on the input
Withdraw	0.5% of the amount out
Relayer fee	0.5% of what the relayer is paid

The fee is a constructor constant capped at 1%. Fees accrue per token and anyone can call `collect(token)` to send them to the platform address. Measured gas on an Ethereum fork: about 1.06M for a relayed send, 1.09M for a withdrawal, 1.24M for a one-hop swap and 1.29M for a two-hop swap.

10 · Swaps

Trading on Uniswap from inside the pool.

A private swap spends notes of one token and creates a note of another, trading against Uniswap v4 from inside the pool. Everyone sees that the pool traded; nobody sees whose notes paid.

No calldata from the caller

An earlier design passed router commands from the user to Uniswap's Universal Router. That let a swap send its output anywhere (skipping withdrawal screening) and could credit notes of worthless tokens. Shroud now trades directly: the pool calls `PoolManager.unlock` itself and, inside the callback, swaps exact-input along a

path of one to three **listed** pools. Every hop must fill completely. The pool then pays exactly the input and takes exactly what the last hop returned, to itself; any other change to its balances leaves an open delta and the whole swap reverts.

Listing rules

- A token is listed if and only if its deposit cap is above zero. Caps only go up.
- A pool can be listed only if it is already initialised, both its tokens are listed, and it has no hook or a listed hook.
- Listings are permanent. Nothing can be unlisted or redirected.

Terms in the proof

The swap terms (output token, amount in, minimum out, relayer tip, output owner tag, deadline, path) are hashed into `extDataHash`. The proof's second output must be the empty note: the swap output takes its place, built on-chain from the amount actually received. Caps apply after the swap too, so a swap can not push a token past its cap. A `quote` function runs the same path and reverts with the result, for wallets to price trades.

What the contract can not stop: someone moving a listed pool's price inside the user's slippage. Only deep pools should be listed, and on Ethereum relayed swaps should go out through a private mempool.

11 · Relayers

Who pays the gas.

Submitting a transaction from your own wallet would link that wallet to the action. Instead, a relayer submits the proof and pays the gas, and is paid inside the transaction: `relayerFee` in the input token, or a `tip` in a swap's output token.

- A relayer can not change the recipient, amounts, fees or notes: they are all bound by `extDataHash`.
- The relayer is screened like a withdrawal recipient, because the prover chooses who gets paid.
- A relayer can refuse to submit, but can not steal. Anyone can run one, and users can always submit with their own wallet instead.
- A relayer sees network metadata (an IP address) for the requests it receives. Users who care should reach it through Tor or a VPN.

Relayers accept fees in ETH and stablecoins and, on Robinhood Chain, in USDG and the listed stocks, so a user can withdraw a stock to a new address that holds no gas.

12 · Robinhood Chain

USDG and tokenized stocks.

Robinhood Chain (chain id 4663) is an Ethereum layer 2 built on Arbitrum technology, home to tokenized US stocks and ETFs. Shroud runs the same contract there, with the same Poseidon libraries at the same addresses.

- **Assets at launch:** ETH, USDG (Global Dollar) and 15 stock tokens: TSLA, NVDA, AAPL, AMZN, GOOGL, META, MSFT, AMD, MSTR, COIN, PLTR, NFLX, CRCL, ORCL and INTC, each capped at roughly \$100k to begin with.
- **Routing:** USDG is the hub. ETH to a stock trades ETH → USDG → stock; stock to stock trades through USDG. One proof covers the whole path.
- **Cost:** a private action costs cents in gas.

Issuer risk.

Stock tokens are controlled by their issuer, which can pause transfers and block or burn balances at any address, the pool's included. A shielded stock balance is only as safe as the issuer's treatment of the pool.

Screening.

There is no Chainalysis sanctions oracle on Robinhood Chain. Whether that pool launches without screening or with a separately maintained list is an open decision, and the app says so either way.

13 · Post-quantum

What happens on Q-day.

Q-day is the day a quantum computer can run Shor's algorithm at scale. It breaks discrete logarithms on elliptic curves, and with them ECDSA wallet signatures, X25519 key exchange and pairing-based proof systems. Grover's algorithm only halves the strength of hash functions: a 256-bit hash keeps about 128 bits, which remains safe. Nobody knows the date. NIST published its first post-quantum standards in August 2024 (FIPS 203, 204 and 205), and US agencies are working to a 2035 goal to move off quantum-vulnerable cryptography.

Part	Against a quantum computer
Note encryption	Holds: X-Wing, safe if ML-KEM-768 holds
Note ownership	Holds: Poseidon hashes, no curve key
What proofs reveal	Holds: Groth16 is perfectly zero-knowledge
Proof soundness	Breaks: Groth16 on BN254 can be forged. Covered by the quantum switch
Your money	Exits with hashes only once the switch is on
Admin key	ECDSA today; a hash-based key is planned
Your Ethereum wallet	Outside Shroud

Why privacy survives

Notes are encrypted with X-Wing (section 6), commitments and nullifiers are hashes, and Groth16 proofs are perfectly zero-knowledge: they carry no information about the witness at all, so there is nothing for a quantum computer to recover from past proofs.

The quantum switch

Groth16's **soundness** rests on discrete logarithms in BN254. Whoever can compute them can recover the setup trapdoor and forge proofs. So the pool has `quantumMode`: one-way, on and never off. It can be switched on by the admin, or by anyone who calls `tripCanary(k)` with the discrete logarithm of a public canary point:

```
seed      = keccak256("Shroud/quantum-canary/v1")
x         = keccak256(seed || i) mod p,  first i with x³ + 3 a square (i = 0 works)
CANARY    = (x, smaller square root)      a BN254 G1 point
trip(k)   : ecMul(G1, k) == CANARY  →  quantumMode = on
```

Nobody knows the canary's discrete log; producing it proves BN254 is broken. Once the switch is on, `shield`, `transact` and `swap` refuse, the tree is frozen, and fee collection still works.

The hash-only exit

1. **Commit.** The wallet publishes `h = keccak256(tag, chainid, pool, spendSecret, nullifiers, exitTo)` for all of its notes. Only the first commit of each `h` is recorded, and nobody without `spendSecret` can make an `h` that a reveal will match.
2. **Wait** one hour (`EXIT_DELAY`).

3. **Reveal** the secrets and each note's token, amount, blinding, leaf index and Merkle path, up to 16 notes per transaction (about 0.56M gas per note). The contract recomputes every hash itself with Poseidon, checks each path against the frozen root and each nullifier against the spent set, finds the matching commit, and pays out with no protocol fee.

The exit needs `spendSecret`, not `authKey`, so a viewing key can never take money out.

The limits, stated plainly.

An exit is public: it reveals the wallet's secrets, so that wallet's whole history can be linked. After the first reveal, anyone who knows a remaining note's blinding (its sender) could commit to take it an hour later, so the app commits for every note first and reveals them all back to back. Most of all, the switch only protects funds if it flips before someone forges a proof: a silent attacker would not trip the canary. The admin is meant to flip it as soon as a capable quantum computer looks close.

Next

Hash-based proof systems (STARKs) are sound against quantum computers and need no trusted setup. Moving Shroud's proofs to one would remove the reliance on the switch entirely. That is research, not a promise: verifying such proofs on Ethereum is still expensive.

14 · Compliance

Screening and disclosure.

- **Sanctions screening.** On Ethereum, depositors, withdrawal recipients and relayers are checked against the Chainalysis sanctions oracle. Deposits fail closed (if the oracle errors, the deposit is refused); exits fail open, so an oracle outage can never trap funds.
- **Viewing keys.** A user can hand an auditor, accountant or counterparty a read-only view of their full history, without giving up control.
- **Deposit caps** per token, which can only be raised, limit how much the pool can hold while it is young.
- **Pausing deposits** is the only stop. Sends, swaps and withdrawals can not be paused; only the quantum switch ends them, and it opens the exit at the same time.

There is no function that freezes, seizes or redirects a note.

15 · Admin

What the owner can do.

The contract has no proxy and no upgrade path. The owner can only:

- raise a token's deposit cap (which also lists the token);
- pause or resume **new deposits**;
- list Uniswap v4 hooks and pools for swaps, permanently;
- switch quantum mode on;
- hand the role to another address, or give it up by setting it to zero.

The platform address receives collected fees and can only hand its own role on. Neither role can touch a note. Today the owner is an ordinary (ECDSA) key; replacing it with a hash-based signature scheme is planned and not built.

16 · Trusted setup

The phase-2 ceremony.

Groth16 needs a per-circuit setup. If anyone kept the setup's secret randomness (the "toxic waste"), they could forge proofs and drain the pool.

- **Phase 1** uses the public Perpetual Powers of Tau (the PSE `ppot_0080_16` file, its SHA-256 pinned in the repository).
- **Phase 2** will be a public multi-party ceremony on Shroud's exact circuit. It is secure if at least one participant destroys their contribution.
- The circuit build is deterministic and its hashes are pinned. A script checks any final key against the circuit and the Phase 1 file, and regenerates the verifier from it.

Not done yet.

The verifier in the repository today comes from a single-party development key. It is for testing only and must never hold real funds. Shroud does not launch before the ceremony.

17 · Security

Who can do what.

Party	Can	Can't
Chain observer	See deposits, withdrawals, the token of each action, and timing	Link a private action to a deposit, sender, recipient or balance
Relayer	Refuse to submit; see network metadata	Change recipient, amounts or fees; steal
Note sender	Know the note they sent	See its later spend, or spend it
Admin	Raise caps, pause deposits, list pools, flip the quantum switch	Move, freeze or seize any note; upgrade the contract
Stock issuer	Pause, block or burn its token at the pool	Affect other assets
Quantum adversary	Forge proofs before the switch flips	Read notes; take funds through the exit without the secrets

Privacy in practice

The anonymity set is everyone who uses the same token in the same pool. It is weakest at launch and grows with use. Deposit and withdraw different amounts at different times; withdraw to fresh addresses; use a relayer; reach it over Tor or a VPN. A compromised browser or device defeats everything above.

Testing

The contracts have 46 Foundry security tests (swap attacks, accounting, the quantum switch and exit). End-to-end suites run real proofs against forks of Ethereum and Robinhood Chain: deposits, sends, multi-hop swaps, withdrawals, relayers, payment links and the full quantum exit. None of this replaces an external audit, which has not happened yet.

18 · \$SHROUD

The coin.

\$SHROUD is an ERC-20 on Robinhood Chain with a fixed supply of 1,000,000,000. All of it goes into a Uniswap v4 ETH pool on Robinhood Chain at launch, and the liquidity is held by the coin contract permanently, so nobody can pull it. Trades pay 2%: a 1% pool fee and 1% on the ETH side. The coin contract itself has no transfer tax, no mint function and no blacklist (the 2% is charged by the pool); a 2% maximum wallet

applied for the first three minutes after launch and was then removed, and the coin contract's ownership was renounced.

The protocol does not depend on the coin. Holding \$SHROUD gives no claim on the pool, its fees or anyone's notes. Nothing in this paper is an offer or a promise of returns.

19 · Roadmap

What is left before launch.

1. Public Phase 2 trusted-setup ceremony, then the production verifier.
2. External audit of the contracts and circuit.
3. A decision on screening for Robinhood Chain.
4. Deployment on Ethereum and Robinhood Chain, token and pool listings, and public relayers.
5. A hash-based admin key, or giving up the admin role.
6. Research into post-quantum proofs.

Parameters

Parameter	Value
Circuit	Transaction(23, 4, 2), about 28,000 constraints, Groth16 on BN254
Tree	Poseidon, depth 23 (8,388,608 leaves), last 200 roots valid
Note encryption	X-Wing (ML-KEM-768 + X25519), HKDF-SHA256, XChaCha20-Poly1305
Note ciphertext	1,222 bytes (contract limit 1,536)
Address	bech32m "zk", 1,249 bytes
Protocol fee	0.5% (hard cap 1%)
Swap path	1 to 3 listed Uniswap v4 pools
Quantum exit	1 hour delay, up to 16 notes per reveal
Chains	Ethereum (1), Robinhood Chain (4663)

References

Further reading.

1. J. Groth. *On the Size of Pairing-based Non-interactive Arguments*. EUROCRYPT 2016.
2. L. Grassi et al. *Poseidon: A New Hash Function for Zero-Knowledge Proof Systems*. USENIX Security 2021.
3. NIST. *FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard*. August 2024.
4. M. Barbosa et al. *X-Wing: The Hybrid KEM You've Been Looking For*. 2024.
5. S. Arciszewski. *XChaCha: eXtended-nonce ChaCha and AEAD_XChaCha20_Poly1305*. IETF draft.
6. H. Krawczyk, P. Eronen. *RFC 5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. 2010.
7. P. Wuille. *BIP-350: Bech32m*. M. Palatinus et al. *BIP-39: Mnemonic code for generating deterministic keys*.
8. Uniswap Labs. *Uniswap v4 Core*. 2024.
9. P. Shor. *Algorithms for Quantum Computation: Discrete Logarithms and Factoring*. FOCS 1994. L. Grover. *A Fast Quantum Mechanical Algorithm for Database Search*. STOC 1996.
10. Privacy & Scaling Explorations. *Perpetual Powers of Tau*.

This paper describes software that is pre-launch and unaudited. It is not financial, legal or tax advice, and not an offer of any token or security. Using privacy tools may be restricted where you live; you are responsible for following the law that applies to you.
