



TECHNICAL WHITEPAPER • V1.0 • OCTOBER 2026

Atomic settlement for institutional crypto OTC

A six-chain HTLC protocol in which the escrow's address is derived from the agreed terms — so an escrow funded on any other terms is a different address, and can never be mistaken for the trade.

Hashlock Markets • Delaware C-Corp • File #10561489

Barış Sözen, Co-Founder & CEO • Aleksei Diakonov, Co-Founder & CTO

Istanbul • Amsterdam baris.sozen@hashlock.markets

Basis of this document. Every technical statement below is taken from the implementation on the audit-r3 branch — the contracts, the Solana program, the Bitcoin script layer and the settlement service — not from product material. Where the code documents a limit, the limit is reproduced here rather than omitted.

Status. Settlement is live on mainnet across six chains and the protocol fee is charged; the external audit is complete (October 2026).

Contents

01	Abstract	10	Solana: PDA escrow
02	The settlement gap	11	Bitcoin: P2WSH
03	Goals and non-goals	12	Off-chain settlement
04	The HTLC primitive	13	Fees
05	Escrow identity	14	Hardening and audit
06	Dual-HTLC settlement	15	Distribution
07	Timelocks	16	Assets and listing
08	Ethereum: factory and clone	17	Known limits
09	Tron: shared pool	18	Summary

Abstract

Two institutions agreeing a trade over the counter still face the oldest problem in settlement: one of them has to move first. Hashlock removes that asymmetry by locking both legs under one cryptographic condition, so the trade completes on both sides or refunds on both sides, with no third party holding value at any point.

That much is a textbook atomic swap. What makes it a settlement system rather than a demonstration is a second property, and it is the one this paper is organised around: [the escrow's on-chain address is derived from the agreed terms of the trade](#). An escrow funded on any other terms — a different amount, a different recipient, a different fee — lives at a different address, and nothing in the system that derives from the agreement can be pointed at it.

Two of the findings from the contract audit rounds were the same invariant missing: a decoy escrow rebinding a swap, and an underpaid protocol fee landing on the escrow the parties had agreed. Both are closed by making the terms determine the address — as a CREATE2 salt on Ethereum, a storage slot key on Tron, and PDA seeds on Solana.

The protocol runs on six chains — Bitcoin, Ethereum, Base, Robinhood Chain, Tron and Solana — live on mainnet under one atomic guarantee, with no bridge and no wrapped representation of any asset.

Neither party has to be a person. The same rail is in production for software: the MCP server is live — hosted at `https://hashlock.markets/mcp`, or run locally over stdio — and an agent holding its own keys can run the whole loop, request through negotiation to funding and claim, with no human step in it — with the server run locally so it holds those keys; the hosted endpoint returns unsigned transactions for the holder to sign. [The institutional desk is the first](#)

market; the construction does not require a human counterparty. What follows is written from the settlement layer, which is the same either way.

Chain	Contract	Note
Bitcoin	No contract — a P2WSH HTLC script per leg	3 confirmations
Ethereum	HTLCFactory Oxa9B6a64837259034E949787B5d17Bfa46304ff71	Chain id 1
Base	The same factory address	Chain id 8453
Robinhood Chain	The same factory address	Chain id 4663
Tron	SharedHTLC pool contract TE97fe1beR5nRZzjixi4azE36qWcHEWmNj	
Solana	HTLC program 7Egsecq7YU5n7PonndM7W7W92EFLZVGrhoaNPW8n1DEc	One PDA per escrow

Live mainnet deployments.

02 The settlement gap

Institutional crypto OTC is negotiated bilaterally, very often in a Telegram group, and then settled by one of three mechanisms, each of which reintroduces the trust the trade was meant to avoid.

Mechanism	How it works	What it costs
Send first	One side transfers, then waits	Full principal at risk for the duration
Human escrow	A trusted agent holds both sides	A spread, a manual timeline, a custodian who can fail
Exchange rail	Both sides move to a venue	Custody risk, withdrawal limits, a visible footprint

The cost of the third pattern is documented: **roughly \$12.9B lost between 2022 and 2025 in custodial failures** — \$8.0B in FTX customer accounts, \$3.4B owed to Genesis creditors, \$1.5B drained from Bybit's hot wallet in one night — with a further \$2B in bridge exploits at Ronin, Wormhole and Nomad. None were failures of price discovery. They were failures of the step after it.

Cross-chain makes it worse. Moving value between chains today generally means a wrapped token and a bridge, replacing counterparty risk with the risk of a validator set or a multisig. The bridge losses are the evidence.

The same gap, without a person on either side

An autonomous agent transacting on its own keys faces the identical problem and has fewer ways around it. It cannot be vouched for, cannot be onboarded by a relationship, and has nobody to call when a transfer goes out and nothing comes back. Its options are to send first or to settle under a condition. The mechanism this paper describes is the second one, and it is already reachable without a human: the MCP server is live, hosted at `https://hashlock.markets/mcp` or run locally over stdio, and the loop from request to claim runs through it. OTC desks are where the volume is today and the argument above is written for them; nothing in the settlement layer is weaker when the counterparty is a program.

03

Goals and non-goals

Goals

- **Binary outcome.** The trade completes in full or refunds in full. No partial state is reachable.
- **Non-custodial throughout.** Funds sit in on-chain escrows. Hashlock holds nothing and has no privileged path to locked funds on any chain.
- **Native assets.** BTC settles as BTC on Bitcoin. No wrapping, no synthetic representation, no bridge.
- **Verifiable before funding.** A counterparty can compute the exact escrow address from the agreed terms and check it before committing a single unit.
- **Atomic fee capture.** The protocol fee moves in the same transaction that creates the escrow.

Non-goals

- **Not an exchange.** No order book, no matching, no market making. Price is agreed bilaterally.
- **Not a bridge.** No asset crosses a chain. Two assets on two chains change hands simultaneously.
- **Not a custodian.** No omnibus wallet, no float, no balance sheet against which to fail.

04

The HTLC primitive

An escrow locks funds under two independent conditions.

- **Hashlock.** Claimable by presenting a preimage `s` with `sha256(s) = h`, where `h` is fixed at funding.
- **Timelock.** If no valid preimage arrives before time `t`, the funder may reclaim.

Exactly one path executes; settle-once flags and checks-effects-interactions ordering enforce this on every chain. **The escrow cannot be left in a state where value is neither claimable nor refundable.**

Why SHA-256 and not keccak256

The hashlock is `sha256(secret)` across every rail, including the EVM contracts. Solidity's native hash is keccak256 and would have been the obvious choice, but Bitcoin Script offers `OP_SHA256` and not keccak. Choosing SHA-256 everywhere is what makes an Ethereum leg and a Bitcoin leg openable by the same preimage — the whole cross-chain construction depends on that one decision.

The preimage is exactly 32 bytes, and this is a cross-chain requirement rather than a local preference. The EVM and Tron claim functions take `bytes32`, so a longer secret cannot be expressed on those rails at all. The Solana instruction originally accepted an unchecked `Vec<u8>`; review caught it. The consequence would have been the counterparty's entire principal — an initiator choosing a 33-byte secret could open a Solana short leg and leave the EVM or Tron long leg unopenable by anyone, then refund it at its timelock and keep both sides. A hashlock reveals nothing about the length of what opens it, so the counterparty could not have checked.

05

Escrow identity

The escrow's address is derived from the agreed terms. This is the invariant the implementation is built around, and it is what distinguishes the system from a generic atomic swap.

Each chain expresses it differently, because each chain offers a different deterministic-address mechanism.

Chain	Mechanism	Derived from
Ethereum • Base • Robinhood Chain	CREATE2 over the clone's immutable args	hashlock, recipient, token, amount, timelock, initiator, fee
Tron	Storage slot key in a shared pool	<code>slot0f(agreedId, funder, fee)</code>
Solana	PDA seeds	<code>["htlc", swap_id, terms_hash]</code>
Bitcoin	P2WSH script hash	The redeem script itself — hashlock, timelock, both pubkeys

What the invariant buys

It stops anything that derives the escrow from the agreed terms — the watcher, the keeper, every transaction builder — from ever being pointed at a different escrow. A party using Hashlock software cannot accidentally fund the wrong escrow, and cannot be induced to.

What it does not buy

It does not stop two consenting parties from settling some other way on-chain and calling it this swap. Nothing on-chain can. The code states this limit in the same place it states the property, on both the EVM and Solana rails, and this paper reproduces it rather than rounding it up.

The condition the property depends on

On Ethereum the fee is one of the immutable args, so it feeds both the address and the `swapId`. That closes the underpaid-fee finding — but only if the off-chain derivation uses the agreed fee, taken from the agreement row. The watcher's binding check deliberately feeds hashlock, recipient, token, timelock and initiator straight from the on-chain event, because varying those moves the clone and the prediction together. Taking the fee off the event too would move the prediction along with an underpaid clone and buy nothing. The code names this trap explicitly, because it is easy to fall into in exactly the direction that voids the protection.

06

Dual-HTLC settlement

A desk sells 10 BTC against USDT held on Ethereum. Either side may initiate; the initiator is whoever generates the secret and funds the long leg. In this walkthrough, call the seller the initiator and the buyer the counterparty.

1. Initiator generates a random 32-byte preimage s , computes $h = \text{sha256}(s)$.
 2. Both sides derive the escrow addresses from the agreed terms and verify them — on Ethereum via `HTLCFactory.computeAddress(...)`, which returns the exact clone address before any value moves.
 3. Initiator locks 10 BTC in a Bitcoin P2WSH HTLC against h , timelock $T1$.
 4. Counterparty calls `createSwap(...)` on Ethereum, which in ONE transaction deploys the clone at the predicted address, escrows the USDT into it, and routes the protocol fee to the treasury. Timelock $T2$, with $T2 < T1$.
 5. Initiator calls `claim(s)` on the Ethereum clone. s is now public on chain.
 6. The watcher extracts s from the Claimed event and the counterparty spends the Bitcoin HTLC with the same s . A Bitcoin spend needs the owner's signature, so no keeper can submit this leg on their behalf.
- Both legs settled. Neither party could have settled only one.

If step 4 never happens, the initiator waits out $T1$ and refunds. If step 5 never happens, both timelocks expire and both sides refund. The only way the initiator obtains the counterparty's asset is by publishing the secret that lets the counterparty obtain theirs.

Claim is callable by anyone. On all three programmable rails, `claim(secret)` is permissionless — the funds can only ever go to the recipient fixed at funding, so opening the call lets a keeper submit on a counterparty's behalf without being able to redirect anything. This is a deliberate liveness property, not an oversight.

Timelocks

The ordering $T2 < T1$ is what makes the guarantee hold in practice. The counterparty leg must expire before the initiator leg, so that when the initiator claims and publishes the secret, the counterparty still has time to observe the reveal, build a transaction and have it confirmed before their own refund path opens.

All three programmable rails enforce a minimum timelock of 30 minutes — `MIN_TIMELOCK` in `HTLCFactory` and `SharedHTLC`, and `MIN_TIMELOCK = 1800` seconds in the Solana program. The parity is deliberate and was, at one point, broken: the Solana constant was 300 while its comment claimed parity, six times weaker than the floor the other two rails enforced and invisible to anyone reading either file. It is now 1800 and the comment says why the number is not arbitrary — whoever must claim needs time to see the funding, get the preimage from the other leg, submit, and be confirmed before the funder may pull the principal back.

The windows the service actually sets

The floor is a per-contract minimum — the shortest timelock a single escrow will accept. It is not the distance between the two legs, and the two numbers should not be confused. The operative windows are set per chain: a Bitcoin leg runs at **24 hours**, every other chain — Ethereum, Base, Robinhood Chain, Tron and Solana — at **4 hours**.

From those, the service derives the pair. The leg on the chain with the longer safe timelock is the long leg, the one the initiator funds; the short leg takes its own chain's timelock; the long leg takes `max(its chain's timelock, short leg + 2 hours)`. The minimum gap between the legs is 2 hours. On a tie, the taker's leg is the long one.

Pair	Short leg (counterparty)	Long leg (initiator)
Bitcoin ↔ an EVM chain	4 h on the EVM leg	24 h on the Bitcoin leg
Two 4-hour chains — an EVM leg against Solana, say (a tie)	4 h	6 h, by the 2-hour gap

When a leg counts as funded

Funding is observed, not asserted. Each chain has a published threshold at which the settlement service treats a leg as funded and lets the swap advance.

Chain	Funded at	Typically
Bitcoin	3 confirmations	~30 minutes
Ethereum	8 blocks	~1.5 minutes
Base	15 blocks	~30 seconds

Chain	Funded at	Typically
Robinhood Chain	250 blocks	Under a minute
Tron	The block is solidified	~1 minute
Solana	<code>finalized</code>	~15 seconds

The consequence is worth stating rather than hiding: the time between a funding transaction and a funded leg is mostly the chain's, not ours. A Bitcoin leg is the slow side of almost every pair, and the timelock schedule above is built around that fact.

Honest constraint. The 30-minute floor is a minimum, not a calibration. The deployed policy counts a Bitcoin leg as funded at **3 confirmations, roughly 30 minutes** — not at the six-confirmation institutional convention — and a congested fee market can push a low-fee transaction well past either figure. A Bitcoin leg therefore needs a window set well above the floor, which is why it runs at 24 hours rather than anywhere near the minimum; the floor only prevents the pathological case. Section 17 returns to this.

The initiator retains a residual option: having seen the counterparty fund, they may decline to reveal and let both legs refund. Over a volatile window this is a free option on price. It is inherent to the HTLC construction, not to this implementation. Short windows bound it; nothing in V1 eliminates it.

08

Ethereum: factory and clone

`packages/contracts/src/HTLCFactory.sol` · `HTLCImplementation.sol` · Solidity 0.8.24, solady

Rather than one contract holding many swaps in a mapping, the EVM rail deploys one minimal-proxy clone per swap. The swap's parameters are appended to the clone's bytecode as immutable args (solady's clone-with-immutable-args) and read via `LibClone.args0nClone`.

The same `HTLCFactory` runs on three chains — Ethereum, Base and Robinhood Chain, at the one address `0xa9B6a64837259034E949787B5d17Bfa46304ff71` — so everything in this section describes all three EVM rails, and each carries its own 4-hour leg timelock and its own confirmation threshold.

There is no `initialize()`

Because every parameter is fixed in the bytecode at deploy time, the contract has no initialiser — which removes the init-frontrun vector entirely. There is no window in which a deployed-but-unconfigured escrow exists.

```
args = abi.encode(hashlock, recipient, token, amount,
                  timelock, initiator, fee)
clone  = CREATE2(implementation, args, salt)
swapId  = keccak256(args)
```

The args order is load-bearing and append-only: clones already deployed carry their own args tail and keep decoding correctly, but inserting a field before `fee` would make every clone deployed afterwards read its parameters from the wrong offsets.

Settlement pays the held balance, not the stated amount

This is the subtlest decision in the contract and it is worth stating precisely. `claim` and `refund` transfer the clone's actual balance, not the immutable `amount`. Paying the stated figure would brick both paths for a fee-on-transfer token: the clone receives `amount - fee`, so every transfer of `amount` reverts on insufficient balance and the escrow is stuck forever — neither claimable nor refundable. And `amount` cannot be corrected afterwards, because it is baked into the bytecode. One clone escrows exactly one swap, so "everything here belongs to this swap" holds and paying the balance is safe.

Funding measures a delta

For the same family of reasons, `_fund` measures what the clone gained rather than reading its balance after the transfer. The clone address is predictable, so it may already hold a donation; crediting that to the swap would misreport the escrow.

Party checks

- **Audit #15.** Neither recipient nor initiator may be the zero address — a native claim or refund to `address(0)` burns the escrow irreversibly — and `recipient != initiator`, since paying yourself is not a swap.
- **Contract audit issue 5.** `initiator == msg.sender` is enforced, so no escrow can name someone else as the party it refunds to. It does not prevent a second clone under the agreed terms with the caller as initiator; that decoy is answered off-chain, by relaying a reveal on the refused escrow and refunding it at its timelock.

Fee sink

The treasury lives in the contract and is rotatable by the owner or a `FEE_ADMIN_ROLE` holder. It is never passed per call, so a caller cannot redirect fees. The per-call fee is computed off-chain, is non-zero only on the taker's leg, and is routed to the treasury in the same `createSwap` transaction. The fee is on top of the amount and never touches the escrow.

Owner powers, stated in full

The owner and `FEE_ADMIN_ROLE` can rotate the fee sink. That is all. There is [no administrative path to escrowed funds on any rail](#) — no pause on the settlement path, no upgrade proxy on the clone, no recovery function.

09

Tron: shared pool

`packages/contracts/src/tron/SharedHTLC.sol`

Tron does not offer a cheap CREATE2 clone pattern in practice, so the Tron rail uses a different shape for the same invariants: one shared HTLC pool contract holding every asset — TRC-20 and native TRX — with swaps held in `mapping(bytes32 => Swap)` and the storage key derived as `slot0f(agreedId, funder, fee)` — [the same terms-determine-identity property, expressed as a slot rather than an address](#).

Rotation is handled operationally rather than by upgrade: the owner calls `freeze()`, which stops new deposits, and a fresh `SharedHTLC` is deployed for new swaps. The frozen pool keeps honouring `claim` and `refund` until its outstanding timelocks lapse, so no in-flight swap is disturbed.

A concrete Tron-specific defect, and its fix

USDT-TRC20 does not reliably return a boolean from `transfer`. A naïve integration that requires `true` fails against the single most important asset on the chain. The pool therefore uses low-level calls that accept either empty or `true` return data — the kind of detail that decides whether a settlement rail works in the MENA corridor, where USDT-Tron is the settlement asset.

Everything else mirrors the EVM rail: non-custodial escrow, sha256 hashlock, recipient fixed at `fund()`, claim callable by anyone with funds going only to the recipient, checks-effects-interactions with settle-once flags, and `MIN_TIMELOCK = 30 minutes`.

10

Solana: PDA escrow

`packages/solana-program/programs/hashlock-htlc/src/lib.rs` · Anchor

The Solana program derives each escrow as a PDA seeded on `["htlc", swap_id, terms_hash]`, so the account address is a function of the agreed terms and [nothing in the program accepts an escrow account the caller merely names](#). A `Config` PDA holds the treasury and authority.

Native and SPL are separate instructions

There is `fund` / `claim` / `refund` for SPL tokens, and `fund_sol` / `claim_sol` / `refund_sol` for native SOL — rather than one path with optional accounts. The reasoning in the source is that the accounts an instruction may touch should be readable without following a branch, on a path that moves money. Native uses `Pubkey::default()` as its mint throughout, both in the terms hash — so a SOL leg and a token leg resolve to different escrow addresses — and in the stored escrow, so `claim` and `claim_sol` cannot be crossed.

Deliberate limits, reproduced from the source

- **Classic SPL only, not Token-2022.** A Token-2022 mint may carry a transfer-fee extension that would deliver less than `amount` into the escrow — the same fee-on-transfer hole the Tron pool closes by measuring the received balance. USDC on Solana is a classic SPL mint, so today's cost is zero; supporting Token-2022 means measuring the delta rather than trusting the argument.
- **Nothing is ever closed.** A closed PDA can be re-initialised at the same address, which would let a settled swap be funded again, so the terminal state has to outlive the funds. The cost is rent that is never returned — roughly 0.0016 SOL for the escrow account and another 0.002 SOL for its token account on the SPL path, both paid by the funder. An earlier draft of the source comment claimed the token account was closed; it is not, and the correction is recorded rather than quietly dropped.
- **Rent-exempt minimum.** Paying lamports to an account that has never existed fails on a real cluster below the rent-exempt minimum (~890,880 lamports). A native claim of dust to a fresh address, or a dust fee to a never-funded treasury, is refused on-chain and the swap must be refunded. The test harness (litesvm) does not model rent-state transitions, so no test catches this — which is precisely why the source records it.

11

Bitcoin: P2WSH

`packages/api/src/chains/bitcoin/htlc-script.ts` · `extract.ts` · `watcher.ts`

No contract is deployed on Bitcoin. The HTLC is a redeem script wrapped in pay-to-witness-script-hash, with exactly two spend paths:

- **Claim.** The recipient reveals the SHA-256 preimage (`OP_SHA256`) and signs.
- **Refund.** The sender reclaims after the timelock (`OP_CHECKLOCKTIMEVERIFY`) and signs.

P2WSH is chosen for SegWit compatibility, deterministic txids and lower fees. The module handles no private keys — it produces unsigned transaction templates (PSBTs) for client-side signing, so the signing authority never leaves the counterparty.

The watcher learns the secret by reading the spending transaction's witness stack: it takes each 32-byte item, hashes it, and returns the one whose SHA-256 equals the hashlock. That is the mechanism by which a Bitcoin claim unlocks the paired EVM, Tron or Solana leg.

The Bitcoin script carries no fee logic. [The script's minimalism is the security argument](#) — the hashlock and the timelock are the only spend paths, reducing correctness to standard, individually auditable primitives. Where the taker funds the Bitcoin leg the fee is still paid on that leg, but outside the script: the fund builder returns `feePayTo` and `feeAmountSats` as a second output from the same wallet, and a fee below the 546-satoshi dust limit is waived. Where the taker funds the paired leg, the fee is taken there.

Off-chain settlement

Settlement happens on-chain; the off-chain services orchestrate the lifecycle, watch the chains and relay preimages. They never hold funds, and they hold no key that can move escrowed value.

What the service hands back is unsigned

This is the custody contract, and it is a property rather than a policy. [The fund, claim and refund endpoints return unsigned transactions](#). The party signs with their own key — a wallet, or an HSM — and broadcasts; the server holds no private key on any rail, and until the initiator reveals the preimage on chain it has only `sha256(secret)`. There is no account on the Hashlock side whose compromise moves a counterparty's funds, because there is no key there to compromise.

Which addresses may receive money

An address that receives value is held to a stricter test than one that merely trades. The payout address — and on Bitcoin and Solana the refund address — must be a wallet the account signed in with, or one proven from a signed-in wallet session. A wallet proven with an API key widens what the account may trade; it never becomes a payout address. [The consequence is that a leaked API key cannot redirect anyone's money](#): it can read, quote and agree, but it cannot name a new destination for principal. Combined with the escrow's two fixed exits, there is no credential in the system whose theft reroutes a settled leg.

The keeper

The keeper is a server process that watches the chains and, where claim and refund are permissionless — the EVM rails, Tron and Solana — may submit them, so a leg settles even if the party it pays is offline. It pays the gas and never receives principal: a claim it submits pays the recipient fixed at funding, and a refund it submits pays the funder. There is no third destination for it to use. [On Bitcoin the keeper cannot act at all](#), because a Bitcoin spend needs the owner's signature — the counterparty signs their own claim, as step 6 of section 06 says, and if they never do, the leg refunds at its timelock.

Lifecycle states

A swap carries one of nine statuses: `agreed`, `initiator_funded`, `counterparty_funded`, `initiator_claimed`, `counterparty_claimed`, `refunding`, `refunded`, `expired`, `failed`. They are not a strict sequence. `counterparty_claimed` can appear before the initiator has claimed its own leg, and the initiator can still claim in that state — the status records what has been observed, and an unclaimed leg stays claimable until its timelock.

The binding check

Before a swap advances, the settlement service re-derives the escrow address from the agreement and compares it with what was funded. The derivation deliberately takes hashlock, recipient, token, timelock and initiator from the on-chain event — because varying any of those moves the real escrow and the prediction together, so binding them proves nothing — while the

salt and the fee come from the agreed row. The check fails closed: a swap does not advance on an escrow it cannot bind.

The reveal handlers deliberately do not

Once the preimage is public, the reveal handlers relay it regardless of any fee hold. The reasoning is in the source: refusing to relay after the secret is out costs a beneficiary their entire principal at timelock. After the reveal, an underpaid fee is a report rather than a block. This is an explicit, documented trade between fee enforcement and counterparty safety, resolved in favour of the counterparty.

Failure direction

If every Hashlock service stopped, locked funds would refund on their timelocks with no operator action. Liveness — the trade completing rather than refunding — depends on the watcher observing a reveal and relaying it in time. That is an availability dependency, not a custody one, and it is the correct direction for a settlement system to fail in.

13

Fees

The fee is collected in the same transaction that creates the escrow, so there is no state in which a trade has settled and the fee has not been collected, and no invoicing relationship to manage. The taker pays it, on top of the leg they fund and in that leg's asset, and it is routed to a treasury address held in the contract rather than locked with the escrowed funds. It is not refunded if the deal does not complete.

The rate is volume-tiered and sub-10 bps on notional, stepping down as a counterparty's settled volume rises. The financial plan is modelled at a flat 8 bps blended rate across the whole window; the volumes that would trigger the lowest tier sit far beyond the plan. A rebate parameter returns a share to referrers — 20% of gross fees in the plan — which converts brokers, desks and escrow operators into a distribution channel paid only out of revenue already earned.

Because the fee is part of the escrow's identity on Ethereum, zero is no longer a free choice: underpaying deploys a different escrow, which the binding check and the transaction builders never address. Section 05 states the condition that property depends on.

14

Hardening and audit

The contract surface is covered by static analysis, symbolic execution, fuzzing, mutation testing, invariant suites and a runtime monitor.

Layer	What it does
Slither	Static analysis over the Solidity surface
Halmos	Bounded symbolic execution of the core invariants
Echidna	Property-based fuzzing against the same invariants
Stryker	Mutation testing — verifies the tests would catch an injected fault
fast-check	Property-based testing of the off-chain lifecycle
Runtime monitor	Continuous invariant checking against deployed contracts

The Foundry suite carries unit, fuzz and invariant tests for both EVM rails —

`HTLCFactory.t.sol`, `HTLCFactory.fuzz.t.sol`, `HTLCInvariant.t.sol` and the matching `SharedHTLC` trio — and the Solana program has integration tests for both the native and SPL paths. The invariants under test are the two that matter: **funds cannot be double-claimed, and funds cannot be lost once locked.**

Audit status, stated precisely

Two internal audit rounds are complete. The external audit with QuillAudits is complete (October 2026). External audit spend is \$10K, with a further \$10K in a bug bounty pool — \$20K already spent.

These contracts are not formally verified. Halmos performs bounded symbolic execution and Echidna is a fuzzer; neither is a proof of correctness, and this paper does not claim one.

Several findings referenced throughout this paper — the decoy escrow, the underpaid fee, the zero-address burn, the unchecked preimage length, the Solana timelock parity — were caught by those rounds and by internal review, and the fixes are visible in the source with the reasoning attached. A reviewer should read that as the system working rather than as a defect list: the alternative is a codebase where those findings are still latent.

15

Distribution

A settlement protocol that requires a desk to adopt a new venue competes against inertia and usually loses. Hashlock runs a Telegram bot as a distribution channel and as where negotiation happens, **because institutional crypto OTC is already negotiated in Telegram groups.**

The documented ways in are three: the web app, the `/v1` REST API — with webhooks and a maker WebSocket feed at `wss://<host>/v1/ws` — and the MCP server, hosted at `https://hashlock.markets/mcp` or run locally over stdio. Mainnet is live and non-custodial.

The MCP server is what makes a machine counterparty a first-class user rather than an integration project: an agent holding its own keys reaches the same negotiation and the same escrow as a desk on the web app, and can run request, negotiation, funding and claim without a person in the loop.

16

Assets and listing

Two classes of asset settle through the same escrow. **Curated** assets are the native coins and the stablecoins Hashlock runs itself. **Listed** assets are tokens that have passed automated checks; they are shown as unverified, beside their contract address, so a taker can see exactly what they are agreeing to. [A listed token settles through the same contracts as a curated one](#) — listing decides which assets are offered, not how they settle, and nothing in the preceding sections changes because a token is listed rather than curated.

What a listing has to pass

Two kinds of check. **Safety:** GoPlus Security, plus Hashlock's own on-chain checks, including a simulated transfer from a real holder — a token that cannot actually be moved by the people who hold it fails here rather than at settlement. **Traction:** published thresholds, applied mechanically.

Bar	Threshold
Market capitalisation	$\geq \$1,000,000$
Liquidity	$\geq \$100,000$
24-hour volume	$\geq \$250,000$
Holders	≥ 500
Age of the first pair	≥ 24 hours
Top-10 holder concentration	$\leq 40\%$

Listed tokens are re-checked daily. A token closes when it falls below the bar, when a safety check fails, or after three months without a trade on Hashlock. Closing stops new swaps; swaps already agreed on that token settle normally, which is the only safe direction — a live escrow is not something a listing decision can withdraw.

On Solana only classic SPL mints are listable. **Token-2022 is refused**, for the reason section 10 states: a transfer-fee extension would deliver less than `amount` into the escrow. The limit is enforced at listing rather than discovered at funding.

Known limits

These are the constraints we would raise ourselves in technical diligence. Most are reproduced from comments the engineers wrote for the next reader, not discovered by an outside reviewer.

The free-option problem

An initiator who has seen the counterparty fund can decline to reveal and let both legs refund — a costless option on price over the timelock window. Inherent to HTLCs, not to this implementation. Short windows bound it; collateral or reputation would price it. Neither is in V1.

Terms-derived identity has a boundary

It stops our software from being pointed at the wrong escrow. It does not stop two consenting parties from settling some other way on-chain and calling it this swap, and it does not survive an off-chain derivation that takes the fee from the event instead of the agreement.

Finality is a parameter

The 30-minute floor prevents the pathological case; it does not calibrate for a Bitcoin leg's confirmation window or a fee spike. The deployed policy counts a Bitcoin leg as funded at 3 confirmations, roughly 30 minutes, and runs its timelock at 24 hours against a 4-hour leg on the other side — a schedule, not a proof. Windows must be set per chain pair. A chain halt on the slower leg is not mitigated in V1 and would surface as a refund rather than a loss.

Solana rent and Token-2022

Escrow accounts are never closed, so rent — roughly 0.0016 SOL, plus 0.002 SOL for a token account — is never returned to the funder. Token-2022 mints are unsupported. A dust-sized native claim to a fresh address fails on a real cluster and the swap must refund; the test harness cannot catch that class of failure.

Tron rotation is operational

Pool rotation is orchestrated off-chain rather than enforced on-chain. The frozen pool continues to honour claims and refunds, so in-flight swaps are safe, but the sequencing is a procedure rather than a property.

Not formally verified, and coverage is uneven

Bounded symbolic execution and fuzzing raise confidence sharply but prove nothing about unbounded state. Coverage is deepest on the EVM rails, mirrored on Tron, integration-level on Solana, and argued from minimalism rather than demonstrated by a suite on Bitcoin. These are not equivalent levels of assurance and should not be presented as such.

Mainnet

Settlement is live on mainnet across six chains and the protocol fee is charged. Testnet verification and signet validation are meaningful evidence about correctness; they are not evidence about behaviour under real value, real adversaries and real fee markets.

Summary

The primitive is not the contribution. Hashed timelock contracts are a decade old. The contribution is a six-chain implementation in which the escrow's address is a function of the agreed terms, so that the software around it cannot be induced to settle against anything else — implemented as a CREATE2 salt, a storage slot key and PDA seeds, with a Bitcoin script whose minimalism is its own argument.

Around that sit the decisions that only appear once a system meets real assets: paying the held balance rather than the stated amount so a fee-on-transfer token cannot brick an escrow, measuring a funding delta because a predictable address can receive a donation, accepting Tron's non-conforming USDT return data, fixing the preimage at 32 bytes because a longer secret on one rail is a stranded principal on another, and relaying a revealed preimage even when the fee is short because the alternative costs a counterparty everything.

The guarantee is narrow and precise: both legs settle, or both refund. Everything above exists to keep that true under conditions less forgiving than a diagram.

Hashlock Markets • Delaware C-Corp • File #10561489 • Confidential; provided for evaluation only. Technical statements describe the implementation on the audit-r3 branch as of October 2026 and are subject to change. Figures describing the financial plan are drawn from the company's financial model and are forward-looking.