CRYPTO25 — A Minimal, Verifiable, In-Kind Crypto Index Vault

A tokenised index whose shares are a pro-rata claim on raw token balances, not on an oracle price.

Contract	`IndexFund.sol` — 548 lines, 329 excluding comments and blanks
Share token	`CRYPTO25` / `C25` (mainnet default); `eSEP` on the live Base Sepolia deployment
Target chain	Base (EVM L2). Bytecode compiled for `evm_version = paris`, portable to any EVM chain
Version	1.0 — 21 September 2026
Status	Feature-complete; live and exercised on Base Sepolia; **pre-audit, never deployed to mainnet**
Verification	140 tests / 19 suites · 100% line coverage on `src/` · 2 symbolic proofs · 6 stateful invariants · 4 killed mutants

> **Status disclosure.** This system has not been independently audited and has never held real money. Its index composition is not final — several constituents are unresolved blockers recorded in §12. Nothing here is an offer, a solicitation, investment advice or legal advice.
>
> **Regulatory disclosure.** A public "deposit stablecoins, receive basket exposure" vault is likely a regulated security or collective-investment scheme in many jurisdictions, including under MiCA and AIFMD in the EU. Counsel must be engaged before any public launch. See §16.

Contents

Part I — Executive overview (non-technical) 1. Summary · 2. The problem · 3. What the product is · 4. The one idea · 5. What a holder is asked to trust · 6. Where the project stands

Part II — Protocol specification 7. Architecture · 8. Share accounting · 9. The three money-path operations · 10. Rebalance, fees, governance · 11. The oracle layer

Part III — The index 12. CRYPTO25 construction and status · 13. Asset admissibility

Part IV — Verification 14. What was proved, and how

Part V — Operations 15. Deployment, application layer, and the path to mainnet

Part VI — Risk 16. Risks and limitations

Appendices — A: external interface · B: verified addresses · C: glossary

---

Part I — Executive overview

1. Summary

CRYPTO25 is an on-chain index fund. Holders own a share token whose value tracks a fixed basket of crypto assets — a market-cap-weighted crypto index plus a yield-bearing stable sleeve — and they can exit at any time, permissionlessly, for their exact pro-rata slice of everything the vault holds.

What distinguishes it from the existing on-chain fund products is not the index. It is the **structure of the claim**. Almost every tokenised fund prices its portfolio with a price oracle and mints or redeems shares against that price. That decision puts an oracle on the critical path: a stale, wrong, or manipulated feed can misprice a share, and a misprice is an extraction of value from everyone else.

CRYPTO25 removes the oracle from the money path entirely. A share is a claim on **raw token balances**. Deposit and redemption are **in-kind**: you put in a slice of every asset and receive shares; you burn shares and receive a slice of every asset. No price is consulted, because none is needed. The vault is therefore solvent *by arithmetic* rather than by assumption — there is no NAV number in the system for anyone to attack.

The design is optimised, in strict priority order, for:

1. **Solvency** — the vault can always honour every share outstanding.
2. **Minimal trust surface** — the operator can do very little, and holders can always exit.

3. **Verifiability** — the core is small enough to read in an afternoon, and its safety properties are machine-checked rather than asserted.

Everything else, including deposit convenience, is deliberately subordinate to those three and lives outside the verified core.

2. The problem

On-chain fund products fail in a small number of recognisable ways, and all of them are structural rather than accidental.

Oracle dependency on the money path. If shares are minted against an oracle-derived Net Asset Value, then anyone who can move, stall, or misreport that oracle can mint shares cheaply or redeem them expensively. The mitigation is usually operational — circuit breakers, deviation bounds, TWAPs, multiple providers — which reduces the probability of the failure without removing it. Meanwhile a feed that is merely *stale*, with no attacker at all, freezes the product.

Operator discretion. Vaults commonly grant an operator the power to rebalance into arbitrary tokens, to pause, or to upgrade the contract. Each of those is a path by which funds can be lost or trapped, and each must be trusted rather than verified. Upgradeability in particular means that nothing a reader verifies today is guaranteed to be true tomorrow.

Exit risk. A holder's ability to leave is frequently conditional — on a pause flag, on an oracle being fresh, on a queue, on the operator's cooperation. Conditional exit is the property whose failure has done the most damage in this sector.

Asset-set mutability. If the operator can add assets after deployment, then no verification of the asset set survives deployment. A single malicious or defective token admitted later can compromise the whole vault.

CRYPTO25 addresses these four structurally, not operationally. The oracle is removed from the money path; operator powers are enumerated and bounded; exit is unconditional and ungated; and the asset set is immutable from the constructor onward.

3. What the product is

A holder of C25 owns a proportional slice of a basket held in a single smart contract. The intended basket is **25 positions in two sleeves**:

- **Sleeve A — 70% of NAV** — a 22-asset market-cap-weighted crypto index with a 30% single-asset cap. Bitcoin and Ether are roughly 47% of the sleeve, so a small basket tracks the broad market closely while remaining liquid enough to rebalance.
- **Sleeve B — 30% of NAV** — yield-bearing stable positions. Because the claim is on raw balances, their yield compounds into the holder's claim automatically, with no action by anyone and no fee on the accrual.

That second point is a genuine structural advantage. A rebasing token like Aave's aUSDC grows its `balanceOf` with no transfer event and no vault action; an ERC-4626 vault share holds its count constant while its price rises. In both cases a claim on raw balances captures the yield for free. An oracle-NAV design has to model and price that accrual; this one simply holds it.

Three ways to hold the product:

Path	Who it is for	Trust
In-kind deposit	Anyone already holding the basket; market makers; institutions	Trusts only the verified core
Zap deposit (USDC in)	Retail convenience	Trusts one auditable periphery contract for the duration of one transaction
Secondary purchase	Anyone	Trusts the venue

Exit is always the same: burn shares, receive the basket, no permission required.

4. The one idea

Everything in the design follows from one decision, so it is worth stating precisely.

A share is a claim on raw token balances, and both sides of the claim round in the vault's favour.

When you deposit, the vault pulls, for every asset i:

```
amountIn_i = ceil( balance_i × shares / totalSupply )        ← rounded UP
```

When you redeem, the vault pays, for every asset i:

```
amountOut_i = floor( balance_i × shares / totalSupply )      ← rounded DOWN
```

Three consequences follow, and they are the whole product:

1. **Solvency is arithmetic.** Every share is backed by tokens the vault actually holds, because shares are only ever minted against tokens that were actually received. There is no scenario where the sum of all claims exceeds the assets. It does not depend on a price being right, a feed being live, or an operator behaving.
2. **No oracle is needed to move money.** `deposit` and `redeem` read no price feed. They succeed when every oracle in the system is dead, reverting, or reporting a negative price. This is not an aspiration; it is verified directly (property P5, §14).
3. **The round trip is never profitable.** Deposit-then-redeem of the same shares returns strictly no more than went in, on every asset, for every amount. This closes the entire family of inflation, donation, and rounding attacks that afflict share-based vaults (property P2, §14).

An oracle-derived NAV *view* exists, for dashboards and reporting. It is never read by any state-changing function, and if it reverts because a feed went stale, nothing about a holder's ability to deposit or exit changes.

The price oracle appears in exactly **one** place in the entire system — inside `rebalance`, and there only as a defensive upper bound on how bad a trade the operator is allowed to execute.

5. What a holder is asked to trust

The honest way to present a trust model is to enumerate the powers that exist, not the ones that were removed. The admin is a single address, intended to be a Gnosis Safe, optionally behind a 48-hour timelock.

Three powers constitute genuinely new trust:

1. **The management fee** — dilution, up to a ceiling fixed permanently at deployment (2% recommended, 10% absolute maximum). It starts at zero. It mints shares; it never moves assets; it cannot affect solvency.
2. **The deposit pause** — a guardian can stop *new deposits* instantly. It cannot stop exits.
3. **The adapter allowlist** — governance chooses which trading venues a rebalance may route through, bounded by a deviation cap that is immutable.

And the powers that do not exist at all:

The operator cannot	Why
Withdraw funds to itself	No such function exists — no sweep, no recover, no emergency withdrawal
Block redemption	`redeem` has no pause check, no oracle read and no admin gate. Verified as a stateful invariant
Change the asset set, the feeds, or the staleness bounds	Fixed in the constructor; no setter exists
Change the deviation cap or the fee ceiling	`immutable`
Upgrade the contract	There is no proxy. The deployed bytecode is final
Mint or burn shares outside seed/deposit/redeem/fee	No other mint path exists
Misprice a share	Rebalance output lands back in the vault; shares stay a claim on raw balances

Beyond the protocol, a holder also inherits the trust assumptions of the **tokens in the basket** — cbBTC is Coinbase custody, Universal's uAssets are Universal's, aUSDC is Aave's. The vault is only as trustworthy as its constituents, and each must be disclosed on its own terms. This is not a problem the contract can solve, and it is not one that should be papered over.

6. Where the project stands

Complete. The contracts, the oracle adapters, the trading adapters, the Zap, the deploy scripts, the test and verification suite, a wallet-connected web application, a history indexer, and a Docker-based deployment. A full vault is **live on Base Sepolia**, launched, seeded, verified on Sourcify, and exercised end-to-end on-chain — deposit via Zap, redemption, and admin rebalances in both directions through both a Uniswap V3 pool and an Aave reserve (§15).

Not complete, and required before mainnet. Two categories, neither of which is code:

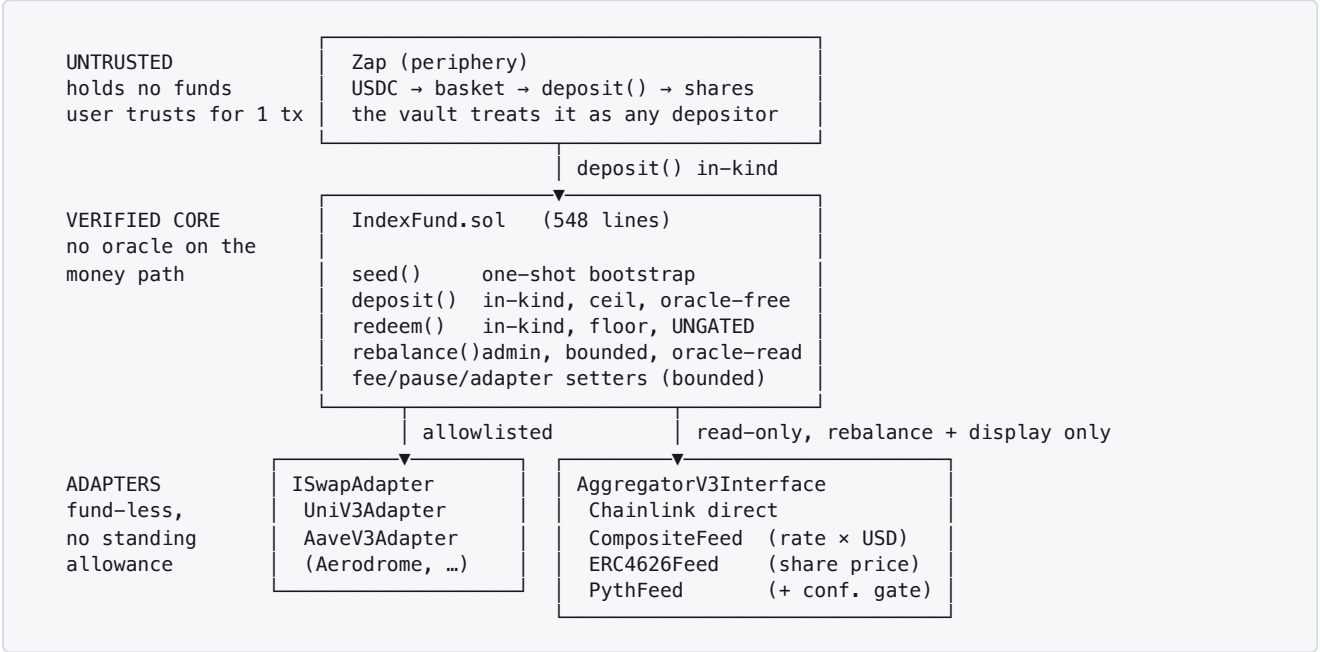
- **External review** — an independent security audit, a Certora specification, and legal sign-off.
- **Index composition** — a live on-chain sweep of every recorded address on 16 September 2026 found that several constituents are not deployable as recorded. sUSDS, a 30%-sleeve position, has no pricing path on Base at all. uLTC's feed is dead. Roughly sixteen sleeve-A token addresses were never captured. And a DEX-depth probe found that only BTC and ETH are comfortably rebalanceable at target fund size. These are decisions, not bugs — but because **the asset set is immutable after deployment**, none of them can be deferred past launch. They are set out in full in §12.

That the sweep found these problems, rather than a post-mortem finding them, is the intended function of the process. The finding is recorded here rather than smoothed over.

Part II — Protocol specification

7. Architecture

The system is deliberately stratified by trust. Each layer can only be as dangerous as the layer below permits.



Why the layering holds. An adapter is called only from inside the `rebalance` envelope, with a scoped approval created and cleared within the same transaction. It never holds funds and never holds a standing allowance. The worst a fully malicious allowlisted adapter can do is execute one swap at the deviation cap — a bounded, one-off loss, not a drain. The Zap, similarly, is just another in-kind depositor from the vault's perspective; nothing in it can affect solvency. And the oracle layer is read-only and confined to one function.

The practical consequence: **supporting a new trading venue is a new adapter contract, never a change to the verified core.** The same is true of a new pricing source. The core's audit surface does not grow as the system does.

8. Share accounting: the solvency argument

Let `B_i` be the vault's balance of asset *i* and `S` the total share supply.

Deposit of `s` shares pulls `ceil(B_i · s / S)` of each asset, then mints `s`. Because the pulled amount is rounded **up**, the vault receives at least the exact pro-rata amount, so backing per share is non-decreasing across every deposit.

Redemption of `s` shares burns first, then pays `floor(B_i · s / S)` of each asset, where `S` is the supply cached *before* the burn and `B_i` is untouched by it. Because the paid amount is rounded **down**, the vault retains at least the exact pro-rata remainder, so backing per share is non-decreasing across every redemption too.

From this, the three properties in §4 follow directly, and each is machine-checked:

	Property	How it is established
P1	`redeem` never returns more than `floor(B_i · s / S)`	Halmos symbolic proof + Foundry fuzz
P2	`deposit(s)` then `redeem(s)` is never profitable on any asset	Halmos proof + fuzz over the real `mulDiv`
Backing	Backing per share never decreases, across any operation	Stateful invariant over 4 actors

Two structural safeguards complete the picture:

- **The first-depositor problem.** The classic ERC-4626 inflation attack requires driving supply to zero or near-zero and then donating assets to skew the ratio. `seed()` permanently locks `MIN_LOCKED = 1000` shares at `0x…dEaD` at launch, so `totalSupply` can never return to zero (property P4). `seed()` is one-shot and reverts forever afterwards.
- **Donation is harmless by construction.** Sending tokens to the vault raises `B_i` and therefore raises the backing of every existing share pro rata. There is no ratio to skew, because the ratio *is* the product.

Strict checks-effects-interactions. `redeem` burns before paying out. `deposit` snapshots all required amounts against pre-deposit state before any external call, then pulls, then mints. Both are `nonReentrant`. The amounts a depositor is charged cannot be skewed by a token callback mid-pull.

9. The three money-path operations

`seed(initialShares, amountsIn)` — admin, one-shot

Bootstraps the vault. Requires every `amountsIn[i] > 0` and `initialShares > MIN_LOCKED`. Locks 1,000 shares at the dead address, mints the remainder to the admin, and starts the fee clock. It can never be called again — `AlreadySeeded` thereafter.

This sets the initial share price and is irreversible. It must be rehearsed against a fork with the exact production amounts before it is executed on mainnet.

`deposit(shares, maxIn)` — permissionless, pausable

```
if (depositsPaused) revert DepositsArePaused();
_accrueFee();                                  // settle fee first: ratio uses post-fee supply
for each asset i:
    amounts[i] = mulDiv(balance_i, shares, supply, Rounding.Ceil);
    if (amounts[i] > maxIn[i]) revert SlippageIn(i, amounts[i], maxIn[i]);
for each asset i: safeTransferFrom(msg.sender, this, amounts[i]);
_mint(msg.sender, shares);
```

`maxIn` is the caller's per-asset slippage ceiling — protection against the basket composition moving between quote and execution. **No feed is read.**

`redeem(shares, minOut)` — permissionless, ungated, unconditional

```
// deliberately NOT gated by depositsPaused
_accrueFee();
supply = totalSupply();
_burn(msg.sender, shares);                     // CEI: burn before paying
for each asset i:
    amounts[i] = mulDiv(balance_i, shares, supply, Rounding.Floor);
    if (amounts[i] < minOut[i]) revert SlippageOut(i, amounts[i], minOut[i]);
    safeTransfer(msg.sender, amounts[i]);
```

This is the most important function in the system and the shortest list of conditions in it. There is no pause check, no oracle read, no admin gate, no queue, and no redemption fee — the streaming management fee settles at the top of the call exactly as it does on deposit, and nothing additional is charged for leaving. A holder can exit while deposits are paused, while every feed is dead, and while governance is unreachable. The invariant `invariant_redeemAlwaysOpenEvenWhenPaused` asserts it continuously against a randomised handler.

10. Rebalance, fees, and governance

Rebalance — the only oracle use in the system

```
function rebalance(address adapter, address tokenIn, address tokenOut,
                   uint256 amountIn, uint256 minAmountOut, bytes calldata routeData)
    external onlyAdmin nonReentrant
```

The sequence, and why each step is there:

1. **Allowlist checks** — the adapter must be allowlisted; both tokens must be vault assets; they must differ.
2. **Oracle bound** — compute `oracleOut` from both USD feeds, derive `floor = oracleOut × (10000 − maxDeviationBps) / 10000`, and **revert if the caller's own `minAmountOut` undercuts that floor**. A stale or bad feed reverts here and the swap simply does not happen. The oracle can only ever *prevent* a trade, never price a share.
3. **Scoped approval** — approve exactly `amountIn` to the adapter, and reset to zero immediately after the call. No standing allowance ever exists.
4. **Measured deltas** — compute `spent` and `amountOut` from the vault's **own before/after balances**, never from the adapter's return value, and re-check `amountOut >= minAmountOut`. Property P6 pins this with an adapter that reports a complete swap while delivering 1%.

Output lands back in the vault, so shares remain a claim on raw balances throughout. The worst case for a fully rogue allowlisted adapter is a single swap at the deviation cap — with a default `maxDeviationBps = 100`, that is 1% of one leg, not of the fund.

Shipped adapters: `UniV3Adapter` (Uniswap SwapRouter02 on Base; `routeData` is a 32-byte fee tier for a single hop, or a packed V3 path for multi-hop) and `AaveV3Adapter` (a single immutable Aave reserve exposed through the same swap interface, so moving USDC into and out of the aUSDC yield position is a rebalance like any other, under the same envelope). A Solidly-style adapter for Aerodrome and an Algebra-style one for QuickSwap are straightforward additions.

An adapter must hold no funds, deliver `tokenOut` to `msg.sender`, and leave no standing approval. All three are asserted directly in the adapter test suites.

The management fee

The fee is a **streaming dilution**. `_accrueFee` mints shares to `feeRecipient` such that the recipient captures `managementFeeBps / 10000` of AUM per year, pro-rated to elapsed time:

```
feeShares = mulDiv(totalSupply, managementFeeBps × elapsed, 10000 × 365 days)
```

It mints only. It never moves an asset and never reads a feed, so solvency holds for any resulting supply — `redeem` still pays `floor(B_i · s / S)` of whatever supply exists. It settles at the top of both `deposit` and

`redeem`, and `accrueFee()` is a permissionless poke so a keeper can keep it fresh.

It is **zero at launch** and bounded by `maxManagementFeeBps`, which is `immutable` and itself capped by `MAX_FEE_CEILING = 1000` (10%).

> **Disclose the fee as a range, not a number.** Accrual is linear per call and the poke is permissionless, so more frequent pokes compound. The realised annual rate therefore sits between the simple rate and `e^rate − 1`. At a 2% ceiling that band is **2.0000% – 2.0201%** — about 2 basis points of AUM. It is bounded above by the immutable ceiling compounded and cannot affect solvency. Eliminating the gap entirely would require exponential accrual, which costs far more gas than the ~2 bps it saves. The bound is pinned by `test_fee_frequentPokeBoundedByE`.

Governance

`admin` is the only privileged address. Two supported configurations:

Setup	`admin`	Effect
Direct	the Safe	Setters take effect immediately. Appropriate for testnet
Timelocked	an OZ `TimelockController` whose sole proposer/executor is the Safe	Every setter and every `rebalance` goes through the delay (48h default). The timelock administers itself, so no standing admin role exists on it

Under the timelocked setup the **guardian remains the Safe**, so `pauseDeposits()` stays instant — pausing is the one power that is useless if delayed. Unpausing goes through the timelock, because resuming deposits is the more privileged direction. Because `seed()` is itself timelocked, seeding must be a timelock proposal rather than part of the deploy transaction.

11. The oracle layer

The core expects one `AggregatorV3Interface` USD feed per asset, plus a **per-asset** maximum accepted age. Per-asset rather than global, because feed cadences differ by orders of magnitude: the live sweep measured ETH/USD at 196 seconds old against a wstETH exchange rate at 4.7 hours. A single global bound would have to cover the slowest feed and would leave the fast ones dangerously loose.

`_readFeed` reverts `BadPrice` on a non-positive answer and `StalePrice` beyond that asset's bound. It reads `feed.decimals()` rather than assuming 8 — which matters, because Base's cbBTC/USD feed reports **18** decimals, uniquely among the seventeen Base feeds checked. The core handles it; any downstream dashboard or spreadsheet that hardcodes 8 is wrong by a factor of 10^{10} on the index's largest position.

Four pricing shapes are supported without any core change, because each presents the same interface:

Adapter	Used for	Mechanism
Chainlink direct	Most constituents	—
`CompositeFeed`	wstETH, Morpho	`rate × USD`, taking the **older** of the two timestamps and failing closed if either component is bad
`ERC4626Feed`	ERC-4626 vault shares	`convertToAssets` × underlying USD

Adapter	Used for	Mechanism
`PythFeed`	Assets with no Base Chainlink feed	Pyth price ID → 8-dp aggregator; `updatedAt = publishTime`; enforces a per-asset **confidence bound** (`maxConfBps`), failing closed on a wide band, because a wide band is Pyth signalling low confidence — typically exactly when a thin asset's venues have diverged

Pyth is **pull-based**: prices must be pushed on-chain by a keeper. A neglected feed cannot endanger holders — `deposit` and `redeem` are unaffected, and `rebalance` simply reverts — but it does disable rebalancing, so keeper liveness is an operational requirement with a monitoring obligation attached.

Part III — The index

12. CRYPTO25 construction and status

Construction

Sleeve A (70% of NAV) — 22 assets, market-cap weighted within the sleeve, 30% single-asset cap. cbBTC at 30.0% and wstETH at 17.4% are ~47% of the sleeve, which is what allows a 22-name basket to track the broad market while remaining liquid enough to trade.

Sleeve B (30% of NAV) — three yield-bearing stable positions: aUSDC (Aave V3), sUSDS (Sky), and a Morpho Steakhouse USDC vault. Their yield compounds into the claim automatically.

aUSDC is confirmed working as designed on Base: `UNDERLYING_ASSET_ADDRESS()` returns Base USDC, and `totalSupply / scaledTotalSupply = 1.145922` — that ratio *is* the accrued Aave liquidity index. `balanceOf` grows while the share count stays fixed, so the yield accrues to the vault's claim with no action by anyone.

On-chain verification and its findings

Every address in the index specification was read live from Base mainnet at **block 51394468, 2026-09-16T17:04:43Z**, via `description()`, `decimals()` and `latestRoundData()`, with the two staleness findings corroborated against a second independent RPC that returned byte-identical `updatedAt` values. Earlier revisions of the specification carried addresses transcribed from a directory and never read on-chain. Three were truncated and unusable; one pointed at a dead feed.

Transcription is not verification. That principle is now enforced by process.

Open composition blockers

These are unresolved and must be decided before deployment. Because the asset set is immutable, none can be fixed afterwards.

#	Blocker	Detail
1	**sUSDS has no pricing path on Base**	The token exposes no ERC-4626 interface — `asset()`, `convertToAssets()`, `totalAssets()`, `chi()` all revert; it is a 170-byte bridged proxy whose accounting lives on Ethereum mainnet. And no `sUSDS/USDS` rate feed exists on Base, so `CompositeFeed` has no rate leg. This is a **30%-sleeve position with no way to price it.** Options: source a rate oracle, substitute a natively-4626 yield asset, or drop it and redistribute sleeve B
2	**~16 sleeve-A token addresses were never recorded**	Only *feeds* were ever captured. Six are now verified on-chain (cbBTC, wstETH, LINK, AAVE, uSOL, uXRP); the remaining Universal uAssets must come from Universal's own documentation and be verified on-chain. **Not from a token-list aggregator** — a scrape of one returned an entirely different set of Universal assets, none of them constituents
3	**uLTC's Chainlink feed is 15 days stale**	Corroborated on two independent RPCs. Either Pyth-adapter it or drop LTC. This makes **8** Pyth assets, not 7
4	**The recorded USDC/USD feed was dead**	The correct live feed is `0x7e860098F58bBFC8648a4311b374B1D669a2bc6B`. The previously recorded address had not updated in 13 days. It is load-bearing in three places — the cash sleeve, aUSDC valuation, and the Morpho composite's quote leg — so a deploy using it would revert `StalePrice` on every rebalance and NAV read from day one

#	Blocker	Detail
5	**The wstETH basis needs an explicit decision**	The only Base rate feed self-describes as `wstETH–stETH Exchange Rate`, not wstETH-ETH. stETH ≈ ETH but is not identical; using it embeds that basis. Make it a decision, not an accident
6	**Thin rebalancing**	See below

Tradeable depth — the finding that changes the composition policy

A depth probe (`script/probe-depth.py`) swept Uniswap V3 across four fee tiers, Aerodrome v2 (stable + volatile) and Aerodrome Slipstream across five tick spacings, against both USDC and WETH, measuring **quote-side reserve** — the quantity that bounds a sell. Measured at Base block 51400159.

`SAFE LEG` ≈ 1% of quote reserve, roughly the largest single swap that stays inside the vault's own deviation cap. `LEGS` is how many such swaps a *full* turnover would need at each target fund size.

Asset	Depth (quote side)	Safe leg	Weight	Pos @ €5M	Legs	Pos @ €20M	Legs
cbBTC	$28,860,997	$288,610	30.00%	$1,134,000	**4**	$4,536,000	**16**
wstETH	$3,733,872	$37,339	20.86%	$788,508	21	$3,154,032	84
uXRP	$499,911	$4,999	11.75%	$444,150	89	$1,776,600	**355**
uSOL	$541,200	$5,412	7.79%	$294,462	54	$1,177,848	**218**
LINK	$173,837	$1,738	3.24%	$122,472	70	$489,888	**282**
AAVE	$212,505	$2,125	1.32%	$49,896	23	$199,584	94

Four conclusions, each with a consequence:

1. **Only BTC and ETH are comfortably rebalanceable on Base.** cbBTC turns over in 4–16 legs. Nothing else is close — and those two are already ~51% of sleeve A after oracle-based exclusions.
2. **Oracle availability and tradeable depth are uncorrelated.** LINK and AAVE — canonical blue chips with healthy direct Chainlink feeds — have *less* Base depth than uSOL and uXRP. Any policy that filters on feeds alone keeps assets it cannot trade and would have dropped assets it can.
3. **Liquidity concentrates on WETH, not USDC.** Almost every USDC-quoted pool for these assets is empty. Multi-hop routing through WETH is therefore the default assumption, not an exception. The shipped `UniV3Adapter` supports it via packed-path mode.
4. **Some depth is only reachable via Aerodrome Slipstream**, which `UniV3Adapter` cannot route to — uSOL's largest pool is $300k of its $541k. Reaching it requires a new adapter, with its own audit surface, or uSOL is excluded.

Why this decides inclusion, not merely execution quality. The vault has no admin withdrawal, and Universal's mint/redeem is an off-chain permissioned API that a contract cannot call. An asset with no Base DEX depth is therefore **permanently un-rebalanceable** — its weight drifts forever and can never be corrected — and the asset set is immutable, so there is no remedy after deployment.

This drives a **two-gate composition policy**, both gates run independently:

> A constituent must be **(a)** priceable by Chainlink on Base — directly or as a composite of Chainlink feeds — with every leg live and inside its heartbeat at deploy; **and (b)** tradeable, on a venue the vault has an adapter

> for, in the leg size its weight actually implies at target AUM, within `maxDeviationBps`. Both gates. No second oracle provider.

Gate (a) alone drops the seven feedless assets, uLTC and sUSDS. Gate (b) is unresolved for most of the remaining book.

Methodology limits, stated plainly. Quote-side reserve is an *upper bound* on tradeable size, not an estimate of it — concentrated liquidity means real capacity is lower, so the leg counts are optimistic. The 1% heuristic is order-of-magnitude, not a quote; real routing quotes at these sizes are required before deploy. And depth moves: this is one snapshot, and incentivised liquidity leaves quickly.

> **A correction, recorded deliberately.** An earlier pass of this probe reported uSOL at ~$267k with "$1 of Uniswap liquidity". That was wrong: the run used no retries, and a dropped RPC call made a $236k pool look nonexistent. The script now retries every call three times. A single failed read is not evidence of absence — the same failure mode produced a spurious "AAVE feed is dead" reading during the address sweep. Treat any single-RPC "it isn't there" result as unproven.

13. Asset admissibility

The asset set is immutable, so admissibility is decided **once, at deploy, by review**. The vault deliberately spends no gas re-checking it on every transfer. That makes it a process control — and one that is documented and tested rather than assumed. The failure modes are pinned in `test/AssetAdmissibility.t.sol`.

Token behaviour	Admissible?	Why
Standard ERC-20 (USDC, WETH, cbBTC, …)	✅	The intended case
Fee-on-transfer	❌ **never**	`deposit` mints against the amount it *asked* to pull, so a transfer fee leaves the vault short and dilutes every existing holder on every deposit. Measured in `test_asset_feeOnTransferDilutesHolders`
Fee switch present but currently **off**	❌ **never**	It can be flipped after the asset set is frozen. Demonstrated in `test_asset_dormantFeeSwitchIsAlsoInadmissible`
Rebasing up (aUSDC, sUSDS)	✅	The claim is on raw balances, so yield auto-compounds to holders with no action
Rebasing down (slashing)	⚠️ tolerable	Redemption stays pro-rata over what remains, so the loss is shared rather than front-runnable
ERC-4626 vault shares (Morpho)	✅	Share count constant, share price grows; price with `ERC4626Feed`
ERC-777 / transfer hooks	❌	`nonReentrant` blocks re-entry, but a callback token is an avoidable trust surface in an immutable set
Upgradeable token with a mutable transfer path	⚠️	Only if the upgrade key is itself acceptable to holders

The fee-on-transfer exclusion is absolute and applies to dormant fee switches. The vault does not and cannot detect this at runtime.

Part IV — Verification

14. What was proved, and how

Verification is layered, and each layer is matched to the tool that suits it. Because **solvency is structural**, most of the assurance comes from exercising the real accounting rather than from abstract proofs.

Measured on 21 September 2026:

Metric	Value
Tests	**140** across **19 suites** — 136 pass, 4 fork tests skip without an RPC
Line coverage on `src/`	**100.00%** (322/322)
Statement coverage	99.53% (420/422)
Branch coverage	93.18% (82/88) — `IndexFund.sol` alone: **97.67%**
Function coverage	100.00% (45/45)
Symbolic proofs	2 (Halmos)
Stateful invariants	6, over 16,384 randomised calls each
Mutants killed	4

Named properties

ID	Property	Method
P1	`redeem` never returns more than the pro-rata floor	Halmos `check_P1_redeemFloorBound` + Foundry fuzz
P2	`deposit(s)` then `redeem(s)` is never profitable on any asset	Halmos `check_P2_roundTripNonProfit` + fuzz over the real `mulDiv`
P3	Only the admin can call `seed` / `rebalance` / any setter	`IndexFund.guards.t.sol` (29 tests)
P4	`seed` is one-shot; `MIN_LOCKED` shares locked permanently; supply never reaches zero	Unit tests + stateful invariant
P5	`deposit` / `redeem` are oracle-independent — they succeed when every feed reverts, returns zero, or returns a negative price	Direct tests against hostile feed mocks
P6	`rebalance` trusts its own measured balance delta, never the adapter's return value	Test with an adapter that reports a full swap and delivers 1%

Stateful invariants

A handler drives the **full** surface — deposit, redeem, share transfers, admin rebalance through the adapter, time warps with fee accrual, fee-rate changes, and pause/unpause — from **four independent actors**, ledgering every token flow.

Invariant	What it rules out
`invariant_assetConservation`	Any leak or phantom backing: each balance is exactly `seed + deposits − redemptions + rebalance gains − rebalance spends`

Invariant	What it rules out
`invariant_supplyEqualsSumOfHolders`	A fee-minting or burn-accounting bug
`invariant_simultaneousFullExitIsCovered`	Over-commitment: if **every** holder redeemed at once, the sum of their actual pro-rata floors fits in the vault, for every asset
`invariant_redeemAlwaysOpenEvenWhenPaused`	A holder ever being trapped, whatever the pause flag says
`invariant_totalSupplyNeverBelowMinLocked`	Supply returning to zero
`_assertBackingNonDecreasing`	Backing per share falling across any deposit or redeem

> **`fail_on_revert = true` is load-bearing.** With it set to `false`, a reverting handler call is silently discarded — which also swallows the handler's own assertions and lets a broken vault "pass" by reverting its way out of every bad state. This was verified, not assumed: mutating `redeem`'s rounding from `Floor` to `Ceil` left every invariant green under `false` and fails immediately under `true`.

Mutation testing — evidence the suite can actually fail

A test suite that cannot fail proves nothing. The invariants are run against deliberately broken builds. Each of these is caught:

Mutation	Caught by
`redeem` rounds `Ceil` instead of `Floor`	backing per share decreased
`deposit` rounds `Floor` instead of `Ceil`	backing per share decreased
Fee minted to the wrong address	`totalSupply` ≠ Σ holder balances
`redeem` gated by `depositsPaused`	`DepositsArePaused` surfacing from the handler

A deliberate scope decision on formal verification

We do **not** attempt to symbolically prove the round trip over OpenZeppelin's `Math.mulDiv`. That function is audited, ubiquitous code, and the proof is intractable for current SMT solvers — the nonlinear bitvector multiplication times out z3 and bitwuzla alike. Re-proving widely-used audited primitives is a poor use of effort, and claiming a proof that timed out would be worse.

The Halmos checks therefore prove the property that is *ours* — that ceil-on-deposit plus floor-on-redeem makes a round trip non-profitable — reduced to a linear fact the solver discharges instantly. Fuzzing exercises the real `mulDiv` end-to-end over thousands of cases. For a machine-checked proof over the full contract storage model, the recommended next step is a **Certora** specification of the same invariants — the pre-mainnet gold standard, and an open item.

The swap *outcome* is deliberately not formalised, because it depends on external DEX state. It is bounded with fuzzing and with the oracle-deviation envelope instead.

Testing strategy

The design makes testing tractable despite the objection that "you cannot get every asset liquid on one DEX", because **`deposit`, `redeem` and `seed` touch no DEX at all.** DEX liquidity is needed only by `rebalance`, and

`rebalance` handles one pair at a time — never a pool holding the whole basket. So testing splits cleanly:

Tier	Covers	DEX needed
0 — unit/fuzz/invariant	Money path, rounding, access control, oracle independence, fee, pause, every revert branch, conservation	None (mock ERC-20s at 6/8/18 dp)
0 — symbolic	P1/P2 rounding facts	None
0 — mutation	That the invariants can actually fail	None
1 — adapters + periphery	Route dispatch, approval reset, fund-less-ness, lying-adapter rejection, zap shortfall and dust	Mock router
1 — sleeve B	Rebasing and ERC-4626 auto-compounding; admissibility failure modes	None
2 — fork	In-kind deposit/redeem with real tokens, real decimals, real Chainlink NAV	Base fork
3 — fork	`rebalance` through the **real** Uniswap V3 USDC/WETH pool	Real pool

Tiers 2 and 3 run in CI as a separate job that **requires** an RPC secret and fails loudly if it is missing — they are never allowed to skip silently.

> **An `evm_version` footgun worth knowing.** The default profile builds at `paris` so deployed bytecode has no `PUSH0` and stays portable across L2s. But `evm_version` also selects the EVM the *test runner* uses, and Base's live Chainlink aggregators and SwapRouter02 are cancun-era bytecode. Under `paris` they revert `EvmError: NotActivated` before reaching any of our code — which silently turned the entire fork-test file red while `forge test` stayed green, because fork tests self-skip without an RPC. A separate `fork` profile sets `cancun`, and the tests now assert the profile themselves. A further subtlety: Foundry applies the *forked chain's* spec while `setUp()` runs, then executes each test body under the configured `evm_version` — the same probe reports "activated" in `setUp` and "not activated" in the test body of the same run. The guard therefore lives in a modifier, not in `setUp`.

Gas at full basket (n = 25)

Call	forge 1.7.1	forge 1.8.3	Spread	CI ceiling
`deposit`	614,848	649,216	5.6%	790,000
`redeem`	531,259	562,527	5.9%	700,000
`navUsd1e18` (view, display only)	612,940	612,940	0%	770,000

`deposit` and `redeem` loop over every asset, so cost scales with basket size. On Base these are negligible in absolute terms. The ceilings are regression guards, not targets.

> Gas here is toolchain-sensitive, so the ceilings carry ~25% headroom. Storage warmth carried across calls within a single test body in forge 1.7 but not in 1.8: with identical bytecode, `redeem` first measured 163,162 on one and 562,530 on the other — a 3.4× swing that broke CI. The tests now do all setup in `setUp()` so every measurement is deterministically cold, which cut the cross-version spread to under 6%, and CI pins an exact forge version. The ceilings catch a 2× regression — an oracle read finding its way back onto the money path — and will not catch 5% drift, by design.

Dependency integrity

`lib/` is **vendored**, not submoduled, so a checkout is self-contained and needs no `forge install`. Versions and integrity digests are recorded in `DEPENDENCIES.md` and verified in CI by `script/check-deps.sh`.

Part V — Operations

15. Deployment, application layer, and the path to mainnet

The live Base Sepolia deployment

A complete vault is **live on Base Sepolia (chain 84532)**, launched at block 46948164 with all 22 launch transactions confirmed. It is not a dry run:

- **Four seed assets**, 100 shares outstanding, zero management fee, deposits unpaused.
- **Two real Uniswap V3 pools** created and funded (WETH/USDC and WBTC/USDC), both with nonzero active liquidity.
- **All five application contracts verified** on Sourcify with exact matches.
- **A full smoke sequence executed on-chain** at blocks 46953620–46953624: USDC approval, a four-asset Zap deposit including Uniswap purchases and an Aave supply, a basket redemption, and admin rebalances in both directions through the Aave reserve. Total execution cost: 0.0000096 ETH.

One detail is worth reporting because it is the system behaving correctly under adversity. The first live smoke attempt **refused to rebalance**, because the dollar feed was more than an hour old. That is the designed safety behaviour, observed in production conditions rather than in a test. A second run that refreshed the feed as the authorised updater first succeeded. Feed refresh must run every 15 minutes to stay comfortably inside the one-hour limit — an operational requirement that the testnet surfaced before mainnet could.

Still outstanding on testnet: operational alerting, and a 48-hour observation window.

Application layer

The user-facing stack is a Next.js application with a Ponder indexer supplying history, deployable with Docker Compose. Several of its design decisions are load-bearing rather than incidental, and they are worth recording because each encodes a way the naive implementation would lie to users:

- **Snapshots, not log replay.** aUSDC rebases — its balance grows with no `Transfer` event — so vault composition *cannot* be reconstructed from events. A block handler reads `balances()` on-chain.
- **NAV is nullable everywhere.** `navUsd1e18()` reverts whenever any feed is past its per-asset staleness limit. Charts render **gaps**; nothing interpolates across them, and no invented USD value is ever substituted. A separate, clearly-labelled reconstruction from the feeds that *were* readable is offered under its own name.
- **Staleness is measured on chain time**, never `Date.now()`. The contract compares `block.timestamp - updatedAt`; using the browser clock makes the UI disagree with the vault whenever the two drift, and a feed the vault considers dead would still render as fresh.
- **Two distinct deposit previews.** `deposit()` accrues the fee first and divides by the post-fee supply, while the Zap budgets against the pre-fee `totalSupply()`. The two preview functions are deliberately separate and are unit-tested against each other.
- **User history comes from `Transfer`, not `Deposit`.** A Zap deposit emits the fund's `Deposit` with the *Zap* as caller, then delivers shares by transfer.
- **The RPC proxy is read-only.** It rejects `eth_sendRawTransaction`; wallets sign and broadcast themselves.

End-to-end tests run under Playwright against an anvil fork of Base Sepolia, impersonating the real on-chain admin rather than holding any key — so the admin console and the stale-feed paths are exercised against the actual deployed configuration.

Deployment

`script/Deploy.s.sol` is fully env-driven; nothing is hardcoded.

Constructor argument	Notes
`admin`	The Safe, or a `TimelockController` it controls
`assets[]`	Canonical order; **must include USDC**. This order is canonical for every array argument
`feeds[]`	USD feed per asset, same order — Chainlink or one of the oracle adapters
`stalenesses[]`	**Per-asset** max oracle age, same order, each set from its feed's real heartbeat
`maxDeviationBps`	Max executed-vs-oracle deviation. Default 100 (1%). **Immutable**
`guardian`	May pause deposits only. Defaults to admin
`feeRecipient`	Receives streamed fee shares
`maxManagementFeeBps`	**Immutable** fee ceiling, default 200 (2%). The fee itself starts at 0
`adapters[]`	Optional; `rebalance` reverts until at least one exists

Then **seed once**: approve each asset and call `seed(initialShares, amountsIn)`. This locks 1,000 shares at the dead address and mints the rest to the admin. It can never be called again, and there is no admin off-ramp afterwards.

The path to mainnet

Everything below is external work. None of it is code, and none of it can be shortened by writing more.

- [ ] **Independent security audit**, plus a **Certora specification** re-stating the invariants of §14 over the full contract storage model.
- [ ] **Legal sign-off** — MiCA / AIFMD analysis and jurisdiction-by-jurisdiction assessment.
- [ ] **Resolve every composition blocker in §12** — sUSDS, the missing token addresses, uLTC, the wstETH basis, and the two-gate policy applied to the full book.
- [ ] **Re-verify every live address on-chain** immediately before deploy. Feeds get deprecated.
- [ ] **Set per-asset staleness and Pyth confidence bounds** from each feed's real measured heartbeat, not from defaults.
- [ ] **Stand up a Pyth keeper** and feed-liveness monitoring, with alerting.
- [ ] **Obtain real routing quotes** at the leg sizes in §12, replacing the reserve-based heuristic.
- [ ] **Decide on an Aerodrome Slipstream adapter**, or treat Slipstream-only liquidity as unavailable when applying gate (b).
- [ ] **Rehearse the seed** against a fork with the exact production amounts. It is one-shot and irreversible.
- [ ] **Sleeve-B capacity reads** — confirm the Aave and Morpho supply caps absorb the intended size.

Part VI — Risk

16. Risks and limitations

Read this section before forming a view on the product. It is written to be complete rather than reassuring.

1 · Regulatory — the largest single risk. A public "stablecoins in, basket exposure out" product is very likely a regulated security or collective-investment scheme in many jurisdictions, including under MiCA and AIFMD. This is not a smart-contract problem and cannot be engineered away. Counsel must be engaged before any public launch.

2 · Not audited; never deployed to mainnet. This is a reference implementation with an unusually strong verification suite, exercised live on a public testnet. It has not been independently audited and has never held real funds. An audit — and ideally the Certora specification — is required first.

3 · Bridge and wrapper trust. Every wrapped asset inherits its issuer's or bridge's trust assumptions: cbBTC is Coinbase custody, Universal's uAssets are Universal's, aUSDC is Aave's. The vault is only as trustworthy as the tokens in the basket, and each must be disclosed per asset. The protocol cannot mitigate this; it can only be honest about it.

4 · Asset admissibility is a process control, not a code check. The asset set is immutable and is validated once, by review, at deploy. **Fee-on-transfer tokens — including those whose fee switch is currently off — must never be included.** The vault does not and cannot detect this at runtime.

5 · Oracle discretion inside rebalance. Although off the money path, a wrong-but-fresh oracle *combined with* a compromised or colluding admin could still execute an unfavourable — but bounded — rebalance. The deviation cap bounds the damage; it does not eliminate admin discretion within that band.

6 · Oracle operations. Per-asset staleness and Pyth confidence bounds must be set from real measured heartbeats rather than left at defaults, and Pyth is pull-based, requiring a keeper and liveness monitoring. A neglected feed cannot endanger holders — the money path is unaffected and `rebalance` simply reverts — but it does disable rebalancing, which means weights drift.

7 · The management fee is a range, not a number. Accrual is linear per call and the poke is permissionless, so the realised rate sits between the simple rate and `e^rate − 1` — at a 2% ceiling, 2.0000%–2.0201%. Bounded and solvency-neutral, but it must be disclosed as a band.

8 · Rebalancing liquidity. Rebalancing thin assets safely requires real DEX depth, and the probe in §12 found that only BTC and ETH have it at target size on Base. An asset with no depth is permanently un-rebalanceable, and the asset set is immutable. **This is a composition constraint, not an execution-quality constraint**, and it is the most consequential finding in this document.

9 · Deposit UX requires trusting the Zap. Pure in-kind deposit requires holding the whole basket already. The USDC entry path is the Zap, deliberately outside the verified core. The vault does not trust it and it holds no funds across transactions, but the *user* trusts it for the duration of one transaction. A USDC-NAV deposit path remains deliberately absent from the core — adding one would drag the oracle back onto the trust-critical path and defeat the entire architecture.

10 · Seed is one-shot and irreversible. It sets the initial share price and can never be re-run.

11 · Market risk. The product tracks a basket of volatile crypto assets. Nothing in this design protects against the basket falling in value. Solvency means every share is fully backed by the tokens held — it says nothing

whatsoever about what those tokens are worth.

12 · Concentration. After the oracle and liquidity gates are applied, BTC and ETH are roughly half of sleeve A and a larger share of what is actually tradeable. Holders should understand the product as a concentrated large-cap index with a stable yield sleeve, not as broad diversification.

Appendices

Appendix A — External interface

User functions (permissionless)

Function	Notes
`deposit(uint256 shares, uint256[] maxIn)`	In-kind, oracle-free, rounds up. Pausable
`redeem(uint256 shares, uint256[] minOut)`	In-kind, oracle-free, rounds down. **Never gated**
`accrueFee()`	Permissionless poke of the fee checkpoint
`transfer` / `approve` / …	Standard ERC-20 on the share token

Views

Function	Notes
`assets()` / `assetsLength()`	The immutable asset list, in canonical order
`balances()`	Raw vault balance of each asset, same order
`pendingFeeShares()`	Fee shares that would mint if poked now
`navUsd1e18()`	**Display only.** Reverts if any feed is stale or bad. Never read internally
`stalenessOf(address)` · `feedOf(address)` · `isAsset(address)` · `isAdapter(address)` · `adapters()`	Configuration reads

Admin functions

Function	Bound
`seed(uint256, uint256[])`	One-shot
`rebalance(adapter, tokenIn, tokenOut, amountIn, minAmountOut, routeData)`	Allowlisted adapter + `minAmountOut` + oracle-deviation floor
`addAdapter` / `removeAdapter`	Allowlist only; never a standing allowance
`setManagementFee(uint256)`	≤ `maxManagementFeeBps` (immutable)
`setFeeRecipient(address)`	Settles to the old recipient first; non-zero
`setGuardian(address)`	Guardian may only pause deposits
`pauseDeposits()`	Guardian **or** admin
`unpauseDeposits()`	Admin only

Immutables and constants

`admin` · `maxDeviationBps` · `maxManagementFeeBps` · `stalenessOf[]` · `feedOf[]` · the asset list · `MIN_LOCKED = 1000` · `DEAD = 0x…dEaD` · `BPS = 10000` · `SECONDS_PER_YEAR = 365 days` · `MAX_FEE_CEILING = 1000` (10%)

Errors

`NotAdmin` · `ZeroAddress` · `LengthMismatch` · `DuplicateAsset` · `BadParam` · `AlreadySeeded` · `NotSeeded` · `InsufficientInitialShares` · `ZeroAmount` · `ZeroShares` · `SlippageIn` · `SlippageOut` · `NotWhitelisted` · `SameToken` · `DeviationTooHigh` · `StalePrice` · `BadPrice` · `FeeTooHigh` · `NotGuardianOrAdmin` · `DepositsArePaused` · `NotWhitelistedAdapter`

Appendix B — Verified Base mainnet addresses

Read live at **block 51394468, 2026-09-16T17:04:43Z**. Re-verify before any deploy; feeds get deprecated.

	Token	USD feed
USDC (cash sleeve)	`0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913`	`0x7e860098F58bBFC8648a4311b374B1D669a2bc6B`
cbBTC	`0xcbB7C0000aB88B473b1f5aFd9ef808440eed33Bf` (8 dp)	`0x3f335B20039774a46EAA7D33D762A2f3A1184FD1` — ⚠ **18-dp feed**
wstETH	`0xc1CBa3fCea344f92D9239c08C0568f6F2F0ee452` (18 dp)	`CompositeFeed(0xB88BAc61…, 0x50015f8b…)`
aUSDC (Aave V3)	`0x4e65fE4DbA92790696d040ac24Aa414708F5c0AB` (6 dp)	USDC/USD direct
uXRP	`0x2615a94df961278DcbC41Fb0a54fEc5f10a693aE` (18 dp)	`0x440A3800e459E3BC77df690D17c718651718344e`
uSOL	`0x9B8Df6E244526ab5F6e6400d331DB28C8fdDdb55` (18 dp)	`0x43151D66D653aD888Bf909F006A01792d5f33243`
LINK	`0x88Fb150BDc53A65fe94Dea0c9BA0a6dAf8C6e196` (18 dp)	`0x6B29CEd17Bd1Fc3e1dA6D82aCA5963e9111660d2`
AAVE	`0x63706e401c06ac8513145b7687A14804d17f814b` (18 dp)	`0x4b6F092e0e13B94fFAF2C59aAbDEb85a5342e9C1`

Composite legs

Leg	Address	dp	Read as	On-chain `description()`
wstETH rate	`0xB88BAc61a4Ca37C43a3725912B1f472c9A5bc061`	18	1.244176	⚠ `wstETH-stETH Exchange Rate`
ETH/USD	`0x50015f8b17fb2C290Dde41fDc246ed0dcEE93a8b`	8	$2,388.68	`ETH / USD`
USDS/USD	`0x2330aaE3bca5F05169d5f4597964D44522F62930`	8	$0.999915	`USDS / USD`
sUSDS rate	**does not exist on Base**	—	—	—

Base Sepolia (84532) — one real token and three feeds exist, which is why the testnet script deploys its own mocks and pools. WETH is the `0x4200…0006` predeploy; USDC, cbBTC, wstETH, sUSDS and uSOL have **zero code** at their mainnet addresses. The Sepolia USDC/USD feed has a **24-hour heartbeat** and is routinely ~20 hours old, so it needs `STALENESSES = 86400`, not `3600`. Uniswap SwapRouter02 is at a different address than on mainnet.

Appendix C — Glossary

Term	Meaning
In-kind	Depositing or redeeming a slice of *every* asset, rather than a single asset priced against the others
NAV	Net Asset Value — the USD value of the portfolio. Here, display only, never on the money path
Sleeve	A sub-portfolio with its own mandate. Sleeve A is the index; sleeve B is yield-bearing stables
bps	Basis point — 1/100th of a percent
Staleness bound	Maximum accepted age of an oracle reading, per asset, fixed at deploy
Deviation cap	`maxDeviationBps` — the maximum a rebalance's floor may undercut the oracle-implied output
Adapter	A fund-less contract wrapping one venue's swap, called only from inside `rebalance`
Zap	Untrusted periphery converting USDC into the basket and depositing in-kind, in one transaction
Seed	The one-shot bootstrap that sets the initial share price and locks `MIN_LOCKED` shares
Rebasing	A token whose `balanceOf` changes with no transfer — e.g. aUSDC accruing Aave yield
Halmos	A symbolic execution tool for EVM bytecode, used here for the two rounding proofs
Certora	A formal verification platform; a specification is an open pre-mainnet item

Source of the claims in this document

Every figure is taken from the repository or from a recorded on-chain read, not from estimate:

Claim	Source
140 tests / 19 suites; coverage figures	`forge test` and `forge coverage`, run 2026-09-21
Properties P1–P6, invariants, mutants	`test/`, `README.md` § Verification
Gas at n = 25	`test/IndexFund.gas.t.sol`
Addresses, decimals, feed ages	`VERIFICATION_S7.md` — Base block 51394468
DEX depth and leg counts	`script/probe-depth.py` — Base block 51400159
Live testnet status and smoke results	`deployments/base-sepolia-live-status.md`, `deployments/base-sepolia-smoke.json`
Composition blockers	`TASKS.md` § B, `VERIFICATION_S7.md`

Document version 1.0 · 21 September 2026 · Pre-audit. Not an offer. Not investment or legal advice.