SIGHASH LABS

Smart Contract Security Audit ParityUSD

November 2025

Table of Contents

1 Project Context

ParityUSD is a BCH-native, overcollateralized stablecoin system using CashScript contracts. Users mint against BCH collateral, pay/collect interest via a stability pool mechanism, redeem, and liquidate based on oracle pricing and protocol rules.

Project Name: ParityUSD

Website: https://parityusd.com

Network: Bitcoin Cash

Language: CashScript

GitHub: https://github.com/ParityUSD

Contracts Source Code: https://github.com/ParityUSD/contracts

First pass code snapshot:

- git commit e8308f5b43bfd4784a607f9b82911f2a96cc6176
- git tree hash 873f00ce35477e775cbd66ba499179af1a47ae4a
- cashc v0.11.3

Second pass code snapshot:

- git commit c3a07631772a238accb922128e23d3c97f309cec
- git tree hash 7446726ac529d1b2e261b16b778e79baf8bdc1f3
- cashc v0.12.0

Logo:

2 Executive Summary

Overall security posture: Strong after remediation. The contracts implement clear invariants, authenticate topology carefully, and now enforce correct accounting for interest, redemptions, and liquidations.

Severity recap:

- Critical (System integrity; redeploy required if found post-release): 1 finding — resolved.
- High (Security/Economic correctness): 1 finding — resolved.
- Medium (Robustness/Correctness): several findings — resolved.
- Low (Minor correctness/consistency): several findings — resolved.
- Informational/Style (Non-functional): several observations — addressed.

All issues identified in the preliminary report have been addressed and verified.

Notable improvements:

- Interest payout to Collector enforced with value-flow assertions.
- Debt cancellation standardized via OP_RETURN burn where applicable.
- Redemption mechanism reworked and hardened against price slippage.

3 Scope

- **In-scope:** all CashScript contracts (ending with *.cash*) under *contracts/*, including core *Parity.cash*, *PriceContract.cash*, *loan/**, *loanKey/**, *redeemer/**, *stabilitypool/**. In total 26 CashScript contract source files were audited.
- **Out of scope:** off-chain frontends, deployment tooling, automated transaction services, oracle services, etc.
- **Approach:** manual line-by-line review, transaction topology reasoning, value-flow accounting, token accounting, and robustness checks. The second pass reviewed the remediations.

4 Severity Definitions

- **Critical (System integrity; redeploy required):** Breaks protocol-level invariants or accounting in ways that would require a redeployment if discovered post-release. Not necessarily an immediate user-funds risk.
- **High (Security/Economic correctness):** Can cause economic loss or protocol misbehavior with limited off-chain mitigations.
- **Medium (Robustness/Correctness):** Edge-case correctness or resilience issues that may enable griefing, stalls, or degrade UX.
- **Low (Minor correctness/consistency):** Minor correctness/consistency impact; fix opportunistically.
- **Informational/Style (Non-functional):** Naming, duplication, indices hardcoding for simplicity, casing.

5 Status Definitions

- **Resolved:** Remediation implemented by the ParityUSD team and re-reviewed by Sighash Labs.
- **Acknowledged:** Known and accepted by the ParityUSD team.
- **Unresolved:** Not fixed (none).

6 Findings

1. **Critical** — Enforce interest routing to Collector
   - **Location:** *contracts/loan/loanContractFunctions/payInterest.cash*
   - **Issue:** Interest could be subtracted from collateral without enforcing routing to *Collector*,
     undermining system-level accounting for staking interest.
   - **Risk:** System integrity; mis-accounted interest flows.
     Would require redeploy if found post-release.
   - **Fix:** Enforced explicit output to Collector; added value-flow assertions; disallowed conflicting change outputs that siphon interest; validated residual collateral after fees.
   - **Status:** Resolved.

2. **High** — Add redemption slippage and oracle freshness protections
   - **Locations:** *contracts/loan/loanContractFunctions/redeem.cash*, *contracts /redeemer/Redemption.cash*
   - **Issue:** Getting the current oracle price only after the end of the redemption procedure could allow insufficient collateral return if price moved rapidly; no bounds/freshness implied fragility.
   - **Risk:** Economic loss via unfavorable redemptions; stalls due to collateral insufficiency.
   - **Fix:** Current oracle price is now already fetched at the start of the redemption procedure.
   - **Status:** Resolved.

3. **Medium** — Debt cancellation via OP_RETURN burns
    - **Locations:** *contracts/loan/loanContractFunctions/liquidate.cash*, *contracts/stabilitypool/poolContractFunctions/LiquidateLoan.cash*
    - **Issue**: Debt reconciliation previously depended on prior patterns; liquidation flows should cancel PUSD via explicit OP_RETURN burn to ensure supply and reserve consistency.
    - **Risk:** Token accounting ambiguity.
    - **Fix:** Added OP_RETURN burn outputs for canceled PUSD.
    - **Status:** Resolved.

4. **Medium** — Safe BCH change outputs with strict value-flow invariants
    - **Locations:** *contracts/redeemer/Redemption.cash* and related flows
    - **Issue:** Disallowing BCH change output required exact fee funding.
    - **Risk:** UX brittleness.
    - **Fix:** Allowed optional change output under strict invariants so that safety checks remain exact;
    - **Status**: Resolved.

5. **Medium** — Replace fixed fee constants with a bounded fee-budget pattern
    - **Locations**: stabilitypool/poolContractFunctions/NewPeriodPool.cash, stabilitypool/poolContractFunctions/LiquidateLoan.cash,
      and anywhere fixed 5000/1500 sat fees were used
    - **Issue:** Fixed fees are brittle under variable network fee rates.
    - **Risk:** Mis-estimated fees; overpaying for fees.
    - **Fix:** After careful consideration, the team has decided to keep the current approach to handling fees for contracts that pay their own fees, rather than requiring user funds. Changing this would introduce considerable complexity and was deemed not worthwhile given the low fee rates on the Bitcoin Cash network.
    - **Status:** Acknowledged.

6. **Low** — Redundant/tautological checks; duplicates
    - **Locations**:
      Redundant nftCommitment checks in *swapInRedemption.cash/swapOutRedemption.cash*; duplicated value checks in *loanKey/**; tautological comparisons in *Parity.cash*.
    - **Issue:** Redundant checks inflate size and reduce clarity without adding safety.
    - **Fix**: Removed redundant checks; consolidated duplicates; retained all safety-relevant assertions.
    - **Status:** Resolved.

7. **Informational/Style** — Naming, predicates, and index topology standardization
    - **Locations:** *LoanSidecar.cash*, *Loan.cash*, index hardcoding opportunities in *loanKey/**, minor predicate simplifications in redemption/loan flows; file casing consistency across repository.
    - **Issue:** Mixed naming styles, avoidable index indirection, and opportunities for smaller predicates.
    - **Fix:** Normalized casing; simplified predicates where safe and size-positive; hardcoded index relations where topology invariants are guaranteed and safety is unchanged.
    - **Status:** Resolved.

7 Cross-Cutting Improvements

Cross-cutting changes focus on making value-flow and accounting properties explicit and robust. Value-flow assertions are systematically applied to prevent leakage and enforce intended sinks such as the Collector. Collateral price slippage is now handled explicitly to avoid contract stalling, and token accounting is standardized around explicit OP_RETURN burns for debt cancellation.

8 Fix Review and Verification

All findings from the preliminary report have been addressed. The updated contracts enforce the intended invariants, with value-flow and token accounting checks made explicit and testable. Verification was performed by re-reviewing the code changes and reasoning through transaction topologies and invariants to ensure that the fixes close the identified issues without introducing new ones.

9 Deployment Transparency & Verifiability

Open-sourcing the contracts is necessary but not sufficient. Security also depends on the deployment transaction structure, bytecode and parameters that are deployed on-chain.

A flawless codebase can still be unsafe if deployment introduces drift. For example, deployed contracts might include incorrect contract parameters or the deployment transaction might include outputs that could introduce unwanted NFT’s into the system. To ensure users interact with the audited system, anyone must be able to independently verify that the on-chain deployment transactions are correct.

Status: The team is building an automated verification script that verifies deployed transactions against contract artifacts produced locally. Anyone will be able to run the script to reproduce the build and assert that the deployed contracts exactly match the audited sources and parameters.

10 Centralisation and Operational Considerations

ParityUSD’s price oracle is presently centralized: PriceContract.cash accepts a single oraclePublicKey and treats any message signed by that key as authoritative. This design is simple but it concentrates trust and creates a clear failure and manipulation point; compromise or misbehavior of the sole signer could impair minting, liquidations, and redemptions.

A practical path to decentralization is to replace the single signer with a committee that co-signs each price update.

Importantly, ParityUSD already supports upgrading the oracle contract and key material. Using the migrateContract contract function, the project can move to the aggregated threshold scheme without redeploying the broader protocol or breaking existing state.

11 Disclaimers

This audit reflects a best-effort, point-in-time review of the provided CashScript contracts and their intended behaviors. It assesses code quality, security invariants, and transaction topology against industry practices as of the report date. It does not constitute a warranty of fitness, correctness, or absence of vulnerabilities, nor does it guarantee that the contracts perform as intended under all conditions.

The number of possible states, inputs, and on-chain environments is effectively unbounded. As such, this report cannot certify the complete security of the system. Independent review and ongoing testing are strongly recommended, including a public bug bounty to further increase assurance.

Smart contracts depend on their underlying blockchain, virtual machine, compilers, libraries, and tooling, all of which may contain defects or undisclosed vulnerabilities.
External dependencies—such as oracles, relays, mempool conditions, and fee dynamics—can materially affect safety and liveness. These factors are outside the scope of this audit and may impact real-world behavior.

No responsibility is accepted for any loss of funds or damages arising from the use, deployment, or operation of the audited contracts. The auditor is not liable for remediation, incident response, or operational outcomes following this review.

12 Conclusion

ParityUSD's contracts, as remediated, present a robust implementation with clear accounting, sound value-flow controls, and improved resilience to variable fees and price movement. All identified issues have been resolved, and the system aligns with best practices for BCH CashScript-based protocols.

SIGHASH LABS

www.sighash.xyz

info@sighash.xyz