



Symbolic Software

SECURITY AUDIT REPORT

4Mica Payment Network Security Audit

4MCA-01

Prepared for ATM Labs (4Mica Network)

PREPARED BY

Dr. Nadim Kobeissi
Symbolic Software

ISSUE

October 15, 2026



Abstract

4Mica is a collateral-backed payment-guarantee network. Users deposit ETH or whitelisted stablecoins, and the stablecoins are supplied to Aave to earn yield. An off-chain operator issues BLS-signed guarantees that a payer's collateral covers a payment to a recipient. Obligations are periodically netted into settlement cycles, and the on-chain `ClearingHouse` collects debtor payments, seizes collateral from defaulters, and credits creditors. ATM Labs engaged Symbolic Software to audit the cryptography and software security of the 4mica-core monorepo: the Rust core-service and supporting crates, the Foundry/Solidity contracts, and the Rust SDK. The review examined the cryptographic and accounting correctness of the guarantee and settlement process across Rust and Solidity, including its behaviour under reorgs, concurrent operations, and attacks.

The BLS implementation matches the on-chain verifier byte-for-byte, settlement netting is provably zero-sum, and the reviewed collateral-conservation, reentrancy, and authentication protections hold. At the audited revision, the main risks concerned settlement and key management. A committed cycle had no terminal failure state, so a failed seizure could stall settlement and freeze creditors' funds. Under the production yield-fee setting, a fee-calculation underflow could cause such a failure. Operator and administrator authority was held by externally owned accounts with execution delays but no multisig, leaving user collateral exposed to a single compromised key. The review also found an encoding mismatch between Rust and Solidity in a deprecated on-chain verification path, and an incomplete V2 guarantee flow that is not enabled.

This report documents nine High, seven Medium, seven Low, and one Informational finding. The ratings account for the production configuration confirmed by the client, including the yield fee, V2 activation, governance roles, and chain finality settings. 4Mica has since fixed all 24 findings, and Symbolic Software reviewed each fix. Several fixes change the contracts, so they take effect only after a new deployment.

Engagement	4MCA-01, cryptography and software security audit
Prepared for	ATM Labs (4Mica Network)
Prepared by	Dr. Nadim Kobeissi, Symbolic Software
Issue	October 15, 2026
Target	4mica-core monorepo, commit 9cd9b77e (branch develop, June 18, 2026)
Scope	Rust core-service and supporting crates, Foundry/Solidity contracts, Rust SDK
Findings	24 findings: 9 High, 7 Medium, 7 Low, 1 Informational
Remediation	All 24 fixed in 15 pull requests merged between June 23 and October 15, 2026, and reviewed by Symbolic Software

Contents

1	Executive Summary	1
1.1	Overall assessment	1
1.2	Findings at a glance	2
1.3	Related findings	3
1.4	Remaining actions	4
2	Introduction	5
2.1	Engagement overview	5
2.2	Scope	5
2.3	Methodology	6
2.4	Severity classification	7
2.5	Terminology: tabs and settlement cycles	7
2.6	Limitations	8
3	Target Overview	9
3.1	System architecture	9
3.1.1	The settlement loop	9
3.2	Trust model	10
3.3	Security invariants	11
4	Privacy Audit	12
4.1	User data and access	12
4.2	Privacy finding	13
4.2.1	4MCA-M01 Any authenticated wallet can read any user's balance and exposure (IDOR)	13
4.3	Assessment	14
5	Security Audit	15
5.1	Verified properties	15
5.2	High findings	16
5.2.1	4MCA-H01 Operator key compromise enables theft of all user collateral	16
5.2.2	4MCA-H02 AccessManager admin can grant itself seizure authority and drain user funds	18
5.2.3	4MCA-H03 Aave illiquidity blocks stablecoin seizure and stalls settlement	19
5.2.4	4MCA-H04 ETH deposits can be credited twice after a reorg, allowing unbacked guarantees	20
5.2.5	4MCA-H05 Stablecoin fee reallocation underflows, reverting seizures and partial withdrawals	21
5.2.6	4MCA-H06 No terminal failure state: one unresolved debtor freezes all creditors in a cycle	22

5.2.7	4MCA-H07 ClearingHouse trusts the operator root without checking obligations or leaf sums	23
5.2.8	4MCA-H08 Deployment defaults concentrate governance in one key and allow delay bypasses	24
5.2.9	4MCA-H09 Yield-fee underflow freezes all creditors in a cycle until manual recovery	25
5.3	Medium findings	26
5.3.1	4MCA-M02 Withdrawal grace periods are not tied to settlement deadlines and can be shortened retroactively	26
5.3.2	4MCA-M03 Configuration permits non-finalized scanning despite incomplete reorg handling	27
5.3.3	4MCA-M04 An event that repeatedly fails processing can halt all chain indexing	28
5.3.4	4MCA-M05 A crash between storing and handling an event permanently skips its effects	29
5.3.5	4MCA-M06 Off-chain processing ignores SettlementSkipped events	30
5.3.6	4MCA-M07 Withdrawal requests ignore locked collateral and can race with guarantee issuance	31
5.4	Low findings	32
5.4.1	4MCA-L01 Incomplete V2 lifecycle lets payers withdraw collateral without paying recipients	32
5.4.2	4MCA-L02 V2 guarantees remain in PendingValidation because no code advances their status	33
5.4.3	4MCA-L03 Weak JWT signing secrets are accepted with only a warning	34
5.4.4	4MCA-L04 BLS key handling exposes command-line input and leaves input buffers uncleared	35
5.4.5	4MCA-L05 Merkle hashing does not distinguish leaves from internal nodes	36
5.4.6	4MCA-L06 EIP-712 request domain omits the verifying contract, enabling cross-deployment replay	37
5.4.7	4MCA-L07 Default collateral debits ignore the status update result, allowing duplicate debits after a reorg	38
5.5	Informational findings	39
5.5.1	4MCA-I01 Guarantee-claim ABI mismatch breaks deprecated on-chain verification	39
6	Conclusions	40
6.1	Remediation	40
6.2	Remaining actions	40
6.2.1	Until the fixes are deployed	40
6.2.2	Before mainnet	40
6.2.3	Further recommendations	42
6.3	Deployment configuration	42
6.4	Closing remarks	43
7	About Symbolic Software	44
	Bibliography	45
	Appendix A Cross-Language Test Vectors	47
A.1	Consistency matrix	47
A.2	Recommended test vectors	47

List of Figures

- 3.1 The 4Mica settlement loop. Green nodes are 4Mica components (the two on-chain contracts and the off-chain operator); the purple node is the untrusted user; the dashed node is the fully trusted external Aave deployment. Solid green arrows are value or control flow; dashed arrows are the event scanner mirroring chain state into the off-chain ledger. . . . 10

List of Tables

- 1.1 Summary of findings. 2
- 2.1 In-scope components of the 4mica-core monorepo. 6
- 2.2 Severity classification used in this report. 7
- 2.3 Mapping from the SOW’s “tab” vocabulary to the audited settlement-cycle model. 7
- 3.1 Actors and their trust levels. 10
- 3.2 Security invariant catalogue. Each finding references the invariants it affects. 11
- 6.1 Pull requests that fix the findings in this report. 41
- A.1 Rust ↔ Solidity consistency matrix at the audited revision. 48

List of Acronyms

ABI	Application Binary Interface
API	Application Programming Interface
BLS	Boneh–Lynn–Shacham (signature scheme)
CI	Continuous Integration
DST	Domain Separation Tag
ECDSA	Elliptic Curve Digital Signature Algorithm
EOA	Externally Owned Account
HSM	Hardware Security Module
HTTP	Hypertext Transfer Protocol
IDOR	Insecure Direct Object Reference
JWT	JSON Web Token
KAT	Known-Answer Test
L2	Layer 2
LoC	Lines of Code
ORM	Object-Relational Mapping
PoC	Proof of Concept
RPC	Remote Procedure Call
SDK	Software Development Kit
SIWE	Sign-In with Ethereum
SOW	Statement of Work
VK	Verification Key

Executive Summary

Symbolic Software audited **4Mica**, a collateral-backed payment-guarantee network. Users deposit collateral, and an off-chain operator issues Boneh–Lynn–Shacham (signature scheme) (BLS)-signed payment guarantees. Obligations are netted into settlement cycles and settled on-chain by a Clearing-House that collects debtor payments, seizes collateral from defaulters, and credits creditors. The review covered the Rust core-service and supporting crates, the Foundry/Solidity contracts, and the client Software Development Kit (SDK), at the commit listed in Section 2.6.

1.1 Overall assessment

The review found the core cryptographic and accounting mechanisms to be correct. The BLS implementation matches the on-chain verifier byte-for-byte, and settlement netting is provably zero-sum. Collateral is conserved, value-moving functions follow checks-effects-interactions, and reentrancy protection holds. Off-chain balance updates use a race-free optimistic-concurrency model, and the authentication checks are sound. Section 5.1 records these results and explains how they limit the impact of several findings.

At the audited revision, the main risk was *settlement liveness*: a committed cycle could end only in full success. If one debtor’s seizure failed, the cycle could stall and freeze all of its creditors’ funds until an authorised operator intervened. If the cause was Aave illiquidity, recovery could be delayed long enough for the loss to become permanent. The yield-fee underflow could also cause a freeze under the confirmed production configuration.

Governance and key concentration posed a second major risk. Operator and administrator authority was held by externally owned accounts, with execution delays but no multisig. A single compromised key could therefore move user collateral after the delay.

Two other areas had lower ratings under the confirmed configuration. The Rust and Solidity guarantee encodings disagreed in a deprecated on-chain verification path. The V2 validation flow was incomplete, but was disabled and unusable on the target chain.

4Mica has since fixed all 24 findings in pull requests merged between June 23 and October 15, 2026, and Symbolic Software reviewed each fix (Section 6.1). Several fixes change the contracts, so they take effect only after a new deployment. Until then, the measures in Section 1.4 still apply.

1.2 Findings at a glance

The report contains **nine High, seven Medium, seven Low, and one Informational** finding, listed in Table 1.1. Several ratings depend on production settings that are not recorded in the source repository: the yield fee, V2 activation, governance roles, and chain finality. The ratings reflect the configuration confirmed by the 4Mica team (Section 6.3); the table notes the relevant conditions. Other minor and defence-in-depth observations are not listed individually. All 24 findings are fixed; each finding's header links to its pull request.

Table 1.1. Summary of findings.

ID	Finding	Severity	Page
4MCA-H01	Operator key compromise enables theft of all user collateral.	High	16
4MCA-H02	AccessManager admin can grant itself seizure authority and drain user funds.	High	18
4MCA-H03	Aave illiquidity blocks stablecoin seizure and stalls settlement.	High	19
4MCA-H04	ETH deposits can be credited twice after a reorg, allowing unbacked guarantees (<i>non-finalized scanning; not in production</i>).	High	20
4MCA-H05	Stablecoin fee reallocation underflows, reverting seizures and partial withdrawals (<i>yield fee enabled</i>).	High	21
4MCA-H06	No terminal failure state: one unresolved debtor freezes all creditors in a cycle.	High	22
4MCA-H07	ClearingHouse trusts the operator root without checking obligations or leaf sums.	High	23
4MCA-H08	Deployment defaults concentrate governance in one key and allow delay bypasses.	High	24
4MCA-H09	Yield-fee underflow freezes all creditors in a cycle until manual recovery (<i>yield fee enabled</i>).	High	25
4MCA-M01	Any authenticated wallet can read any user's balance and exposure (IDOR).	Medium	13
4MCA-M02	Withdrawal grace periods are not tied to settlement deadlines and can be shortened retroactively.	Medium	26
4MCA-M03	Configuration permits non-finalized scanning despite incomplete reorg handling (<i>not used in production</i>).	Medium	27
4MCA-M04	An event that repeatedly fails processing can halt all chain indexing.	Medium	28
4MCA-M05	A crash between storing and handling an event permanently skips its effects.	Medium	29
4MCA-M06	Off-chain processing ignores SettlementSkipped events.	Medium	30
4MCA-M07	Withdrawal requests ignore locked collateral and can race with guarantee issuance.	Medium	31
4MCA-L01	Incomplete V2 lifecycle lets payers withdraw collateral without paying recipients (<i>V2 not enabled</i>).	Low	32

Table 1.1 continued

ID	Finding	Severity	Page
4MCA-L02	V2 guarantees remain in PendingValidation because no code advances their status (<i>V2 not enabled</i>).	Low	33
4MCA-L03	Weak JWT signing secrets are accepted with only a warning.	Low	34
4MCA-L04	BLS key handling exposes command-line input and leaves input buffers uncleared.	Low	35
4MCA-L05	Merkle hashing does not distinguish leaves from internal nodes.	Low	36
4MCA-L06	EIP-712 request domain omits the verifying contract, enabling cross-deployment replay.	Low	37
4MCA-L07	Default collateral debits ignore the status update result, allowing duplicate debits after a reorg.	Low	38
4MCA-I01	Guarantee-claim ABI mismatch breaks deprecated on-chain verification.	Info	39

1.3 Related findings

Several findings share a cause or combine to increase their impact. The descriptions below refer to the audited revision.

Settlement failures

A committed cycle has no abort, refund, or loss-sharing path (4MCA-H06). Three conditions can cause a seizure to fail and leave the cycle incomplete: Aave illiquidity (4MCA-H03), the yield-fee underflow (4MCA-H05, with the cycle-wide effect described in 4MCA-H09), and a debtor withdrawing before finality (4MCA-M02, 4MCA-M07). Each can freeze creditors' funds until an authorised operator intervenes. Recovery from Aave illiquidity also requires liquidity to return; a prolonged shortage can lead to permanent loss. The yield-fee trigger is possible in production.

Governance and key concentration

Roles held by externally owned accounts without a multisig (4MCA-H08) expose two theft paths: compromise of the operator key (4MCA-H01) or the AccessManager admin key (4MCA-H02). Governance authority also permits the grace-period changes described in 4MCA-M02. Execution delays postpone these actions but leave each key with the same authority.

Cross-language encoding and V2

The guarantee-claim encoding mismatch (4MCA-I01) affects the deprecated on-chain verification path. Separately, the incomplete V2 lifecycle (4MCA-L02) could let a payer keep collateral while the recipient remains unpaid (4MCA-L01). V2 is disabled and unusable on the target chain, so these V2 failures do not affect the current deployment.

Chain scanning

A configuration flag permits scanning non-finalized blocks with only a warning (4MCA-M03). If enabled, it could allow a reorg to produce unbacked guarantees (4MCA-H04). Production uses finalized-only scanning, which prevents both failures.

1.4 Remaining actions

The first three measures protect a deployment that still runs the audited contracts. The governance measures must be in place before a mainnet deployment holds real user funds. Section 6.2 gives the full list.

Priority	Required outcome	Findings
Until deployment	Disable the yield fee (<code>yieldFeeBps = 0</code>) until the fixed contracts are deployed, to prevent the failures the underflow causes in production.	4MCA-H05, 4MCA-H09
Until deployment	Keep the V2 flow disabled, to prevent loss of recipient funds. V2 is also unusable on the target chain.	4MCA-L01, 4MCA-L02, 4MCA-I01
Until deployment	Keep finalized-only chain scanning (<code>FINALIZED_HEAD_DEPTH = 0</code>), as deployed, to prevent the reorg failures.	4MCA-H04, 4MCA-M03
Before mainnet	Strengthen governance: make the AccessManager admin a timelocked multisig, give each role a separate holder, protect the operator key, and verify these settings against the live deployment.	4MCA-H01, 4MCA-H02, 4MCA-H08

Introduction

2.1 Engagement overview

ATM Labs engaged Symbolic Software to audit the cryptography and software security of **4Mica**, a collateral-backed payment-guarantee network. This report (engagement reference 4MCA-01; client reference 4MICA) documents the findings of that review.

4Mica lets users deposit collateral and obtain cryptographically enforced lines of credit for autonomous payments. The review covered:

- an off-chain operator service (`core-service`) that authenticates users, verifies payment requests, locks collateral, and issues BLS-signed payment guarantees;
- a set of on-chain contracts (`Core4Mica`, the collateral vault, and `ClearingHouse`, the settlement engine) that hold collateral, run periodic settlement cycles, seize collateral from defaulters, and credit creditors; and
- a Rust SDK that payers and recipients use to deposit, sign, issue, verify, and settle.

The audit examined the cryptographic and accounting correctness of deposits, issuance, netting, and settlement across Rust and Solidity. It also examined their behaviour under reorgs, concurrent operations, and attacks. Part 3 describes the architecture.

2.2 Scope

The audit covered the `4mica-core` monorepo at the commit pinned in Section 2.6. Table 2.1 lists the in-scope components.

The following were **out of scope**: the `metrics`, `derive`, and `telemetry-exporter` crates and the `core/src/metrics` module; test code; and read-only endpoints except where an endpoint raises an authorisation or privacy concern (for example, user balance reads, examined in Part 4). Third-party libraries (OpenZeppelin, Solady, Aave, `blst`, `jsonwebtoken`) were not themselves audited, but their *integration* and their version and trust assumptions were in scope.

The review covered the Statement of Work (SOW) work packages and the additional items in the client's `SCOPE.md`: settlement clearing and default liveness, Aave and stablecoin yield accounting, the clearing Merkle proofs, and the migration and schema invariants.

Table 2.1. In-scope components of the 4mica-core monorepo.

Component	Path (approx. LoC) and role
core-service	<code>core/</code> (~11.3k). HTTP API, authentication, guarantee issuance, settlement orchestration (cron), the Ethereum event scanner, and persistence.
crypto crate	<code>crates/crypto/</code> (~0.9k). BLS12-381 (blst), the keccak Merkle tree, and hex helpers.
rpc crate	<code>crates/rpc/</code> (~1.9k). Guarantee request/claim types, the ABI version-envelope codec, and the V2 validation-binding canonicalisation.
blockchain crate	<code>crates/blockchain/</code> (~0.25k). Ethereum transaction-submission helpers used by core-service to send and track settlement transactions.
migration, entities	<code>crates/migration/</code> <code>crates/entities/</code> The PostgreSQL schema and its invariants; ORM models and enumerations.
Contracts	<code>contracts/src/</code> (~1.76k). Core4Mica (vault), ClearingHouse (settlement), GuaranteeDecoderRouter, ValidationRegistryGuaranteeDecoder (V2/ERC-8004), BLSHelpers, the accounting library, and interfaces.
SDK	<code>sdk/</code> In scope only for the client side of SIWE and signing, where it informs the guarantee and authentication review.

2.3 Methodology

The review was primarily manual and guided by the threat model. For each required property (invariants INV-1–INV-12 in Section 3.3), we traced the code that enforces it in Rust and Solidity, on-chain and off-chain. The work comprised:

- **Manual source review.** We reviewed every in-scope file, on-chain and off-chain, following the data and value flows end-to-end (deposit → issuance → netting → commit → pay/claim/seize → finalize) and through the event scanner that mirrors chain state into the database.
- **Cross-language comparison.** We compared every constant, type string, Domain Separation Tag (DST), ABI struct, and hash preimage duplicated between Rust and Solidity field-by-field and, where feasible, byte-for-byte. These definitions are maintained separately in each language. Appendix A lists the results.
- **Targeted proofs-of-concept.** Where a defect could be demonstrated without the full on-chain stack, we built focused Proof of Concepts (PoCs). We reproduced the guarantee-claim ABI mismatch (4MCA-I01) by executing the real Rust encoder and decoding its output with the canonical alloy ABI codec. We confirmed the stablecoin fee-reallocation underflow (4MCA-H05) with an exact integer model of the on-chain arithmetic.
- **Dependency checks.** The contract submodules were absent from the audit copy (Section 2.6). We retrieved Solady’s BLS.sol [1] and OpenZeppelin’s MerkleProof [2] at the pinned submodule commits to compare their DST, point encoding, and Merkle hashing with the Rust implementation.

We also recommend adding dynamic and property-based tests: Foundry invariant/fuzz suites for the vault and ClearingHouse, proptest round-trips for the codecs, Anvil reorg simulation, and a cross-language Known-Answer Test (KAT) suite. These are detailed with the relevant findings and consolidated in Part 6 and Appendix A.

2.4 Severity classification

Findings are rated on a five-point scale (Table 2.2). Severity is assessed as the product of impact and likelihood, with particular attention to funds at risk, settlement liveness, and user privacy. Each finding records its affected invariants and any deployment conditions that affect its severity. This report details all findings of Medium severity and above, plus selected lower-severity findings. Other minor, defence-in-depth, and code-quality observations are not listed individually.

Table 2.2. Severity classification used in this report.

Severity	Typical criteria
Critical	Direct theft or loss of user collateral or treasury funds; minting of unbacked guarantees; acceptance of a forged signature; permanent global insolvency.
High	Conditional fund loss; double-use of collateral; a single key instantly breaking the solvency invariant; a stalled cycle freezing many creditors; an authorisation bypass that exposes or alters funds.
Medium	Bounded fund loss or griefing; a privacy leak such as cross-account balance disclosure; reorg-induced inconsistency requiring manual repair; denial of service of a subsystem.
Low	Hard-to-trigger or low-impact issues; defence-in-depth gaps; detectable misconfiguration.
Info	Code quality, dead or vestigial code, naming and version hygiene, and missing tests.

2.5 Terminology: tabs and settlement cycles

The SOW and the original scope document were written against an earlier *tab*-based remuneration model. The audited code has since migrated to a *settlement-cycle* / *ClearingHouse* model. Some old names remain in the code and, at the time of review, in the published SDK documentation. Table 2.3 maps those terms to the current implementation. This report uses the current vocabulary.

Table 2.3. Mapping from the SOW's "tab" vocabulary to the audited settlement-cycle model.

Legacy term	Current implementation
remunerate()	Removed. Default coverage now runs through <code>ClearingHouse</code> . <code>settleDefaultsFromCollateralBatch</code> and <code>fundCreditorsFromPoolBatch</code> , which call <code>Core4Mica.seizeCollateral</code> and <code>creditCollateral</code> .
remunerationGracePeriod	Split into the on-chain <code>withdrawalGracePeriod</code> and the off-chain <code>SettlementCycleConfig</code> windows.
tabExpirationTime	Per-cycle <code>payment_submission_deadline</code> / <code>payment_finality_deadline</code> .
synchronizationDelay	<code>SETTLEMENT_CLEARING_COMMIT_DELAY_SECS</code> plus the scanner's finality model.
Tabs / tab lifecycle	Settlement cycles and netted guarantees; the Tabs table was dropped by migration.
tabId	An inert field: still present in the on-chain <code>Guarantee</code> struct but unused for accounting; the Rust codec places <code>cycle_id</code> in the same ABI slot. This difference is examined in 4MCA-I01.

2.6 Limitations

- **Audited revision.** The review targeted commit `9cd9b77e1815a3db83031b773b2e4d92742ea912` (branch `develop`, dated 2026-06-18, “feat: auto-settle defaulted cycles”). All `file:line` references are against this revision and must be updated for later revisions.
- **Remediation review.** After the review, 4Mica fixed each finding in a pull request against `4mica-core`, merged between June 23 and October 15, 2026, and Symbolic Software reviewed each fix. The finding descriptions, locations, and ratings describe the audited revision. Section 6.1 lists the fixes.
- **Contract submodules absent; on-chain reproduction pending.** The Foundry submodules (`contracts/lib/{solady, openzeppelin-contracts, ...}`) were not checked out in the audit copy and `forge` was unavailable locally. The relevant library sources were obtained from the pinned submodule commits directly (Solady BLS.sol at `208e4f31`, OpenZeppelin at `b9c7078b`). The findings rest on manual review and Rust-side reproductions. Findings that would benefit from a Foundry or fork reproduction say so explicitly, and we recommend adding that test suite.
- **Production configuration.** The severity of several findings depends on deployment-time configuration that is set through the environment and not visible in the source: the governance roles and execution delays, whether the yield fee is enabled, whether the V2 validation flow is activated, the chain finality mode, the target chain(s), the Aave deployment, and the stablecoin whitelist. The 4Mica team has confirmed the values that affect severity, and the ratings and conditions in this report reflect them. Section 6.3 summarises the confirmed configuration.
- **Nature of the engagement.** A security audit is a best-effort, time-boxed review. The absence of reported findings in a given area is not proof of the absence of vulnerabilities there, and this report does not constitute a warranty, certification, or endorsement of the system’s fitness for any particular purpose.

Target Overview

3.1 System architecture

4Mica has three main components: an off-chain operator service, on-chain contracts, and a client SDK. This section describes how the components listed in Table 2.1 interact, transfer funds, and rely on one another.

Core4Mica, the collateral vault (Solidity)

Holds user collateral. ETH is held directly; whitelisted stablecoins are supplied to Aave and tracked as scaled aToken balances so that collateral earns yield. Users can deposit collateral, request a withdrawal, and finalize it after a grace period. The ClearingHouse uses `seizeCollateral` and `creditCollateral` to move collateral during settlement. The vault also stores the BLS Verification Key (VK) and exposes a public `verifyAndDecodeGuarantee` entry point.

ClearingHouse, the settlement engine (Solidity)

Runs settlement cycles. The operator commits a cycle with a Merkle root over the netted participant positions; debtors pay their net debit, creditors claim their net credit, defaulters' collateral is seized, and the cycle finalizes only when strict accounting equalities hold.

core-service, the operator (Rust)

Authenticates users over SIWE [3], verifies Elliptic Curve Digital Signature Algorithm (ECDSA)-signed payment requests, locks collateral in its database, issues BLS-signed guarantees, builds and nets settlement cycles, submits them on-chain, and runs an event scanner that mirrors chain state back into PostgreSQL. It holds the BLS signing key and the operator Externally Owned Account (EOA).

Supporting crates and SDK (Rust)

`crypto` (BLS12-381 via `blst` [4], keccak Merkle tree), `rpc` (guarantee types, the ABI version-envelope codec, V2 validation canonicalisation), and `migration/entities` (schema and ORM). The SDK drives the payer and recipient flows.

3.1.1 The settlement loop

Figure 3.1 shows the settlement process. A user deposits collateral into Core4Mica. The payer signs a payment request, which the operator verifies before locking collateral and returning a BLS-signed guarantee. The operator periodically nets outstanding obligations into a cycle and commits its Merkle root to the ClearingHouse. Debtors pay, creditors claim, and defaulters' collateral is seized to cover their obligations. The event scanner applies on-chain events to the operator's ledger.

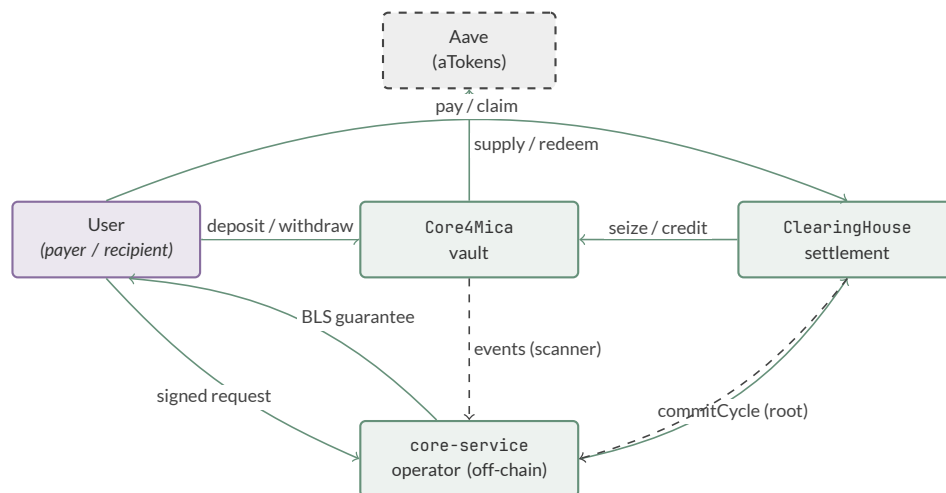


Figure 3.1. The 4Mica settlement loop. Green nodes are 4Mica components (the two on-chain contracts and the off-chain operator); the purple node is the untrusted user; the dashed node is the fully trusted external Aave deployment. Solid green arrows are value or control flow; dashed arrows are the event scanner mirroring chain state into the off-chain ledger.

3.2 Trust model

Table 3.1 lists the system’s trust assumptions. Two of them account for several findings. First, the ClearingHouse relies on the operator to submit correct netting results. It checks that totals are zero-sum and that each payout has a valid Merkle proof, but does not check whether the root reflects real obligations. Second, the off-chain ledger relies on a single Remote Procedure Call (RPC) provider for block numbers, finality, logs, and reorg detection.

Table 3.1. Actors and their trust levels.

Actor	Trust	Capabilities
On-chain caller / payer / recipient	Untrusted	Call any public contract function; deposit, withdraw, pay, claim, finalize; request guarantees over the API; present guarantees.
Listed stablecoin	Semi-trusted	Governance-whitelisted ERC-20, assumed 1:1, non-rebasing, and non-fee-on-transfer.
ERC-8004 validation registry [5]	Semi-trusted	Returns the validation status required by the V2 decoder; trusted only if in the decoder’s immutable allowlist.
RPC provider	Trusted (flagged)	Supplies blocks, logs, finality, and reorg data.
Aave [6]	Fully trusted	Holds stablecoin liquidity and supplies the liquidity index that drives all scaled-to-nominal arithmetic.
4Mica operator	Trusted	The core-service; holds the BLS signing key and operator EOA; issues guarantees; builds and submits cycles.
Governance (AccessManager)	Trusted; delayed	Sets the grace period, VK, versions, Aave config, and fees; pauses; operates the ClearingHouse.

3.3 Security invariants

Table 3.2 lists the properties the system must uphold. Each finding cites the invariants it affects.

Table 3.2. Security invariant catalogue. Each finding references the invariants it affects.

ID	Name	Statement
INV-1	Solvency / seizure window	No user can withdraw collateral that is still needed to back an unsettled guarantee; the withdrawal grace period must exceed the worst-case cycle time-to-finality plus a seizure margin.
INV-2	Conservation / zero-sum	Collateral and yield are neither created nor destroyed except via deposit, withdrawal, seizure, credit, and Aave yield; netting is zero-sum.
INV-3	No double-use of collateral	Locked plus pending-withdrawal collateral never exceeds the total, per (user, asset), under all interleavings.
INV-4	Guarantee uniqueness & binding	Each guarantee is unique and bound to its payer, recipient, amount, asset, and cycle; it cannot be replayed across cycles, versions, contracts, or chains.
INV-5	Settlement liveness	Every committed cycle reaches a terminal state; no participant can be permanently stranded.
INV-6	Cross-language soundness	Any artifact produced in Rust (signature, claim ABI, Merkle leaf/root, validation hash) verifies or decodes identically on-chain, and vice versa.
INV-7	Reorg convergence	After any reorg the off-chain ledger and on-chain state converge; no orphaned-block effect persists.
INV-8	Event exactly-once	Each on-chain event is applied exactly once to domain state, idempotent under retry, replay, or duplicate delivery.
INV-9	Auth integrity & least privilege	Only the authenticated subject can act on or read its own resources; privileged actions require the correct role.
INV-10	Signature soundness	Every signature check (BLS, ECDSA, ERC-1271 [7], JSON Web Token (JWT), SIWE) is non-malleable, non-replayable, domain- and chain-bound, and cannot be bypassed.
INV-11	Governance safety	Privileged actions are appropriately separated and time-delayed; no single key can instantly subvert solvency.
INV-12	Accounting precision	Scaled-to-nominal rounding, fee reallocation, and surplus handling never let value leak or balances underflow.

Privacy Audit

This part examines which user data is public and which should remain private. It covers balances, outstanding exposure, and payment relationships, followed by the cross-account balance disclosure in 4MCA-M01.

4.1 User data and access

4Mica stores user data on-chain and in the operator database.

On-chain (public)

`Core4Mica` and `ClearingHouse` emit events for deposits, withdrawal requests and finalizations, settlement-cycle lifecycle steps, seizures, and credits, naming the addresses and amounts involved. A user's ETH and stablecoin collateral level, and their participation in settlement, are therefore publicly observable as part of the settlement design.

Off-chain (operator database)

For each user, the `core-service` also stores the authenticated wallet address; per-asset balances split into `total` and `locked` (the collateral backing outstanding guarantees, which is not recorded directly on-chain); guarantee and payment history (request id, recipient, amount, asset, timestamps); withdrawal requests; settlement-cycle membership; and authentication state (SIWE nonces, refresh tokens). The `locked` exposure and the off-chain linkage of these records to authenticated sessions are private; they cannot be readily derived from on-chain data.

Only the user or an explicitly authorised operator role should be able to read that user's off-chain balance and exposure. The reviewed authentication checks include chain-id binding, SIWE address binding, single-use nonces, and constant-time scope checks (Section 5.1). The balance endpoint, however, omits the authorisation check described below.

4.2 Privacy finding

4MCA-M01 Any authenticated wallet can read any user's balance and exposure (IDOR)

Medium

INV-9

✓ Fixed #132

This Insecure Direct Object Reference (IDOR) exposes any user's per-asset collateral, outstanding-guarantee exposure, and account existence to any authenticated wallet. The finding is rated Medium because it discloses data without allowing funds to be moved.

Location: `core/src/service/query.rs` — `get_user_asset_balance` (lines 28–44).

Location: `core/src/http.rs` — the route (lines 84–87).

Location: `core/src/auth/access.rs` — the unused helper `require_user_match` (lines 69–76).

Location: `core/src/auth/constants.rs` — the default scope (line 23).

Location: `core/src/service/auth.rs` — the default scope (lines 117–127).

Description. The balance-read handler checks only `require_scope(auth, SCOPE_PAYMENT_READ)`. It then passes the caller-supplied `user_address` to the repository without checking that it matches `auth.wallet_address`. The response exposes the target user's total collateral and locked amount; a `Some` or `None` result also reveals whether the account exists.

Other user-scoped reads check the caller's identity: `list_recipient_payments` calls `require_recipient_match`, and the clearing-proof and action handlers require `addresses_match || admin || facilitator`. The corresponding balance helper, `require_user_match(auth, &user_address)`, already exists but is not called here. This suggests an oversight.

Reachability. Any caller who completes a SIWE login can reach this endpoint: `load_wallet_claims` returns `DEFAULT_SCOPES = ["payment:read"]` for any wallet without a provisioned role row, so every authenticated wallet holds the required scope.

Reproduction. With the development stack running, log in as wallet A (SIWE → JWT), then issue `GET /core/users/{B}/assets/{asset}` with A's bearer token for an unrelated address B; the response returns B's balance.

Recommendation

Add `access::require_user_match(auth, &user_address)` to `get_user_asset_balance`, with an `admin / facilitator` cross-account override if such reads are intended for those roles (mirroring the clearing-proof handlers). Confirm whether `payment:read` is meant to be a subject-scoped read (a wallet reads only its own balance) or a global read scope, since it is currently granted to every authenticated wallet by default (Section 6.3).

4.3 Assessment

The other reviewed off-chain reads require authentication and check that the caller may access the requested account. Adding the existing subject-binding check to the balance handler would restrict access to the account owner or an explicitly authorised role. Users should also be told that deposits, withdrawals, and settlement activity are publicly visible on-chain.

Security Audit

This part presents the security findings in descending order of severity. The header of each finding lists the invariants it affects (Table 3.2) and its remediation status, which links to the corresponding pull request. The panel below the header summarises the impact. The source locations, the description, and any deployment conditions that affect the rating come next. Section 6.3 records the configuration confirmed by the 4Mica team. The privacy finding is presented separately in Part 4.

5.1 Verified properties

The review verified the following properties. Several of these checks limit the impact of the findings that follow.

- **BLS signing and verification agree byte-for-byte.** The DST matches Solady's hard-coded tag. The min-pk basic scheme [8] (hashing to G2 as specified in RFC 9380 [9]), empty augmentation, and pairing identity are correct. G1/G2 point encoding and Fp2 component ordering match EIP-2537 [10] and Solady, as confirmed by a known-answer check against the standard G1 generator embedded on-chain as `negatedG1Generator`. Subgroup and infinity checks cover all external paths that construct keys or signatures.
- **Merkle hashing matches OpenZeppelin.** Sorted-pair commutative-keccak hashing, empty and single-leaf trees, and odd-node self-pairing produce roots consistent with the on-chain verifier. The 224-byte clearing leaf preimage is safe against second-preimage attacks in its current use.
- **Cross-language encodings agree, except for the claim struct.** The `(uint64, bytes)` version envelope is compatible across Rust and Solidity. V2 validation subject and request hash preimages match field-for-field. The guarantee domain separator binds the chain id and contract address, and EIP-712 [11] domain name casing is consistent.
- **Collateral conservation and reentrancy protection hold.** Every function that moves value follows checks-effects-interactions and is `nonReentrant`. ETH and stablecoin balances are conserved, and the scaled-balance reconciliation identity is consistent. The yield-fee inversion helper terminates without over-extracting. The `aToken` validation and withdrawal clamping are correct.
- **Settlement netting is zero-sum and protected against overflow.** Net debits and credits are complementary, and all aggregation uses checked addition. Merkle leaves bind the chain id and `ClearingHouse` address, preventing replay across chains or contracts. Payment, claim, and default guards prevent a position from being resolved twice.
- **Off-chain balance updates are race-free.** Every balance mutation uses a version compare-and-swap with retry. Guarantee uniqueness is enforced, and locked collateral cannot be released twice. The withdrawal-request race in 4MCA-M07 arises because withdrawal requests do not update the balance version.

- **The default scanner configuration uses finalized blocks.** Validation requires finalized mode. Without the depth override described in 4MCA-M03, only finalized events are applied and reorg handling is not needed. Most event handlers are idempotent under retry.
- **The core authentication checks are correct.** The JWT algorithm is fixed, preventing algorithm confusion, and chain id is checked throughout the flow. The SIWE address is checked at three points, and the signature covers the raw message. Nonces are single-use through an atomic compare-and-swap. Refresh tokens rotate after one use, scope checks are constant-time, and secrets are zeroized.
- **The on-chain role design separates privileges.** Five roles cover all privileged functions. The function-to-role mapping is complete; an unmapped function defaults to admin-only access. The role that moves collateral is assigned to the `ClearingHouse`. Findings 4MCA-H02 and 4MCA-H08 concern how these roles are assigned and administered at deployment.

5.2 High findings

4MCA-H01 Operator key compromise enables theft of all user collateral

High

INV-2, INV-11

✓ Fixed #168

An attacker with the operator key can steal all user collateral. The contract does not bind seizures to real obligations or check leaf sums, and it imposes no obligation-based per-user cap, multisig, or timelock (4MCA-H07).

Location: `contracts/src/ClearingHouse.sol` — `commitCycle` (lines 122–164) and `settleDefaultsFromCollateralBatch` (216–221).

Location: `contracts/src/Core4Mica.sol` — `seizeCollateral` (lines 457–477).

Location: `contracts/script/Core4MicaFullStack.s.sol` — deployment configuration.

Description. `commitCycle` checks that the cycle id is unused; that the id, root, and totals are nonzero; that `totalNetDebit == totalNetCredit`; and that `paymentFinalityDeadline >= paymentSubmissionDeadline`. It does not check that the deadlines are in the future, that the root corresponds to signed obligations, or that leaf amounts sum to the committed totals. Default settlement requires only `block.timestamp > paymentFinalityDeadline`, with the deadline chosen by the operator. `seizeCollateral` checks the debtor's withdrawable balance but not whether the debtor owes the amount claimed.

A compromised `FOURMICA_OPERATOR` key can therefore settle a fabricated cycle for any victim *V* and attacker *A*. Without execution delays on the settlement selectors, the following steps can run within one block:

1. `commitCycle` a fabricated cycle whose root is the tree of leaves $(v, v_bal, NetDebtor)$ and $(A, v_bal, NetCreditor)$, with both deadlines set to 1 (in the past). This passes every check.
2. `settleDefaultsFromCollateralBatch([V]): block.timestamp > 1` holds, so it seizes *V*'s collateral into the pool.
3. `fundCreditorsFromPoolBatch([A]):` credits *A*'s collateral with *V*'s seized funds; *A* withdraws after the grace period.

Repeated per user and asset, this drains *all* user collateral. *V* never interacted with the cycle and never signed anything.

Conditions. Requires compromise of the operator key. Production adds an execution delay to the externally owned operator account. The delay prevents settlement within a single block and makes a malicious settlement observable before execution, but leaves the operator's authority and the validation gap unchanged (Section 6.3).

Recommendation

Bind the committed root to verifiable issued obligations. At minimum, accumulate seized and credited leaf amounts and check them against `totalNetDebit` and `totalNetCredit` at `finalizeCycle`. Require a future `paymentFinalityDeadline` to prevent settlement in the commit block, and consider per-user seizure caps tied to provable obligations.

Before mainnet, assign `FOURMICA_OPERATOR` to a multisig with a non-zero execution delay on settlement selectors. Protect signing keys with an Hardware Security Module (HSM), separate the role from the deployer EOA, and configure admin and grant delays to prevent immediate grants of a zero-delay operator role. See also 4MCA-H02, 4MCA-H07, and 4MCA-H08.

4MCA-H02 AccessManager admin can grant itself seizure authority and drain user funds

High

INV-11, INV-2

✓ Fixed #136

Compromise of the AccessManager admin key permits theft of all user funds and control of the protocol. This key provides a simpler theft path than the operator key in 4MCA-H01 and requires separate protection.

Location: `contracts/src/Core4Mica.sol` — `seizeCollateral` (lines 457–477).

Location: `contracts/script/Core4MicaFullStack.s.sol` — deployment configuration: `managerAdmin` defaults to the deployer (line 121), `CLEARING_HOUSE_ROLE` wiring (243–251), and no `setGrantDelay`, `setTargetAdminDelay`, or `setRoleAdmin` call anywhere.

Location: `AccessManager` — `OpenZeppelin` library contract.

Description. The holder of `ADMIN_ROLE` (id 0) in `OpenZeppelin AccessManager` [12] administers every role by default. Unless configured otherwise, the deployment script assigns this role to the deployer EOA with execution delay 0 and sets no grant or target-admin delay.

`seizeCollateral` requires `CLEARING_HOUSE_ROLE` and sends seized funds to `msg.sender`. It checks the role and the debtor's withdrawable balance, but no cycle, root, or proof. Without a grant delay, the admin can call `grantRole(CLEARING_HOUSE_ROLE, attacker, 0)`, after which the attacker can immediately call `seizeCollateral(victim, asset, victim_balance)`. Repeating this for each user and asset drains all collateral. The admin can also use `setTargetFunctionRole` to assign the function to a role it already holds.

The same authority allows the admin to grant itself `GOVERNANCE`, which controls the grace period, BLS verification key, and Aave configuration, or `TREASURY`, which controls yield and surplus.

Conditions. Requires compromise of the admin key. Production uses an externally owned admin account with an execution delay, making malicious role changes observable before execution. A multisig admin with a timelock and configured grant and admin delays is needed to address the remaining key-concentration risk (Section 6.3).

Recommendation

Assign `ACCESS_MANAGER_ADMIN` to a multisig with a timelock. Configure non-zero admin execution and grant delays, including `setTargetAdminDelay` and `setGrantDelay`, so role changes are publicly visible before taking effect. Renounce the deployer's admin role after setup. Have the deployment script verify that the production admin is separate from the deployer and that all delays meet a non-zero minimum. These measures also address 4MCA-H01 and 4MCA-H08.

4MCA-H03 Aave illiquidity blocks stablecoin seizure and stalls settlement

High

INV-5, INV-2

✓ Fixed #133

Aave illiquidity can freeze every creditor's funds in an affected cycle and can stall all active stablecoin cycles at once. The operator cannot force Aave to supply liquidity, and a shortage that outlasts the withdrawal grace period can lead to permanent loss. No key compromise, reorg, or enabled yield fee is required.

Location: `contracts/src/Core4Mica.sol` — `seizeCollateral` (lines 457–477), the `_debitUserStablecoin` shortfall revert (866–872), and `_withdrawStablecoinAndMeasureScaledBurn` (906–915).

Location: `contracts/src/ClearingHouse.sol` — the `try/catch skip` (lines 246–254), the `funded gate` (192, 269), and the `finalize equalities` (323–331).

Location: `core/src/service/clearing.rs`, lines 155–264.

Description. Stablecoin seizure redeems the debtor's `aToken` position through `_aavePool().withdraw(asset, amount, ClearingHouse)`. If Aave cannot supply the full amount because the reserve is illiquid, paused, or frozen, a short withdrawal causes `_debitUserStablecoin` to revert with `StablecoinWithdrawShortfall`. There is no fallback that seizes the `aToken` or scaled position directly.

`settleDefaultsFromCollateralBatch` catches the failed seizure and emits `SettlementSkipped`, leaving the debtor unresolved. As a result, `totalResolvedDebit < totalNetDebit`, which prevents finalization, and `funded < totalNetCredit`, which prevents every creditor from claiming. The off-chain pipeline submits settlement once and does not retry skipped seizures. There is also no on-chain abort or refund state.

The operator can retry the unresolved debtor, but seizure cannot succeed until Aave liquidity returns. A shortage can coincide with the market stress that causes defaults. If it lasts beyond the withdrawal grace period, the debtor can exit before seizure and the creditor loss can become permanent.

Conditions. Affects stablecoin cycles only. It is possible under the default configuration; its likelihood depends on the target Aave reserves (Section 6.3).

Recommendation

Allow seizure without an immediate Aave withdrawal. Transfer the debtor's `aToken` to the pool or reassign the scaled balance, then redeem the underlying asset when creditors claim, with retries if liquidity is unavailable. Also add the terminal failure state and loss policy in 4MCA-H06, and make the off-chain service retry skipped seizures.

4MCA-H04 ETH deposits can be credited twice after a reorg, allowing unbacked guarantees**High**

INV-3, INV-2

✓ Fixed #134

Unbacked guarantees can cause permanent loss to creditors. A user need not cause the reorg: a naturally occurring reorg can duplicate an ETH deposit credit, and the resulting inflated balance does not self-correct.

Location: `core/src/service/event_handler.rs` — `handle_collateral_deposited` (lines 18–40) and the `sync_stablecoin_balance_from_chain` early return for ETH (240–242).

Location: `core/src/ethereum/scanner.rs` — deduplication keyed by block hash (lines 328–348), `maybe_handle_reorg`, which deletes only the cursor (484–506), and the scanner stall (108–112).

Location: `m20260507_000007` — migration with the `uniq_collateral_event_identity` index.

Location: `core/src/config.rs` — `FINALIZED_HEAD_DEPTH` (lines 80–83, 116–128).

Description. ETH collateral balances are derived from events without reconciliation against the chain. Stablecoin deposit and withdrawal handlers overwrite the off-chain total with the on-chain guaranteeCapacity, but `sync_stablecoin_balance_from_chain` returns early for `asset == address(0)` (ETH). There is no equivalent ETH reconciliation.

Both deduplication layers include `block_hash` in the event identity. If a processed ETH `CollateralDeposited` transaction is re-mined into another block, the new block hash bypasses both checks. `maybe_handle_reorg` deletes only the scan cursor, leaving the orphaned credit in place. The deposit handler then credits the deposit again.

The off-chain ETH balance can remain roughly twice the real collateral. Issuance uses only the off-chain check `free = total - locked - pending`, allowing guarantees worth roughly twice the available collateral. Settlement cannot seize the missing funds, so the cycle stalls (4MCA-H06) and creditors remain unpaid.

Conditions. Requires the scanner to process blocks that can still reorganise, for example with `FINALIZED_HEAD_DEPTH > 0`, which validation permits with a warning (4MCA-M03). Production uses finalized-only scanning with `FINALIZED_HEAD_DEPTH = 0`, so this finding cannot occur under the confirmed configuration (Section 6.3). The High rating reflects the loss possible if the override is enabled. Operators may be tempted to enable it when a Layer 2 (L2) RPC stops advancing the finalized tag: the scanner stalls, and a code comment in the scanner suggests this setting as a workaround.

Recommendation

Reject `FINALIZED_HEAD_DEPTH > 0` unless the operator explicitly acknowledges the reorg risk (4MCA-M03). If a depth heuristic is required on an L2, use a reorg-stable identity for event and collateral deduplication (`tx_hash + log_index`) and roll back application state after a reorg. Add ETH balance reconciliation equivalent to the stablecoin path: read on-chain collateral after each deposit and withdrawal, then overwrite the off-chain total.

4MCA-H05 Stablecoin fee reallocation underflows, reverting seizures and partial withdrawals

High

INV-12, INV-1, INV-5

✓ Fixed #133

With `yieldFeeBps > 0` (up to 50%), the fee-reallocation underflow affects seizures and partial withdrawals. A seizure reverts when the debtor's yield is at a fee-step boundary and the net debit does not exceed principal. The `ClearingHouse` skips the debtor, leaving creditors unpaid and the cycle stalled (4MCA-H06, 4MCA-H09). The estimated per-seizure failure probability is roughly the fee rate. A user's partial withdrawal can also revert; the user can work around this by withdrawing the full balance or submitting a new request, though a new request restarts the grace period.

Location: `contracts/src/Core4Mica.sol` — `_debitUserStablecoin` (lines 853–886), specifically line 865, `protocolFeeReallocatedUnderlying = gross - userYieldWithdrawn - remainingGross`; reached from `seizeCollateral` (475) and `_finalizeStablecoinWithdrawal` (844).

Location: `contracts/src/Libraries/Core4MicaAccounting.sol` — `grossForNetYield` (lines 33–45).

Description. A stablecoin debit reallocates the protocol fee on realised yield. It computes `remainingGross = grossForNetYield(remainingUserNet)`, then computes the reallocated fee as `gross - userYieldWithdrawn - remainingGross`.

`grossForNetYield` returns the largest gross amount whose net equals its argument. Fee rounding creates boundaries where `netYield(g) == netYield(g+1)`, so the helper returns `g+1`. When a debit is no greater than the user's principal, `userYieldWithdrawn == 0`. At these boundaries, `remainingGross > gross`, and the subtraction reverts with `Solidity panic 0x11`.

Empirical confirmation. An exact integer model found that the fraction of yield states that revert equals the fee rate: $1\% \rightarrow 1.00\%$, $5\% \rightarrow 5.00\%$, $25\% \rightarrow 25.00\%$, and $50\% \rightarrow 50.00\%$. This held at realistic accrued-yield scales of approximately 1 DAI or 500 USDC. No underflow occurs when `yieldFeeBps == 0`. The model also confirmed that `grossForNetYield` terminates in at most one iteration and does not overcharge.

Conditions. Requires `yieldFeeBps > 0`, as confirmed in production (Section 6.3). Setting `yieldFeeBps = 0` prevents the underflow until a fix is deployed.

Recommendation

Use saturating subtraction for fee reallocation: `protocolFee = (gross - userYieldWithdrawn) > remainingGross ? (gross - userYieldWithdrawn - remainingGross) : 0`. This can undercharge by at most one unit, in the user's favour. Alternatives are to compute the fee directly on realised yield or make `grossForNetYield` return the smallest preimage, as its documentation states. Add a Foundry fuzz test over (`principal`, `gross`, `feeBps`, `withdrawalAmount`) to check that `_debitUserStablecoin` never reverts for `amount <= withdrawable`. Set `yieldFeeBps = 0` until the fix is deployed.

4MCA-H06 No terminal failure state: one unresolved debtor freezes all creditors in a cycle**High**

INV-5

✓ Fixed #133

One unresolved debtor can lock all funds paid into the cycle and prevent every creditor from receiving payment. Seizure can fail because of the fee underflow (4MCA-H05), an early withdrawal (4MCA-M02), or Aave illiquidity (4MCA-H03). These conditions break the contract's assumption that collateral locked at issuance remains available for each debtor's net debit.

Location: `contracts/src/ClearingHouse.sol` — `skip-on-seize-failure` and the in-code `TODO` (lines 241–256), the funded gate (192, 269), the `finalize` equalities (317–331), and the absence of any non-`Finalized` terminal state (25–30).

Location: `core/src/service/clearing.rs` — `open_cycle_settlement` (lines 155–264, marking `Settling` at 257) and `finalize_settling_cycle` (268–300).

Description. A committed cycle can end only in `Finalized`. There is no cancellation, abort, or `Shortfall` state, and no refund path for funds already paid in. When `seizeCollateral` reverts, `settleDefaultsFromCollateralBatch` catches the failure and emits `SettlementSkipped`. The debtor remains neither paid nor defaulted.

As long as that debtor is unresolved, `totalResolvedDebit < totalNetDebit` prevents finalization. Every creditor payout also requires `funded >= totalNetCredit`, so one unpaid debtor freezes all creditors even when other debtors have paid. A `TODO` in the contract acknowledges this problem.

The off-chain pipeline provides no automatic recovery. `open_cycle_settlement` submits the batches once and marks the cycle `Settling` regardless of skipped seizures. Subsequent calls to `finalize_settling_cycle` return `Ok(false)` without retrying them.

Conditions. Rated High because it allows a single failed seizure to block an entire cycle. The Aave trigger in 4MCA-H03 requires external liquidity to return before recovery; the yield-fee trigger in 4MCA-H09 can be resolved by an operator retry once the fee is set to zero or accrued yield moves past the fee-step boundary.

Recommendation

Add a terminal failure path: a `Shortfall` / `Cancelled` state with an explicit loss policy (sharing losses pro rata or partially crediting creditors) and a refund route for funds already paid into a cycle that cannot finalize, so a cycle can always terminate. Make the off-chain pipeline decode `SettlementSkipped` (4MCA-M06) and retry skipped seizures rather than marking `Settling` once and abandoning them. Fix the underlying triggers (4MCA-H05, 4MCA-M02, 4MCA-H03).

4MCA-H07 ClearingHouse trusts the operator root without checking obligations or leaf sums

High

INV-2, INV-3, INV-11

✓ Fixed #135

The contract relies on off-chain netting without checking obligations or leaf sums and does not itself impose an execution delay. Operator compromise permits theft across the system (4MCA-H01); operator bugs can stall cycles or cause excess seizures.

Location: `contracts/src/ClearingHouse.sol` — `commitCycle` (lines 122–164) and the operator batches (211–256, 261–315).

Description. `commitCycle` checks that the cycle id is unused; that the id, root, and totals are nonzero; that `totalNetDebit == totalNetCredit`; and that deadlines are ordered. It does not verify signed obligations or check whether debtor and creditor leaf sums match the committed totals. A participant needs only a Merkle proof against the operator's root.

If an operator bug produces inconsistent totals, the finalization equalities may be impossible to satisfy, stalling the cycle or stranding seized collateral (4MCA-H06). A compromised operator can name arbitrary users as debtors and route their collateral to attacker-controlled creditors. Guarantee signatures are not checked during settlement; 4MCA-H01 describes the resulting theft path.

Recommendation

At minimum, accumulate the seized and credited leaf amounts during settlement and assert they equal the committed `totalNetDebit / totalNetCredit` at `finalizeCycle`, to reject inconsistent totals. Longer term, bind the committed root to verifiable issued obligations on-chain. Protect the operator key with an HSM or multisig and add an execution delay to the settlement selectors (4MCA-H01, 4MCA-H08), and document the trust model for users.

4MCA-H08 Deployment defaults concentrate governance in one key and allow delay bypasses

High

INV-11, INV-1

✓ Fixed #136

These deployment defaults enable the grace-period changes in 4MCA-M02 and the operator and admin theft paths in 4MCA-H01, 4MCA-H02, and 4MCA-H07. One key can lower the grace period, replace the BLS VK, reconfigure Aave, change the yield fee, drain the treasury, and redirect user collateral through settlement.

Location: `contracts/script/Core4MicaFullStack.s.sol` — default delays (lines 57–60), `managerAdmin = envOr("ACCESS_MANAGER_ADMIN", deployer) (121)`, role grants with `env-or-deployer` holders and `env-or-default` delays (211–216, 251), the delay accessors (376–393), and the absence of any `setGrantDelay`, `setRoleAdmin`, or `setTargetAdminDelay` call anywhere in the script.

Description. The role design separates privileges (Section 5.1), but deployment defaults can assign every role to one key and permit the admin to bypass role delays:

1. **One key holds every role.** If `*_ROLE_HOLDER` and `ACCESS_MANAGER_ADMIN` are unset, the deployer EOA becomes the admin and holds `GOVERNANCE`, `TREASURY`, `GUARDIAN`, and `FOURMICA_OPERATOR`.
2. **Delays can be zero.** `GUARDIAN` and `FOURMICA_OPERATOR` default to zero delay. `GOVERNANCE` and `TREASURY` default to 72 hours, but environment settings can reduce them to zero without validation.
3. **The admin can bypass role delays.** The script configures no grant, role-admin, or target-admin delay, and `ADMIN_ROLE` starts with zero execution delay. Unless a delay is added, the admin can immediately call `grantRole(GOVERNANCE_ROLE, freshKey, 0)` or assign a function to a role it holds. The 72-hour delay therefore constrains only non-admin role holders.

Conditions. Production uses externally owned accounts with execution delays (Section 6.3). This prevents immediate execution, but each key retains its authority. Multisigs with timelocks and separate role holders are needed to address key concentration. Findings 4MCA-H01 and 4MCA-H02 describe the consequences of compromise.

Recommendation

Use a multisig with a timelock for `ACCESS_MANAGER_ADMIN`. Assign `GOVERNANCE`, `TREASURY`, `GUARDIAN`, and `FOURMICA_OPERATOR` to separate holders, distinct from the deployer. Configure non-zero admin execution and grant delays, and enforce minimum non-zero delays for `GOVERNANCE` and `TREASURY`. Renounce the deployer's temporary roles after setup. Verify these settings in the deployment script and against the live deployment.

4MCA-H09 Yield-fee underflow freezes all creditors in a cycle until manual recovery

High

INV-5, INV-2, INV-12

✓ Fixed #133

An unprivileged debtor can freeze an entire cycle under the production yield-fee setting. In the proof of concept, a skipped seizure from one debtor owing 400 USDC blocks all 1000 USDC owed to creditors. Recovery requires manual intervention because the pipeline neither decodes `SettlementSkipped` (4MCA-M06) nor retries. If the operator does not recover the cycle before the debtor's grace period ends and the debtor withdraws, the freeze can become a loss.

Location: `contracts/src/Core4Mica.sol` — trigger: the `_debitUserStablecoin` fee reallocation at line 865 (see 4MCA-H05).

Location: `contracts/src/ClearingHouse.sol` — cycle failure (lines 246–254, 192, 269, 323–331).

Location: `core/src/service/clearing.rs` — off-chain pipeline (lines 155–264).

Description. This finding combines the fee underflow in 4MCA-H05 with the missing failure state in 4MCA-H06. When `yieldFeeBps > 0`, `seizeCollateral(debtor, stablecoin, netDebit)` can underflow at a fee-step boundary. An exact integer model confirmed that these boundaries occupy a $\text{yieldFeeBps}/10000$ fraction of yield states: 5.00% at a 5% fee. The failure can occur during ordinary defaults, and a malicious debtor can increase its likelihood without special privileges.

The contract catches the failed seizure and leaves the debtor unresolved. `totalResolvedDebit < totalNetDebit` prevents finalization, while the full-funding requirement blocks every creditor's claim. The off-chain pipeline never resubmits the seizure, so all creditors remain unpaid until an authorised operator intervenes.

Recovery. The operator can retry `settleDefaultsFromCollateralBatch` while the cycle is `Defaulted` (`ClearingHouse.sol:217`). The catch path does not mark the skipped debtor as defaulted, so the debtor remains eligible for seizure. A retry can succeed after governance sets `yieldFeeBps = 0`, or at a later block when accrued yield has moved past the fee-step boundary. Once the seizure succeeds, the cycle can pass the funding and finalization checks.

The default `withdrawalGracePeriod` is 22 days, compared with approximately 1.7 days to cycle finality. This gives the operator time to retry before the debtor can finalize a withdrawal of the collateral.

Conditions. Affects stablecoin cycles with `yieldFeeBps > 0`, as confirmed in production (Section 6.3). ETH cycles are unaffected. Setting `yieldFeeBps = 0` prevents the underflow; already skipped seizures still need to be retried.

Recommendation

Fix the underflow using direct fee calculation or saturating reallocation (4MCA-H05), so seizures do not underflow for `netDebit <= withdrawable`. Add a terminal failure state and loss policy (4MCA-H06). Decode `SettlementSkipped`, alert on failures, and automatically retry skipped seizures (4MCA-M06). Keep `yieldFeeBps = 0` until these changes are deployed.

5.3 Medium findings

4MCA-M02 Withdrawal grace periods are not tied to settlement deadlines and can be shortened retroactively

Medium

INV-1

✓ Fixed #133

A governance change or an incorrect off-chain settlement window can let a debtor withdraw collateral before seizure, leaving guarantees unbacked and creditors exposed to loss. The finding is rated Medium because it depends on privileged action or configuration, and because production delays governance execution. Finding 4MCA-M07 covers the corresponding off-chain reservation gap.

Location: `contracts/src/Core4Mica.sol` — the `finalizeWithdrawal` grace check (lines 437–442), `setWithdrawalGracePeriod` (212–215), and `WithdrawalRequest`, which stores only `{timestamp, amount}` (115–118).

Location: `contracts/src/ClearingHouse.sol` — cycle deadlines (lines 136–138).

Description. INV-1 requires `withdrawalGracePeriod` to exceed the worst-case time to cycle finality plus a seizure margin. Otherwise, a debtor can withdraw collateral before the operator can seize it. On-chain, `finalizeWithdrawal` checks only `block.timestamp >= request.timestamp + withdrawalGracePeriod`. It does not check settlement deadlines or reserve collateral for outstanding guarantees.

The request stores only `{timestamp, amount}`, and finalization reads the current global grace period. Lowering the grace period through `setWithdrawalGracePeriod` therefore shortens the wait for existing requests as well as new ones. If a runtime change breaks the timing invariant, the off-chain service logs a warning but continues issuing guarantees. Governance delays can also be configured down to zero (4MCA-H08).

Conditions. Production delays governance execution, which prevents an immediate grace reduction and makes changes observable before they take effect. The delay does not prevent retroactive changes or enforce the relationship between withdrawal and settlement deadlines (Section 6.3).

Recommendation

Store the effective grace period in each `WithdrawalRequest`, so reductions apply only to new requests. Enforce an on-chain minimum, and consider making cycle-finality timing available to `Core4Mica` for an on-chain check. Stop off-chain issuance if a runtime grace change violates the invariant. Enforce a non-zero governance execution delay (4MCA-H08).

4MCA-M03 Configuration permits non-finalized scanning despite incomplete reorg handling

Medium

INV-7, INV-8

✓ Fixed #134

Rated Medium because it requires a configuration override that is not enabled in production (`FINALIZED_HEAD_DEPTH = 0`). If the override is enabled, reorgs can leave persistent ledger inconsistencies and duplicate cycle events. The more severe ETH double-credit scenario, which permits unbacked guarantees, is reported separately as 4MCA-H04.

Location: `core/src/config.rs` — `FINALIZED_HEAD_DEPTH` (lines 80–83) and `validate` (116–128), which warns without rejecting the setting.

Location: `core/src/ethereum/scanner.rs` — the depth heuristic (lines 261–275), `maybe_handle_reorg` (484–506), the ~25-block rescan (121–124), and cursor-only reorg detection (493–495).

Description. Configuration validation requires `finalized` mode, but merely warns when `FINALIZED_HEAD_DEPTH > 0`. With that override, `confirmed_head` becomes `latest - depth`, which can include blocks that later reorganise. Operators may enable it when a chain's finalized head stops advancing or carry it over from development settings.

Reorg handling is incomplete. `maybe_handle_reorg` deletes only the cursor; the helpers for rolling back application state are unused. Both deduplication layers include `block_hash`, so re-mined events can be applied twice, including a defaulter's collateral debit. Reorg detection compares only the cursor block's hash, so it misses reorgs that do not affect that block, and the rescan covers approximately 25 blocks.

Recommendation

Make `validate` reject `FINALIZED_HEAD_DEPTH > 0` unless the operator explicitly acknowledges the reorg risk (4MCA-H04). Use a reorg-stable identity for event and collateral deduplication (`tx_hash + log_index`), and/or use the existing helpers to roll back application state. Compare multiple stored blocks during reorg detection, rather than only the cursor block.

4MCA-M04 An event that repeatedly fails processing can halt all chain indexing

Medium

INV-7, INV-5

✓ Fixed #137

All chain indexing can stop, and the off-chain state falls further behind the chain over time.

Location: `core/src/ethereum/scanner.rs` — the handler loop, delete-on-failure, and `return Err` (lines 362–441); the cursor upsert, which runs only after the whole batch (182–188); and `is_retryable_handler_error` (509–516).

Description. If a handler returns an error other than `UserBalanceLockConflict` or `DatabaseFailure`, the scanner deletes the newly stored event and returns `Err`. This aborts the batch before its cursor is updated. The next run reads the same range, reinserts the event, and fails again.

An event that consistently fails processing can therefore halt the entire scanner indefinitely. Examples include malformed logs, `checked_sub` underflows in collateral handlers, version-cast overflows, and non-transient RPC errors. There is no mechanism to record the failure separately and skip the event after repeated attempts. All later on-chain events remain unapplied until someone intervenes.

Recommendation

After N failed attempts, record the event separately, alert the operator, and advance past it so indexing can continue. Expand the retryable-error set where safe. Errors that indicate a code defect, such as a collateral-handler underflow, should raise an alert.

4MCA-M05 A crash between storing and handling an event permanently skips its effects

Medium

INV-8

✓ Fixed #138

A crash at a specific point can silently omit an event that will never be reprocessed, leaving the off-chain ledger inconsistent with the chain.

Location: `core/src/ethereum/scanner.rs` — `store` (lines 328–354), `handle` (362–442), and the end-of-batch cursor `upsert` (182–188).

Description. For each log, the scanner commits a `blockchain_event` row and then runs the handler, in separate database operations. A crash between these steps leaves the event stored, its effects unapplied, and the cursor unchanged. On restart, the scanner rescans the range, receives `inserted = false` because the row exists, and skips the handler.

The event's collateral credit, debit, or cycle transition is then permanently omitted from the off-chain ledger. Most handlers wrap their own writes in a transaction, but that does not close the gap between event storage and handler execution. The collateral-deposit handler also performs its writes directly on the connection without a transaction.

Recommendation

Run the handler in the same transaction that records the event. Alternatively, mark the event pending until the handler commits, then mark it processed. A crash must leave unprocessed events eligible for retry.

4MCA-M06 Off-chain processing ignores SettlementSkipped events

Medium

INV-5

✓ Fixed #133

Missing skip information prevents monitoring and recovery of the settlement failures in 4MCA-H06, including the triggers in 4MCA-H03 and 4MCA-H09.

Location: `contracts/src/ClearingHouse.sol` — `SettlementSkipped` is emitted at lines 231, 237, 253, 275, 281, 286, and 306.

Location: `core/src/ethereum/scanner.rs` — the scanner decodes only `CycleCommitted`, `DebtorPaid`, `CreditorClaimed`, `DebtorDefaulted`, and `CycleFinalized` (lines 377–389).

Location: `event_data.rs`, lines 128–204.

Description. The scanner does not decode `SettlementSkipped`. The operator pipeline therefore does not record which participant was skipped or the reason: already resolved, invalid proof, failed seizure, insufficient liquidity, or failed credit. Combined with the one-time settlement submission in 4MCA-H06, this prevents automatic detection and recovery of stalled cycles.

Recommendation

Decode and persist `SettlementSkipped`, alert on it, and automatically retry the skipped seizure or credit (subject to the 4MCA-H05 and 4MCA-H03 fixes), or flag it for manual intervention.

4MCA-M07 Withdrawal requests ignore locked collateral and can race with guarantee issuance**Medium**

INV-3, INV-1

✓ Fixed #141

The withdrawal checks provide no additional protection against the collateral shortfall described in 4MCA-M02. Concurrent requests and guarantee locks can also over-commit collateral and halt indexing when a withdrawal finalizes.

Location: `core/src/persist/repo/withdrawals.rs` — the request checks only `amount > total` (lines 49–55); finalization debits `total`, preserving `locked` (228–241).

Location: `core/src/persist/repo/guarantees.rs` — the lock uses `free = total - locked - pending` (lines 54–64).

Location: `balances.rs` — the version CAS (lines 76–120).

Location: `m20250901_000002` — migration with the `locked <= total` CHECK (lines 101–102).

Description. Withdrawal requests check only `amount <= total`, without subtracting collateral already locked for guarantees. Issuance uses `free = total - locked - pending`, but the on-chain vault has no corresponding locked balance. A user can therefore request a full withdrawal despite outstanding guarantees; the grace period is the only opportunity to seize that collateral before withdrawal (4MCA-M02).

Requests also bypass the balance version CAS. They read `total` and insert a `Withdrawal` row without updating `UserAssetBalance`. A withdrawal request and a guarantee lock can each pass their checks concurrently, leaving `locked + pending > total`.

When finalization reduces `total` below `locked`, the database's `locked <= total` constraint rejects the `CollateralWithdrawn` handler. Repeated retries fail, stalling the scanner (4MCA-M04). The constraint prevents silent ledger corruption, but indexing stops.

Recommendation

Reserve `locked` against withdrawal requests off-chain (check `amount <= total - locked`), and route the withdrawal request through, or otherwise serialise it with, the balance version CAS so that request and lock cannot interleave to over-commit. Also address the timing and reservation issues in 4MCA-M02.

5.4 Low findings

4MCA-L01 Incomplete V2 lifecycle lets payers withdraw collateral without paying recipients

Low

INV-5, INV-2

✓ Fixed #146

If V2 were enabled, every V2 guarantee would expose recipients to non-payment, whether or not the payer acted deliberately. The incomplete lifecycle prevents settlement, and 4MCA-I01 prevents the recipient from verifying the guarantee through the on-chain decoder.

Location: `core/src/service/guarantee.rs` — the `PendingValidation` status (lines 380–388), and `finalize_guarantee_payable`, `dispute_guarantee`, and `cancel_guarantee` (286–323), which have *no production caller*.

Location: `core/src/service/netting.rs` — `netting` reads only `FinalizedPayable` (line 141).

Location: `persist/repo/guarantees.rs`, lines 355–385.

Location: `core/src/service/clearing.rs` — `seizure` fires only from `process_defaulted_debtor` (lines 455–528).

Description. V2 guarantees start in `PendingValidation`. The functions that can change that status (`finalize_guarantee_payable`, `dispute_guarantee`, and `cancel_guarantee`) have no production callers. A search of `core`, `crates`, and `sdk` confirmed this.

`Netting` reads only `FinalizedPayable` guarantees, so a pending guarantee never contributes a debtor position to a committed cycle and never settles. It therefore never reaches the default processing associated with `process_defaulted_debtor` and `DebtorDefaulted`. The payer's on-chain collateral remains available for withdrawal after the grace period because off-chain locks do not restrict on-chain withdrawals (4MCA-M07).

A payer can thus keep the goods or services received from the recipient and withdraw the collateral without paying. The recipient's signed guarantee never settles, and its on-chain verification also fails because of 4MCA-I01.

Conditions. V2 is disabled in production. According to the confirmed configuration, it is also unusable on Base mainnet because the chain has no ERC-8004 validation registry (Section 6.3). The finding is rated Low under those conditions and must be resolved before V2 is enabled.

Recommendation

Keep V2 issuance disabled until its lifecycle driver (4MCA-L02) and decoder (4MCA-I01) are fixed. The driver must observe validation results and call `finalize_guarantee_payable` or `dispute`. A timeout must call `cancel` to release collateral when validation does not resolve. Add an integration test covering `issue → validate → finalize → net → settle`.

4MCA-L02 V2 guarantees remain in PendingValidation because no code advances their status**Low**

INV-5, INV-3

✓ Fixed #146

If V2 were enabled, every V2 guarantee would remain unsettled and permanently consume the payer's free issuance capacity. Together with the decode failure in 4MCA-I01, this leaves the V2/ERC-8004 flow unusable and permits the recipient loss described in 4MCA-L01.

Location: `core/src/service/guarantee.rs` — `PendingValidation` (lines 380–388) and `finalize_guarantee_payable`, `dispute_guarantee`, and `cancel_guarantee` (286–323).

Location: `core/src/service/cycle.rs` — `short-circuit` (lines 139–157).

Location: `core/src/persist/repo/guarantees.rs` — `netting` reads only `FinalizedPayable` (lines 355–385).

Description. V2 guarantees start in `PendingValidation`, but no production code calls `finalize_guarantee_payable`, `dispute_guarantee`, or `cancel_guarantee` to advance them. A search across `core`, `crates`, and `sdk` found only their definitions: no HTTP route, scheduler task, event handler, or test invokes them.

Netting includes only `FinalizedPayable` guarantees. Pending V2 guarantees never settle, and a cycle containing only V2 guarantees is marked `Finalized` without resolving them. The collateral locked at issuance can be released only by cancellation or netting, neither of which is reached. The payer's off-chain issuance capacity therefore remains consumed.

Conditions. V2 is disabled in production and unusable on the target chain (Section 6.3). The finding is rated Low under that configuration, but must be fixed before activation.

Recommendation

Implement the V2 finalization driver: an event-driven watcher (or an operator endpoint) that observes the on-chain ERC-8004 validation result and calls `finalize_guarantee_payable` (or `dispute`), plus a timeout path to `cancel` that releases collateral for validations that never resolve. Add an integration test exercising `issue → validate → finalize → net → settle`, alongside the 4MCA-I01 fix. Until the driver exists, reject V2 requests at issuance.

4MCA-L03 Weak JWT signing secrets are accepted with only a warning

Low

INV-10

✓ Fixed #134

A weak production secret permits authentication bypass, user impersonation, and access to admin API functions. The finding is rated Low because it requires the operator to configure a weak secret; the main development default is already rejected.

Location: `core/src/config.rs` — the `AUTH_JWT_SECRET` validation (lines 366–378).

Description. Startup rejects an empty JWT secret and one insecure development default. It accepts another placeholder value and secrets shorter than 32 bytes with only a `warn!`. HS256 uses a shared secret; an attacker who captures a token can attempt to brute-force a weak secret offline. Recovering it allows the attacker to forge tokens with arbitrary `sub`, `role`, and `scopes`, including `role=admin`, which permits user suspension and cross-account reads.

Conditions. Depends on the entropy of the production `AUTH_JWT_SECRET`, which is environment-driven and not visible in the repository.

Recommendation

Reject the placeholder and secrets shorter than 32 bytes at startup.

4MCA-L04 BLS key handling exposes command-line input and leaves input buffers uncleared

Low

INV-10

✓ Fixed #142

Local access to the operator host or process table can expose the key used to sign guarantees. The local-access requirement limits the rating to Low.

Location: crates/crypto/src/bin/print_vk.rs, lines 21–29.

Location: crates/crypto/src/bls.rs — `KeyMaterial::from_bytes` (lines 322–325).

Description. Disclosure of the BLS signing key permits guarantee forgery. The `print-vk` utility accepts it through `std::env::args()`, exposing it to process listings, `/proc`, and shell history, even though environment-variable input is available. Separately, `KeyMaterial::from_bytes` accepts a `&[u8]` without zeroizing the caller's buffer. The `FromStr` path does use Zeroizing.

Recommendation

Have `print-vk` read the key from an environment variable or stdin. Remove the command-line argument, or clearly document and discourage its use. Have `from_bytes` accept a `Zeroizing/SecretBox` input, or document that the caller must zeroize the buffer. Supply `BLS_PRIVATE_KEY` through a secret manager, keeping it out of command-line arguments and shell history.

4MCA-L05 Merkle hashing does not distinguish leaves from internal nodes

Low

INV-4, INV-6

✓ Fixed #143

Rated Low because current production use is safe, but the API does not enforce the constraints that make it safe. Accepting caller-supplied leaves, shortening leaf preimages to at most 64 bytes, or making them variable-length could permit false settlement-membership proofs. This could let a non-participant claim a credit or a debtor prove a smaller debit.

Location: `crates/crypto/src/merkle.rs` — `verify_proof` (lines 66–71) and the doc-comment caveat (23–37).

Location: `core/src/evm/clearing.rs` — the production leaf derivation (lines 24–43).

Description. `MerkleTree` and `verify_proof` use bare B256 values without distinguishing leaf hashes from internal-node hashes. A caller can therefore present an internal-node hash as a leaf and obtain a valid membership result for a value that was never a leaf.

Current production use avoids this ambiguity. The only leaf preimage, `hash_participant_leaf`, is fixed at 224 bytes, distinct from the 64-byte internal-node preimage. `verify_proof` is called only in tests. On-chain, the `ClearingHouse` recomputes leaves from typed arguments rather than accepting a supplied hash (Section 5.1).

Recommendation

Use separate hash domains for leaves and internal nodes: prefix leaves with `0x00` and nodes with `0x01`. This differs from OpenZeppelin's sorted-pair scheme and requires a matching on-chain change. At minimum, restrict `verify_proof` to a typed leaf-preimage input instead of a raw B256, and add a regression test that rejects an internal-node value presented as a leaf.

4MCA-L06 EIP-712 request domain omits the verifying contract, enabling cross-deployment replay

Low

INV-10

✓ Fixed #144

Exploitation requires a second 4Mica deployment with the same domain values on the same chain, which limits the rating to Low. Requests are verified off-chain, and issued guarantees are scoped by `req_id` and `cycle`. Adding the contract address to the request domain would prevent cross-deployment replay.

Location: `core/src/evm/guarantee.rs` — the `eip712_domain!` declaration over `name`, `version`, and `chainId` (lines 96–100).

Description. The EIP-712 [11] domain for guarantee requests binds `name`, `version`, and `chainId`, but omits `verifyingContract` and a salt. A request signature is therefore valid for any 4Mica deployment with the same name, version, and chain id. The separate guarantee-claims domain correctly binds the contract address (Section 5.1).

Recommendation

Add `verifyingContract` (the Core4Mica address) to the EIP-712 request domain so request signatures are deployment-bound, and coordinate the same change in the SDK and wallet signing path.

4MCA-L07 Default collateral debits ignore the status update result, allowing duplicate debits after a reorg

Low

INV-8, INV-2

✓ Fixed #145

Rated Low because finalized-only scanning prevents this reorg scenario in production (4MCA-M03; Section 6.3). Checking the CAS result would provide the same protection against duplicate events as the credit path.

Location: `core/src/service/clearing.rs` — `process_defaulted_debtor` (lines 455–528) versus `process_credit_claim` (384–450).

Location: `m20260507_000007` — migration with the collateral event-identity index.

Description. The credit path updates collateral only when its `Claimable` → `Claimed` CAS changes the status, so duplicate events do not repeat the credit. The default path ignores the result of its `Unpaid` → `Defaulted` CAS and debits collateral regardless.

That debit relies only on event-identity deduplication, which includes `block_hash`. If a reorg re-mines `DebtorDefaulted` into another block, the new block hash bypasses deduplication. The handler debits collateral again or underflows and stalls the scanner.

Recommendation

Apply the collateral debit and guarantee settlement transition only when the `Unpaid` → `Defaulted` CAS returns changed, matching the credit path. This makes default processing idempotent under event redelivery.

5.5 Informational findings

4MCA-I01 Guarantee-claim ABI mismatch breaks deprecated on-chain verification

Info

INV-6

✓ Fixed #131

`verifyAndDecodeGuarantee` fails for every guarantee produced by the core service. The 4Mica team confirmed that this entry point is deprecated and unused; settlement uses Merkle proofs, so no active settlement flow is blocked. Decoding fails closed, preventing acceptance of a malformed guarantee through this mismatch. The finding is therefore Informational.

The mismatch also shows why separately maintained encodings need cross-language tests. The Merkle leaf, version envelope, and validation hashes are also maintained separately in each language. Appendix A specifies regression vectors for these encodings.

Location: `crates/rpc/src/guarantee/codec.rs` — `GuaranteeClaimsV1` and `GuaranteeClaimsV2` (lines 11–44).

Location: `contracts/src/Core4Mica.sol` — `Guarantee` (lines 17–28) and the inline V1 decode (637–643).

Location: `contracts/src/ValidationRegistryGuaranteeDecoder.sol` — `GuaranteeV2` (lines 22–42).

Description. The core service BLS-signs the envelope `abi_encode(version, GuaranteeClaimsVN)`. On-chain, `verifyAndDecodeGuarantee` hashes that envelope, verifies the signature, then decodes the inner bytes into a Solidity struct. The signature check is correct (Section 5.1), but the structs disagree.

Rust's `GuaranteeClaimsV1` has nine static fields; Solidity's `Guarantee` has ten. Solidity includes `totalAmount` at position 7, which Rust omits, shifting every later field. Rust produces 288 bytes, while the decoder expects a 320-byte tuple, so decoding fails with a buffer overrun. V1 always uses the inline ten-field decoder: its decoder address is initialized to `address(0)`, and `configureGuaranteeVersion` rejects a non-zero V1 decoder.

V2 has the same missing field: Rust's `GuaranteeClaimsV2` has 18 fields and a 672-byte inner encoding, while Solidity's `GuaranteeV2` has 19 fields. The shift also corrupts the dynamic-string offset, causing V2 decoding to fail.

Empirical confirmation. We ran the Rust encoder and decoded its output with the `alloy` ABI codec. V1 produced 288 bytes and failed the ten-field decode with a buffer overrun. The missing 32-byte `totalAmount` word accounts for the difference ($288 + 32 = 320$). V2 produced 672 bytes and failed on a malformed offset. `alloy` follows the same ABI specification as Solidity; reproduction against the on-chain decoder in Foundry remains outstanding.

Conditions. The on-chain verification path is deprecated and unused (Section 6.3).

Recommendation

If the deprecated path is retained, reconcile the guarantee-claim struct, adding `total_amount` to the Rust claim structs at position 7 (between `amount` and `asset`), or removing it from the Solidity structs (a contract change requiring redeployment). Add the cross-language tests in Appendix A to Continuous Integration (CI): decode a Rust-encoded guarantee through `verifyAndDecodeGuarantee`, and check parity for the Merkle leaf, version envelope, and validation hashes.

Conclusions

The review found the core BLS, netting, collateral-conservation, and authentication mechanisms to be correct (Section 5.1). It also identified nine High, seven Medium, seven Low, and one Informational finding. At the audited revision, the main risks were stalled settlement cycles and operator or administrator key compromise. An encoding mismatch and an incomplete V2 lifecycle affected paths that are deprecated or disabled in the confirmed deployment.

4Mica has since fixed all 24 findings, and Symbolic Software reviewed each fix (Section 6.1). Several fixes change the contracts, so they take effect only after a new deployment, and this report does not establish that production has been redeployed. Section 6.2 lists the measures that apply until then and the recommendations that remain open.

6.1 Remediation

Table 6.1 lists the pull requests that fix the findings, ordered by pull request number. Each finding's header links to the same pull request.

Pull request #168 also replaces BLS guarantee signatures with ECDSA EIP-712 signatures and removes the BLS code. The BLS results in Section 5.1 and Appendix A, and finding 4MCA-L04, therefore describe the audited revision. Changes to 4mica-core that no finding links to, such as the removal of V2 guarantees in pull request #151 and the treasury and controller split in #169, are outside the scope of this report.

6.2 Remaining actions

6.2.1 Until the fixes are deployed

- Set `yieldFeeBps = 0`. The yield fee was enabled in production at the time of the review, so the fee-reallocation underflow (4MCA-H05) and the cycle-freeze it triggers (4MCA-H09) remain possible until the fixed contracts are deployed. Cycles that stalled before then still require recovery.
- Keep the V2/ERC-8004 flow disabled (4MCA-L01, 4MCA-L02, 4MCA-I01). V2 is also unusable on the target chain.
- Keep finalized-only chain scanning (`FINALIZED_HEAD_DEPTH = 0`), as deployed. This prevents the reorg failures described in 4MCA-H04 and 4MCA-M03. The fixed scanner no longer has this setting.

6.2.2 Before mainnet

- **Governance and keys.** The deployment script now enforces the hardened configuration by default. It applies only to new deployments and is skipped in development mode. Assign `ACCESS_MANAGER_ADMIN` to a multisig with a timelock. Give the `GOVERNANCE`, `TREASURY`, `GUARDIAN`, and

Table 6.1. Pull requests that fix the findings in this report.

Pull request	Merged	Findings	Change
✓ Fixed #131	June 23	4MCA-I01	Removes <code>totalAmount</code> from the Solidity <code>Guarantee</code> and <code>GuaranteeV2</code> structs to match the Rust encoding, and adds Rust and Solidity test vectors for both guarantee versions.
✓ Fixed #132	June 29	4MCA-M01	Limits balance reads to the caller's own account, with an override for the admin and facilitator roles.
✓ Fixed #133	July 2	4MCA-H03, 4MCA-H05, 4MCA-H06, 4MCA-H09, 4MCA-M02, 4MCA-M06	Moves seized stablecoin positions into an escrow without an Aave withdrawal, fixes the fee-reallocation underflow, and adds a <code>Shortfall</code> terminal state with pro-rata payouts. Also resubmits skipped seizures, decodes <code>SettlementSkipped</code> , and stores the grace period in each withdrawal request, with an on-chain minimum.
✓ Fixed #134	July 3	4MCA-H04, 4MCA-M03, 4MCA-L03	Removes the finalized-head depth setting, so production scanning always uses finalized blocks, and reconciles ETH balances from the chain. Production startup also stops on a weak JWT secret.
✓ Fixed #135	July 6	4MCA-H07	Requires a future submission deadline at commit, and caps settled debits and credits at the committed totals.
✓ Fixed #136	July 7	4MCA-H02, 4MCA-H08	Makes the deployment script require, by default, a contract admin separate from the deployer, minimum admin and grant delays, and role holders other than the deployer. The deployer gives up its admin role.
✓ Fixed #137	July 9	4MCA-M04	Sets aside events that fail permanently, and retries transient errors a bounded number of times.
✓ Fixed #138	July 10	4MCA-M05	Marks each event pending until its handler completes, and reprocesses pending events on restart.
✓ Fixed #141	July 13	4MCA-M07	Makes withdrawal requests update the balance version, so they cannot interleave with guarantee locks, and reports requests that exceed free collateral.
✓ Fixed #142	July 14	4MCA-L04	Reads the BLS key only from the environment into zeroized memory, and makes <code>from_bytes</code> take zeroizing input.
✓ Fixed #143	July 16	4MCA-L05	Double-hashes Merkle leaves in Rust and in the <code>ClearingHouse</code> , accepts leaves only as a typed <code>LeafHash</code> , and adds cross-language Merkle test vectors.
✓ Fixed #144	July 15	4MCA-L06	Adds the verifying contract to the EIP-712 request domain in the core service and the Rust SDK.
✓ Fixed #145	July 15	4MCA-L07	Applies the default-path collateral debit only when the <code>Unpaid</code> → <code>Defaulted</code> status update succeeds.
✓ Fixed #146	July 16	4MCA-L01, 4MCA-L02	Adds a scheduled driver that finalizes, disputes, or cancels pending V2 guarantees, and keeps V2 issuance off unless explicitly enabled.
✓ Fixed #168	October 15	4MCA-H01	Requires the debtor's signed EIP-712 payment authorizations before a default can seize collateral, up to the authorized amount.

operator roles separate holders: the script requires holders other than the deployer, but not holders distinct from each other. Protect the operator key with an HSM or multisig, give the operator role a non-zero execution delay, and verify these settings against the live deployment (4MCA-H01, 4MCA-H02, 4MCA-H08).

- **Testing.** Pull requests #131, #143, and #168 add cross-language test vectors for guarantees, Merkle proofs, and payment authorizations. Foundry invariant and fuzz suites for the vault and ClearingHouse, proptest round-trips for the codecs, and Anvil reorg simulation remain recommended, run in CI.
- **Configuration details.** Confirm the production details listed at the end of Section 6.3.

6.2.3 Further recommendations

The fixes resolve the findings as reported. The following recommendations from the findings are not covered by them:

- **Settlement liveness.** A cycle can be marked `Shortfall` only after every debt is resolved. A debtor entry that can never resolve, for example because its proof is invalid, still prevents the cycle from reaching a terminal state. Add a path for that case (4MCA-H06).
- **Root binding.** Pull request #168 binds debtor entries to signed payment authorizations; creditor entries are not yet bound to obligations. A minimum payment window at commit would also help, since any future deadline is accepted (4MCA-H01, 4MCA-H07).
- **Solvency window.** Governance can lower the on-chain grace-period minimum, subject to its execution delay. Off-chain, a withdrawal request that exceeds free collateral is reported rather than refused, so the on-chain grace period remains the protection (4MCA-M02, 4MCA-M07).
- **Reorg handling.** Event deduplication still includes the block hash. Use a reorg-stable identity if a non-finalized scanning mode is ever used in production (4MCA-H04, 4MCA-M03).

6.3 Deployment configuration

The severity of several findings depends on production configuration and design intent that are not visible in the source repository. The 4Mica team confirmed the following for the deployment at the time of the review, and the ratings and conditions in this report reflect that configuration:

Governance and keys (4MCA-H01, 4MCA-H02, 4MCA-H08)

The operator and the AccessManager admin are externally owned accounts protected by execution delays. A single compromised key can still authorise the transfer of user funds after the delay. Before mainnet, move the admin to a multisig with a timelock, separate role holders from the deployer, and configure admin and grant delays.

Yield fee (4MCA-H05, 4MCA-H09)

The yield fee is enabled in production, so the fee-reallocation underflow and the cycle-wide freeze it triggers are possible. Setting `yieldFeeBps = 0` is the interim mitigation until the fixed contracts are deployed.

V2 activation (4MCA-L01, 4MCA-L02, 4MCA-I01)

The V2 validation flow is disabled and, according to the confirmed configuration, unusable on the target chain: Base mainnet has no ERC-8004 validation registry.

Chain finality (4MCA-H04, 4MCA-M03)

The chain scanner runs in finalized-only mode (`FINALIZED_HEAD_DEPTH = 0`), so blocks that could reorganise are not processed. The reorg findings cannot occur under this configuration.

On-chain verification (4MCA-I01)

`verifyAndDecodeGuarantee` is deprecated and not relied upon (on-chain settlement uses Merkle proofs), which is why the guarantee-claim ABI mismatch is informational.

The fixes answer two of the design questions raised during the review. A failed seizure now leads to pro-rata loss sharing through the `Shortfall` state (4MCA-H06), and `payment:read` lets a wallet read only its own balance, with an override for the `admin` and `facilitator` roles (4MCA-M01). The following details remain to be confirmed before a mainnet deployment holds real user funds:

- The exact production execution, admin, and grant delays, and the timeline for moving governance to a multisig with a timelock and separate role holders (4MCA-H01, 4MCA-H02, 4MCA-H08).
- The Aave deployment targeted, whether its addresses-provider is an upgradeable proxy outside 4Mica's control, and how reserve-illiquidity or reserve-pause risk is assessed for the whitelisted assets (4MCA-H03).
- The stablecoin whitelist and the transfer semantics of each listed token, and the production values of the configuration parameters referenced throughout this report.

6.4 Closing remarks

4Mica has fixed every finding in this report, and Symbolic Software reviewed each fix. Most of the remaining work is operational. Deploy the fixed contracts and `core-service`, and keep the measures in Section 6.2.1 in place until then. Hold the governance roles in separately protected keys, confirm the production details in Section 6.3, and consider the recommendations in Section 6.2.3.

About Symbolic Software



Symbolic Software, established in Paris, France, in 2017, is a software consultancy specializing in applied cryptography and software security. The firm has completed over 300 cryptographic software audits for the European information security sector, and publishes peer-reviewed cryptographic research software.

Its audits have covered the cryptographic components of global platforms, including password managers and cryptocurrencies. The company develops Verifpal® and Noise Explorer, research software for cryptographic engineering that has supported peer-reviewed scientific publications. Symbolic Software has worked with Cure53 and the Linux Foundation, and has audited MetaMask and COVID-19 contact tracing applications in Europe.

<https://symbolic.software>

Bibliography

- [1] Vectorized. *Solady: Optimized Solidity snippets (BLS.sol, commit 208e4f31)*. 2025. URL: <https://github.com/Vectorized/solady/blob/208e4f31cfae26e4983eb95c3488a14fdc497ad7/src/Utils/ext/ithaca/BLS.sol> (visited on 10/15/2026).
- [2] OpenZeppelin. *Cryptography Utilities: MerkleProof and Hashes (Contracts 5.x API)*. 2024. URL: <https://docs.openzeppelin.com/contracts/5.x/api/utils/cryptography#MerkleProof> (visited on 10/15/2026).
- [3] Wayne Chang and Gregory Rocco and Brantly Millegan and Nick Johnson and Oliver Terbu. *ERC-4361: Sign-In with Ethereum*. Status: Final. Ethereum Improvement Proposals. Oct. 2021. URL: <https://eips.ethereum.org/EIPS/eip-4361> (visited on 10/15/2026).
- [4] Supranational. *blst: Multilingual BLS12-381 signature library*. Version 0.3.16. 2025. URL: <https://github.com/supranational/blst> (visited on 10/15/2026).
- [5] Marco De Rossi and Davide Crapis and Jordan Ellis and Erik Reppel. *ERC-8004: Trustless Agents*. Status: Draft. Ethereum Improvement Proposals. Aug. 2025. URL: <https://eips.ethereum.org/EIPS/eip-8004> (visited on 10/15/2026).
- [6] Emilio Frangella and Lasse Herskind. *Aave V3 Technical Paper*. Aave. Jan. 2022. URL: https://github.com/aave/aave-v3-core/blob/master/techpaper/Aave_V3_Technical_Paper.pdf (visited on 10/15/2026).
- [7] Francisco Giordano and Matt Condon and Philippe Castonguay and Amir Bandehali and Jorge Izquierdo and Bertrand Masius. *ERC-1271: Standard Signature Validation Method for Contracts*. Status: Final. Ethereum Improvement Proposals. July 2018. URL: <https://eips.ethereum.org/EIPS/eip-1271> (visited on 10/15/2026).
- [8] Dan Boneh and John Bradley and Sergey Gorbunov and Riad S. Wahby and Hoeteck Wee and Christopher A. Wood and Zhenfei Zhang. *BLS Signatures (draft-irtf-cfrg-bls-signature-07)*. Internet-Draft, IRTF Crypto Forum Research Group. Work in progress. July 2026. URL: <https://datatracker.ietf.org/doc/draft-irtf-cfrg-bls-signature-07/> (visited on 10/15/2026).
- [9] Armando Faz-Hernández and Sam Scott and Nick Sullivan and Riad S. Wahby and Christopher A. Wood. *RFC 9380: Hashing to Elliptic Curves*. Internet Research Task Force (IRTF), Crypto Forum Research Group. Aug. 2023. DOI: 10.17487/RFC9380. URL: <https://www.rfc-editor.org/rfc/rfc9380> (visited on 10/15/2026).

- [10] Alex Vlasov and Kelly Olson and Alex Stokes and Antonio Sanso. *EIP-2537: Precompile for BLS12-381 curve operations*. Status: Final. Ethereum Improvement Proposals. Feb. 2020. URL: <https://eips.ethereum.org/EIPS/eip-2537> (visited on 10/15/2026).
- [11] Remco Bloemen and Leonid Logvinov and Jacob Evans. *EIP-712: Typed structured data hashing and signing*. Status: Final. Ethereum Improvement Proposals. Sept. 2017. URL: <https://eips.ethereum.org/EIPS/eip-712> (visited on 10/15/2026).
- [12] OpenZeppelin. *Access Control: Using AccessManager (Contracts 5.x)*. 2024. URL: <https://docs.openzeppelin.com/contracts/5.x/access-control#using-accessmanager> (visited on 10/15/2026).

Cross-Language Consistency Matrix and Recommended Test Vectors

Rust and Solidity must agree byte-for-byte on the constants, encodings, and hash preimages used by the guarantee path. These definitions are maintained separately in each language. Section A.1 records the comparison results. Section A.2 specifies cross-language tests for 4MCA-I01 and for the properties verified in Section 5.1.

A.1 Consistency matrix

Table A.1 lists each shared definition, its Rust and Solidity locations, and the comparison result at the audited revision. All agree except the guarantee-claim struct (4MCA-I01). Pull request #131 fixed the mismatch by removing `totalAmount` from the Solidity structs.

We verified each “Equal” row manually against the pinned library sources. The clearing leaf and Merkle tree also have Rust unit tests, but none of the rows has a cross-language known-answer test to detect changes in Solidity or its libraries. The tests below would provide that check.

A.2 Recommended test vectors

We recommend adding the following to CI, spanning both the Rust and Foundry suites. Pull requests #131 and #143 added cross-language test vectors for the guarantee path and the Merkle tree, covering valid inputs for tests 3 and 6. Pull request #168 later replaced the BLS signatures with ECDSA, so tests 1, 2, and 4 apply to the audited revision only.

1. **G1 verification-key known-answer test (Rust).** For a fixed secret key, assert `public_key().to_solidity_words()` equals the pinned `[VK_X0, VK_X1, VK_Y0, VK_Y1]`. Include the standard generator as a reference vector (use `sk = 1`, so the public key is G_1 and the words equal the known generator encoding).
2. **G2 signature known-answer test (Rust).** For a fixed `(sk, msg)`, assert the compressed signature and its `to_solidity_words()` output equal pinned values.
3. **End-to-end verification test (Foundry).** Load a blst-produced `(VK, msg, sig)` fixture, set it as the version verification key, and assert `verifyAndDecodeGuarantee` succeeds and returns the correct fields for both the V1 inline path and the V2 decoder path. Check that it reverts for a signature with one bit flipped, a wrong verification key, and a tampered message. This test would have detected 4MCA-I01.
4. **DST guard.** Assert (in Foundry or a build step) that the Solady DST in use equals `BLS_SIG_BLS12381G2_XMD:SHA-256_SSWU_RO_NUL_`, and mirror it with a Rust test that checks the constant. A submodule update that changes the DST should fail this test.

Table A.1. Rust ↔ Solidity consistency matrix at the audited revision.

Shared definition	Status	Locations and notes
BLS DST	Equal	bls.rs:19 ↔ Solady BLS.hashToG2; reassembled from the two mstore chunks and confirmed identical.
G1/G2 point encoding, Fp2 order	Equal	bls.rs:85-184 ↔ EIP-2537 / Solady; confirmed by a known-answer check against the on-chain negatedG1Generator.
Guarantee-claim struct	Mismatch	codec.rs:11-44 ↔ Core4Mica.sol:17-28 and ValidationRegistryGuaranteeDecoder.sol:22-42; Rust omits totalAmount (4MCA-I01).
Version envelope (uint64, bytes)	Equal	codec.rs:99-117 ↔ Core4Mica.sol:622; embedded-version cross-check present.
Validation subject-hash preimage	Equal	validation.rs:17-60 ↔ ValidationRegistryGuaranteeDecoder.sol:165-171.
Validation request-hash preimage	Equal	validation.rs:27-80 ↔ ValidationRegistryGuaranteeDecoder.sol:173-187; includes the pre-hashed tag.
Binding-domain constants	Equal	validation.rs:9-10 ↔ ValidationBindingConstants.sol:5-7.
Core guarantee domain separator	Equal	off-chain builder ↔ Core4Mica.sol:154; binds chain id and contract address.
Clearing leaf preimage (224-byte)	Equal	evm/clearing.rs:24-43 ↔ ClearingHouse.sol:346-365; binds chain id and ClearingHouse address.
Merkle hashing (sorted-pair)	Equal	merkle.rs:54-60 ↔ OpenZeppelin Hashes.commutativeKeccak256; sorted-pair, odd-node, and empty/single-leaf semantics match.
Participant role enumeration	Equal	netting.rs:23-25 ↔ ClearingHouse.sol:32-34.

- 5. Validation-hash parity.** Assert that the Rust `compute_validation_subject_hash` and `compute_validation_request_hash` equal the on-chain `_computeValidationSubjectHash` and `_computeValidationRequestHash` for shared inputs, after fixing the V2 decode in 4MCA-I01.
- 6. Merkle compatibility test.** For a fixed leaf set, assert that the Rust `MerkleTree` root and a Rust-generated proof verify under an on-chain OpenZeppelin `MerkleProof` harness, including an odd leaf count.