UTEXO BRIDGE ENCLAVE SIGNER SECURITY AUDIT REPORT

OXORIO

AUGUST 12, 2026

1 EXECUTIVE SUMMARY

OXORIO

1.1 EXECUTIVE SUMMARY

This document presents the security audit conducted by Oxorio for the UTEXO Bridge enclave signer.

UTEXO Protocol develops a trust-minimized bridge connecting Bitcoin/RGB assets with EVM networks, with the stated design goal that even a compromise of the off-chain operators must not allow assets to be stolen from the bridge.

The enclave signer (`enclave-signer`) is the off-chain Rust component of this system, running inside an AWS Nitro Enclave. It generates and derives the bridge's key material, validates RGB consignments, SPV proofs, and EVM cross-checks before signing, produces the Bitcoin/RGB and EVM signatures that authorize releases, attests its identity through the Nitro Security Module, and replicates its seed across the cluster through an enclave-to-enclave cloning handshake.

The audit process involved a comprehensive approach, including manual code review, automated analysis, and extensive testing and simulations of the codebase to assess the project's security and functionality. The audit covered a total of 26 source files, encompassing 9,281 total lines (6,741 lines of code). The codebase was thoroughly examined, with the audit team referencing the provided protocol specification to address any questions regarding the expected behavior. For an in-depth explanation of the security audit methodology, please refer to the Security Assessment Methodology section of this document.

Throughout the audit, a collaborative approach was maintained with UTEXO Protocol to address all concerns identified within the audit's scope. Each issue has been either resolved or formally acknowledged by UTEXO Protocol, contributing to the robustness of the project.

As a result, following a comprehensive review, our auditors have verified that the UTEXO Bridge Enclave Signer, considering `dev@2df6d50` together with the audited remediation heads listed in the Project Brief, has met the security and functionality requirements established for this audit, based on the code and documentation provided, and operates as intended within the defined scope.

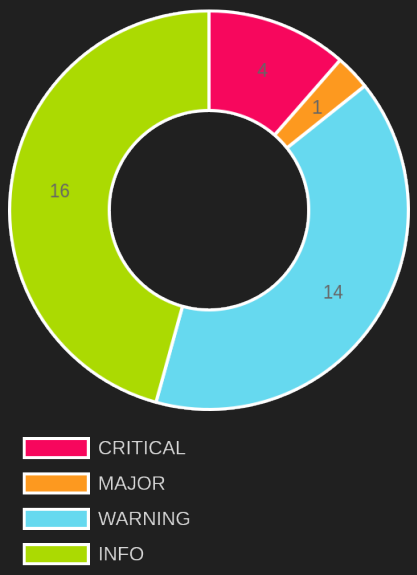
1.2 SUMMARY OF FINDINGS

The table below provides a comprehensive summary of the audit findings, categorizing each by status and severity level. For a detailed description of the severity levels and statuses of findings, see the Findings Classification Reference section.

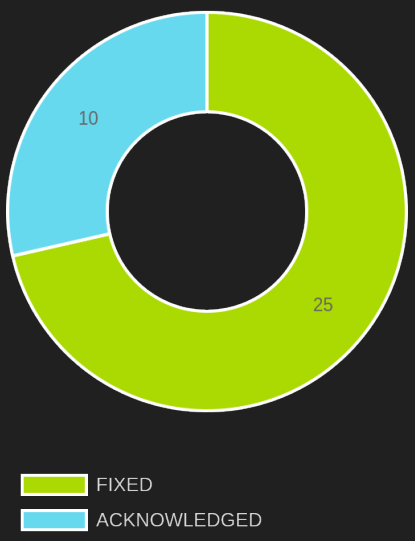
Detailed technical information on the audit findings, along with our recommendations for addressing them, is provided in the Findings Report section for further reference.

All identified issues have been addressed, with UTEXO Protocol fixing them or formally acknowledging their status.

Severity	TOTAL	NEW	FIXED	ACKNOWLEDGED	NO ISSUE
CRITICAL	**4**	0	3	1	0
MAJOR	**1**	0	1	0	0
WARNING	**14**	0	13	1	0
INFO	**16**	0	8	8	0
TOTAL	**35**	**0**	**25**	**10**	**0**



Issue distribution by severity



Issue distribution by status

2 AUDIT OVERVIEW

CONTENTS

1. EXECUTIVE SUMMARY 2

1.1. EXECUTIVE SUMMARY 3

1.2. SUMMARY OF FINDINGS 4

2. AUDIT OVERVIEW 5

2.1. DISCLAIMER 10

2.2. PROJECT BRIEF 11

2.3. PROJECT TIMELINE 12

2.4. AUDITED FILES 13

2.5. PROJECT OVERVIEW 15

2.6. CODEBASE QUALITY ASSESSMENT 16

2.7. FINDINGS BREAKDOWN BY FILE 18

2.8. CONCLUSION 19

3. FINDINGS REPORT 20

3.1. CRITICAL 21

C-01 The enclave infers security mode from empty fields and feature flags instead of enforcing an explicit production policy in config.rs, main.rs, evm_crosscheck.rs, psbt_crosscheck.rs, server.rs, attestation.rs, Cargo.toml 21

C-02 SignRawDigest signs arbitrary gas-key digests without a transaction-shape allowlist in server.rs, proto/enclave.proto, keys.rs 26

C-03 An empty rgb_asset_id bypasses the consignment-to-pinned-asset binding, allowing a foreign-asset burn to authorize a release in server.rs, evm_crosscheck.rs 29

C-04 The EVM recipient in fundsOut calldata is not bound to the RGB consignment's destination, allowing release redirection in evm_crosscheck.rs, rgb.rs 30

3.2. MAJOR 32

M-01 rgb-validation without spv: consignment anchoring trusts only the host-controlled Esplora in enclave/Cargo.toml 32

M-02 No binding or uniqueness of the RGB ↔ EVM operation identifiers, enabling double-spend in both bridge directions in evm_crosscheck.rs, psbt_crosscheck.rs, server.rs, proto/enclave.proto . 35
M-03 Signer-set rotation desync: the BTC/RGB multisig and the EVM signer set rotate independently in MultisigProxy.sol 39
M-04 Mint path (signPsbt) has no SPV / RGB / nonce / deadline → stale-data minting in enclave/src/server.rs 42
M-05 SignPsbt vanilla mode bypasses EVM-to-RGB authorization when evm_tx_hash is empty in psbt_crosscheck.rs, server.rs, keys.rs 46
M-06 SignPsbt bridge mode trusts listener-supplied EVM validity and finality flags in psbt_crosscheck.rs, server.rs, proto/enclave.proto 49
M-07 The enclave serves connections serially with no read or idle timeout in main.rs, server.rs, framing.rs 53
M-08 Cloning authorization rests on a static shared secret while reproducible PCRs cannot distinguish a legitimate enclave from an attacker's identical image in server.rs, attestation.rs, cloning.rs, main.rs 55
M-09 Inability to pass EVM request validation under a specific enclave configuration in evm_crosscheck.rs 60
3.3. WARNING 63
W-01 Malformed ABI tail in calldata: no dynamic offset/bounds validation in evm_crosscheck.rs . 63
W-02 Cross-layer selector drift: enclave whitelist wider than the on-chain allowlist in evm_crosscheck.rs 65
W-03 Cross-layer batch signature drift: executeBatch exists on-chain, enclave signs single-call only in enclave/src/signing/evm.rs 67
W-04 DoS-ordering: expensive RGB validation runs before the cheap consignment_hash check in enclave/src/server.rs 69
W-05 The EVM-lock-record predicate (the spec's same-OpId obligation) is delegated to the untrusted Listener in evm_crosscheck.rs 71
W-06 psbt_output_amount is supplied by the listener instead of being derived from the PSBT in psbt_crosscheck.rs, proto/enclave.proto 74
W-07 SignPsbt returns a successful response even when no inputs were signed in server.rs, keys.rs 77
W-08 The in-memory SPV header chain grows without an upper bound in chain.rs, server.rs 79

W-09 Ineffective typesystem check during consignment validation in rgb.rs 81
W-10 The attestation certificate-chain verifier omits BasicConstraints / KeyUsage / pathLen checks in attestation-verify/src/lib.rs 82
W-11 Header-staleness and request-deadline checks rely on a potentially host-settable enclave clock in server.rs, spv 83
W-12 The cloning replay guard is count-capped and rejects new handshakes when full, enabling an availability DoS in state.rs, server.rs 85
W-13 The replay guard records nonces before authenticity checks, and handle_get_attested_public_key is an arbitrary-nonce attestation oracle, enabling a secret-less cloning DoS in server.rs 86
W-14 The mainnet SPV checkpoint is not aligned to a retarget boundary, wedging the chain at the first boundary above it in spv/checkpoint.rs, spv/chain.rs 89
3.4. INFO 90
I-01 SignRawMessage signs arbitrary bytes with no domain / version / nonce in enclave/src/server.rs 90
I-02 dev-mode panic-path on short calldata in enclave/src/server.rs 92
I-03 Blocking Esplora HTTP on the signing path through the untrusted host in enclave/src/validation/rgb.rs 93
I-04 Per-field input sizes are bounded only by generic framing, enabling CPU/memory amplification in spv_crosscheck.rs, evm_crosscheck.rs, framing.rs 95
I-05 COSE verifier accepts DER signatures and does not check alg == ES384 in attestation-verify/src/lib.rs 97
I-06 Amount-binding is non-strict (≤) rather than the spec's strict == in evm_crosscheck.rs 99
I-07 BIP-86 account extended private keys are not zeroize-on-drop in enclave/src/keys.rs 101
I-08 The Esplora TCP forwarder exposes a generic local egress path inside the enclave in main.rs, vsock_forwarder.rs 104
I-09 consignment_hash is an integrity check, not authorization, in evm_crosscheck.rs 107
I-10 Selector-specific EVM validators rely on caller ordering instead of failing closed in evm_crosscheck.rs, server.rs 109
I-11 Cross-chain metadata is request/calldata-supplied rather than derived from enclave-validated facts in evm_crosscheck.rs, server.rs 111

I-12 SignEvm validates selected scalar fields instead of parsing the full fundsOut calldata into a typed intent in evm_crosscheck.rs 114
I-13 The master seed is held in memory for the whole enclave lifetime and replicated cluster-wide, maximizing key-compromise blast radius in keys.rs, state.rs, server.rs 117
I-14 Inability of the enclave to operate with tokens that use the standard decimals value of 18 in evm_crosscheck.rs 119
I-15 Suppression of warnings when fresh transactions are present in consignment in rgb.rs 121
I-16 No reproducible-build pipeline exists for independently deriving the expected PCR measurements in build/ 123
4. APPENDIX 125
4.1. SECURITY ASSESSMENT METHODOLOGY 126
4.2. CODEBASE QUALITY ASSESSMENT REFERENCE 128
Rating Criteria 129
4.3. FINDINGS CLASSIFICATION REFERENCE 130
Severity Level Reference 130
Status Level Reference 130
4.4. ABOUT OXORIO 132

2.1 DISCLAIMER

At the request of the client, Oxorio consents to the public release of this audit report. The information contained herein is provided "as is" without any representations or warranties of any kind. Oxorio disclaims all liability for any damages arising from or related to the use of this audit report. Oxorio retains copyright over the contents of this report.

This report is based on the scope of materials and documentation provided to Oxorio for the security audit as detailed in the Executive Summary and Audited Files sections. The findings presented in this report may not encompass all potential vulnerabilities. Oxorio delivers this report and its findings on an as-is basis, and any reliance on this report is undertaken at the user's sole risk. It is important to recognize that blockchain technology remains in a developmental stage and is subject to inherent risks and flaws.

This audit does not extend beyond the programming language of the codebase to include areas such as the compiler layer or other components that may introduce security risks. Consequently, this report should not be interpreted as an endorsement of any project or team, nor does it guarantee the security of the project under review.

THE CONTENT OF THIS REPORT, INCLUDING ITS ACCESS AND/OR USE, AS WELL AS ANY ASSOCIATED SERVICES OR MATERIALS, MUST NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE. Third parties should not rely on this report for making any decisions, including the purchase or sale of any product, service, or asset. Oxorio expressly disclaims any liability related to the report, its contents, and any associated services, including, but not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Oxorio does not warrant, endorse, or take responsibility for any product or service referenced or linked within this report.

For any decisions related to financial, legal, regulatory, or other professional advice, users are strongly encouraged to consult with qualified professionals.

2.2 PROJECT BRIEF

Title	Description
Client	UTEXO Protocol
Project name	UTEXO Bridge Enclave Signer
Category	TEE signer (AWS Nitro Enclave)
Website	https://utexo.com
Documentation	https://docs.utexo.com
Repository	enclave-signer
Initial Commit	`b6b2b256`
Final Commit	Integrated baseline: `dev@2df6d50` Audited remediation heads: PR #174 `091cc8ff`, PR #181 `b37d81a6`, PR #187 `d76efc0b`, PR #191 `a06e9565`
Platform	AWS Nitro Enclaves
Network	Ethereum / Bitcoin / RGB
Languages	Rust
Lead Auditor	Alexander Mazaletskiy - am@oxor.io
Project Manager	Ales Kuznetsova - ales@oxor.io

2.3 PROJECT TIMELINE

The key events and milestones of the project are outlined below.

Date	Event
May 25, 2026	The audit team initiated work on the project.
June 1, 2026	Submission of the preliminary audit report #1.
June 8, 2026	Submission of the preliminary audit report #2.
June 11, 2026	Submission of the preliminary audit report #3.
June 15, 2026	Submission of the preliminary audit report #4.
June 22, 2026	Submission of the audit report.
July 15, 2026	The team began the re-audit of the client's remediations.
July 21, 2026	Submission of the final audit report after the remediation re-audit.
August 11, 2026	Re-audit of the additional remediation PRs and verification of their repository integration status.
August 12, 2026	Submission of the final audit report.

2.4 AUDITED FILES

The following table contains a list of the audited files. The scc tool was used to count the number of lines and assess complexity of the files.

	File	Lines	Blanks	Comments	Code	Complexity
1	enclave/src/keys.rs	1385	177	98	**1110**	2%
2	enclave/src/validation/evm_crosscheck.rs	1028	80	235	**713**	6%
3	enclave/src/server.rs	909	84	122	**703**	6%
4	enclave/src/validation/spv_crosscheck.rs	822	74	150	**598**	3%
5	enclave/src/spv/chain.rs	722	80	163	**479**	7%
6	enclave/src/validation/rgb.rs	591	47	181	**363**	4%
7	enclave/src/signing/taproot.rs	548	53	43	**452**	6%
8	enclave/src/state.rs	457	48	79	**330**	4%
9	enclave/src/cloning.rs	360	44	71	**245**	2%
10	enclave/src/validation/psbt_crosscheck.rs	291	29	48	**214**	4%
11	enclave/src/signing/psbt.rs	287	24	27	**236**	4%
12	enclave/src/signing/evm.rs	276	37	17	**222**	1%
13	enclave/src/spv/validation.rs	238	25	63	**150**	5%
14	enclave/src/spv/merkle.rs	209	22	64	**123**	7%
15	enclave/src/attestation.rs	209	27	32	**150**	4%
16	enclave/src/spv/checkpoint.rs	203	17	77	**109**	5%
17	enclave/src/main.rs	181	16	20	**145**	10%
18	enclave/src/config.rs	127	15	29	**83**	5%
19	enclave/src/spv/types.rs	104	18	15	**71**	3%
20	enclave/src/vsock_forwarder.rs	95	10	9	**76**	13%
21	enclave/src/error.rs	88	23	1	**64**	3%
22	enclave/src/framing.rs	83	15	4	**64**	5%
23	enclave/src/spv/mod.rs	36	2	22	**12**	0%
24	enclave/src/lib.rs	23	3	0	**20**	0%
25	enclave/src/validation/mod.rs	6	0	0	**6**	0%
26	enclave/src/signing/mod.rs	3	0	0	**3**	0%
	Total	**9281**	**970**	**1570**	**6741**	**4%**

Lines: The total number of lines in each file. This provides a quick overview of the file size and its contents.

Blanks: The count of blank lines in the file.

Comments: This column shows the number of lines that are comments.

Code: The count of lines that actually contain executable code. This metric is essential for understanding how much of the file is dedicated to operational elements rather than comments or whitespace.

Complexity: This column shows the file complexity per line of code. It is calculated by dividing the file's total complexity (an approximation of cyclomatic complexity that estimates logical depth and decision points like loops and conditional branches) by the number of executable lines of code. A higher value suggests greater complexity per line, indicating areas with concentrated logic.

2.5 PROJECT OVERVIEW

The enclave signer runs inside an AWS Nitro Enclave and communicates with the untrusted parent host over a vsock channel. On startup it generates or restores a BIP-39 master seed (`keys.rs`, `KeyManager`) and derives the EVM, EVM-gas, BTC, and BIP-86 taproot keys; secrets are held in `SecretBox` / `Zeroizing` containers. `attestation.rs` produces and verifies Nitro Security Module attestations against expected PCR measurements, and `cloning.rs` implements the enclave-to-enclave handshake (X25519 ECDH, HKDF, AEAD) that replicates the seed across the cluster.

Signing requests are dispatched in `server.rs` and gated by a validation pipeline before any signature is produced. `validation/rgb.rs` validates RGB consignments, `spv/` verifies Bitcoin SPV inclusion and header-chain consistency, and `validation/evm_crosscheck.rs` / `validation/psbt_crosscheck.rs` cross-check EVM and PSBT requests against the validated event and the pinned bridge configuration. The actual signatures are produced in `signing/` (`evm.rs`, `psbt.rs`, `taproot.rs`). `vsock_forwarder.rs` proxies outbound connections (for example to Esplora) and `framing.rs` handles message framing on the wire.

2.6 CODEBASE QUALITY ASSESSMENT

The Codebase Quality Assessment table offers a comprehensive assessment of various code metrics, as evaluated by our team during the audit, to gauge the overall quality and maturity of the project's codebase. By evaluating factors such as complexity, documentation and testing coverage to best practices, this table highlights areas where the project excels and identifies potential improvement opportunities. Each metric receives an individual rating, offering a clear snapshot of the project's current state, guiding prioritization for refactoring efforts, and providing insights into its maintainability, security, and scalability. For a detailed description of the categories and ratings, see the Codebase Quality Assessment Reference section.

Category	Assessment	Result
Access Control	Signing endpoints are gated by validation. A unified, fail-closed production policy is now resolved and attested at boot (`C-01` fixed), and the gas-transaction path enforces canonical decoding with pinned chain/destination/zero-value plus gas/fee caps and a selector allowlist (`C-02` fixed). Cloning authenticity checks now precede nonce consumption and run the state transition atomically (`W-13` fixed). The residual trust model still relies on a static cluster-wide cloning secret (`M-08`, acknowledged). Related: `C-01`, `C-02`, `M-08`, `W-13`.	**Good**
Arithmetic	Cryptographic and amount arithmetic is generally sound. The `u256`→`u64` amount conversion limit and the non-strict amount binding remain accepted limitations rather than open defects (`I-14`, `I-06`, acknowledged). Related: `I-14`, `I-06`.	**Good**
Complexity	The validation flow remains intricate and concentrated in large files such as `evm_crosscheck.rs` and `keys.rs`, but the previously reported structural gaps were closed: malformed-ABI handling was hardened (`W-01` fixed), and the `fundsOut` calldata is validated field-by-field with a fail-closed selector guard and a single canonical typed `FundsOutParams` intent (`I-12` fixed). Related: `I-12`, `W-01`.	**Good**
Data Validation	PSBT and taproot anti-forgery checks are strong, and asset pinning and the previously ineffective typesystem check are now enforced (`C-03`, `W-09` fixed). Operation identifiers are enclave-authored with a transactional local replay reservation that rolls back on failure (`M-02` fixed); the consignment destination is not bound to the EVM recipient (`C-04`, acknowledged). Related: `C-03`, `C-04`, `M-02`, `W-09`.	**Good**

Category	Assessment	Result
Decentralization	A single master seed reconstructs every key and is replicated cluster-wide, so any single-enclave compromise has cluster-wide blast radius; this together with the rotation-desync and key-zeroization items remains an accepted residual risk (`I-13`, `M-08`, `M-03`, `I-07`, acknowledged). Related: `I-13`, `M-08`, `M-03`, `I-07`.	**Fair**
Documentation	The code is reasonably documented and a protocol specification exists. The operation-identifier remediation and the enclave-authored EVM-lock-record predicate are in place (`M-02`, `W-05` fixed); the non-strict amount binding is documented and accepted (`I-06`, acknowledged). Related: `W-05`, `M-02`, `I-06`.	**Good**
External Dependencies	The enclave depends on Esplora, the AWS Nitro Security Module, and RGB/Bitcoin libraries, and trusts the host only as a transport. Consignment anchoring now requires SPV and fails closed when it is unavailable (`M-01` fixed). The typed, destination-pinned Esplora client removes the generic local TCP egress (`I-08` fixed), and the Esplora call on the signing path is timeout- and retry-bounded (`I-03` fixed). Related: `M-01`, `I-03`, `I-08`.	**Good**
Error Handling	The dispatch and validation paths fail closed and use explicit, typed errors throughout; the `dev-mode` short-calldata panic path was removed (`I-02` fixed). Related: `I-02`.	**Good**
Logging and Monitoring	In-enclave observability is intentionally limited; there is little durable audit trail of signing decisions. Related: no findings map to this metric.	**Good**
Low-Level Calls	Cryptographic primitives rely on vetted crates with constant-time comparisons and small-order point rejection; unsafe code is minimal. Related: no findings map to this metric.	**Good**
Testing and Verification	Unit tests cover signing and validation, and the count-capped replay-guard DoS was fixed (`W-12` fixed). The `policy.rs` boot gate is unit-tested across build/config combinations, and PR #177 adds the failed-completion retry regression test for cloning (`W-13` fixed). Related: `W-12`, `W-13`.	**Good**

2.7 FINDINGS BREAKDOWN BY FILE

This table provides an overview of the findings across the audited files, categorized by severity level. It serves as a useful tool for identifying areas that may require attention, helping to prioritize remediation efforts, and provides a clear summary of the audit results.

File	TOTAL	CRITICAL	MAJOR	WARNING	INFO
enclave/src/server.rs	**15**	3	1	6	5
enclave/src/validation/evm_crosscheck.rs	**14**	3	0	4	7
enclave/src/validation/rgb.rs	**6**	1	1	2	2
enclave/src/keys.rs	**4**	1	0	1	2
enclave/src/vsock_forwarder.rs	**3**	0	1	0	2
proto/enclave.proto	**3**	1	0	2	0
attestation-verify/src/lib.rs	**2**	0	0	1	1
enclave/Cargo.toml	**2**	1	1	0	0
enclave/src/attestation.rs	**2**	1	0	1	0
enclave/src/main.rs	**2**	1	0	0	1
enclave/src/signing/evm.rs	**2**	0	0	1	1
enclave/src/spv/chain.rs	**2**	0	0	2	0
enclave/src/validation/psbt_crosscheck.rs	**2**	1	0	1	0
ethereum/src/MultisigProxy.sol	**2**	0	0	2	0
build/Dockerfile.enclave	**1**	0	0	0	1
build/build-enclave.sh	**1**	0	0	0	1
enclave/src/config.rs	**1**	1	0	0	0
enclave/src/framing.rs	**1**	0	0	0	1
enclave/src/spv/checkpoint.rs	**1**	0	0	1	0
enclave/src/spv	**1**	0	0	1	0
enclave/src/state.rs	**1**	0	0	1	0
enclave/src/validation/spv_crosscheck.rs	**1**	0	0	0	1
ethereum/src/Bridge.sol	**1**	0	0	1	0
ethereum/src/settlement/RgbSettlementModule.sol	**1**	0	0	1	0

2.8 CONCLUSION

A comprehensive audit was conducted on 26 source files, initially revealing 4 critical and 9 major issues, along with numerous warnings and informational notes. The audit highlighted that the enclave's cryptographic primitives are sound, but the authorization and exposure model around them is weaker than the zero-trust design intends: cloning authorization rests on a static shared secret that reproducible PCRs cannot distinguish, the master seed is held for the whole lifetime and replicated cluster-wide, and several signing paths lack operation-uniqueness or bind insufficiently to the validated event. Further findings concern request-validation completeness and configuration handling in the EVM cross-check and RGB consignment validation.

Following our initial audit, UTEXO Protocol worked closely with our team to address the identified issues. The proposed changes focused on binding cloning and signing authorization to an attested operator identity, minimizing seed residency and shared blast radius, enforcing operation-uniqueness with a durable cluster-wide anchor, and tightening configuration and validation so that inadequate inputs cannot pass. Through multiple rounds of interaction, all identified issues have been successfully addressed or formally acknowledged.

As a result, the project has passed our audit. Our auditors have verified that the UTEXO Bridge Enclave Signer, considering `dev@2df6d50` together with the audited remediation heads listed in the Project Brief, operates as intended within the defined scope, based on the information and code provided at the time of evaluation. Of the 43 findings, 31 are verified fixed and 12 are acknowledged as accepted residual risks; none remain open. The robustness of the codebase has been significantly improved, meeting the necessary security and functionality requirements established for this audit.

Moreover, we advise increasing the test coverage of the codebase, particularly adversarial, replay, and resource-exhaustion scenarios. Comprehensive testing is essential to uncover edge cases and ensure that the enclave performs as expected under various conditions.

3 FINDINGS REPORT

OXORIO

3.1 CRITICAL

C-01	The enclave infers security mode from empty fields and feature flags instead of enforcing an explicit production policy in `config.rs`, `main.rs`, `evm_crosscheck.rs`, `psbt_crosscheck.rs`, `server.rs`, `attestation.rs`, `Cargo.toml`
Severity	**CRITICAL**
Status	• FIXED

This is a systemic finding. The individually tracked items (unconfigured bridge config, the empty-`evm_tx_hash` mode selection, the empty-`rgb_asset_id` asset bypass, and the dev-feature release guards) are individual manifestations of it. The common root is that production-vs-legacy trust is selected implicitly by empty config, empty request fields, or build features.

Location

File	Location	Line
config.rs	`BridgeConfig::from_env`, `BridgeConfig::is_configured`	36-62
main.rs	unconfigured bridge config warning	47-53
evm_crosscheck.rs	legacy request asset path	206-224
psbt_crosscheck.rs	empty `evm_tx_hash` selects vanilla mode	41-45
server.rs	EVM/PSBT cross-checks skipped under `dev-mode`	387-496
attestation.rs	`mock-attestation` raw CBOR / zero PCR path	50-101
Cargo.toml	dangerous testing features	84-101

Description

In these files, the enclave has no single explicit `Production` security policy that is enforced before signing. Instead, multiple security-critical behaviours are selected by the absence of data or by build features:

```
// config.rs
pub fn is_configured(&self) -> bool {
    self.chain_id != 0 || self.bridge_contract != [0u8; 20] || !self.rgb_asset_id.is_empty
}
```

The developer comment states the intended fallback:

```
// when no operator config is present we fall back
// to the legacy "trust the request" behaviour
```

The same pattern appears at request level. In `evm_crosscheck.rs`, an empty `req.rgb_asset_id` is treated as a legacy path; in `psbt_crosscheck.rs`, an empty `evm_tx_hash` selects vanilla mode and skips bridge checks; in `server.rs`, `dev-mode` skips enriched signing validation; in `attestation.rs`, `mock-attestation` uses all-zero PCRs and no COSE wrapping.

These are different security states, but the code does not model them as a single attested policy. A verifier can see some missing configuration in the attestation bundle, but the enclave still accepts signing requests in the corresponding legacy mode. Similarly, a request with an omitted field is indistinguishable from a request where a compromised listener deliberately omitted the field to avoid a cross-check.

This finding amplifies several already reported issues:

- No operator config means no chain/contract/asset pin, yet signing is not refused.
- An empty `rgb_asset_id` can bypass asset identity binding.
- An empty `evm_tx_hash` can bypass PSBT bridge-mode authorization (see the vanilla-mode `SignPsbt` bypass finding).
- `dev-mode`, `mock-attestation`, and `allow-seed-import` remain build-time trust collapses unless release builds make them impossible.

Under the report threat model, the listener/host is precisely the actor allowed to choose request fields and influence deployment/configuration mistakes. Therefore, security mode must not be inferred from attacker-controlled emptiness or operator discipline alone.

Proof of Concept

1. Start from the pinned source at `enclave-signer dev@b6b2b256`.
2. Observe that `BridgeConfig::from_env` converts missing or invalid env values to zero/empty values.
3. Observe that `BridgeConfig::is_configured` is an OR over the fields, not a fully-configured production check.

4. In `validate_evm_request`, when `bridge_config.is_configured` is false, the pinning block is skipped and listener-supplied chain/contract/asset values are not compared to an operator pin.
5. In configured mode, set `req.rgb_asset_id = ""`; the asset mismatch check is skipped because it is gated on `!req.rgb_asset_id.is_empty`.
6. In `SignPsbt`, set `evm_tx_hash = []`; `validate_psbt_request` returns from vanilla mode before bridge checks.
7. Build with `dev-mode` or `mock-attestation`; comments and feature definitions show that signing validation or the attestation root of trust is removed.

Expected result: production bridge signing should be selected by an explicit, attested, fail-closed policy. Actual result: multiple empty-field and feature-flag paths silently downgrade validation.

Recommendation

We recommend introducing one explicit attested signing policy, for example:

```
enum SigningPolicy {
    Production {
        chain_id: u64,
        bridge_contract: [u8; 20],
        rgb_asset_id: String,
        allow_vanilla_psbt: bool,
    },
    Development,
}
```

For production:

1. Require a fully configured `(chain_id, bridge_contract, rgb_asset_id)` before the enclave becomes `Active` or before any bridge signing endpoint is reachable.
2. Commit the policy, including whether vanilla PSBT signing is enabled, into attestation `user_data`.
3. Reject empty request fields for values that are supposed to be bound. Empty should mean invalid, not legacy.
4. Compare validated facts directly to pinned facts (`validated.contract_id == policy.rgb_asset_id`) rather than to optional listener-declared fields.
5. Add a `compile_error!` release guard for each dangerous development feature — `dev-mode` (EVM/PSBT cross-check and amount-binding skipped), `mock-attestation` (zero-PCR attestation docs accepted), and `allow-seed-import` (parent installs a chosen seed) — for example `#[cfg(all(feature = "<name>", not(debug_assertions)))]`, so none of them

can be compiled into a release build, and add a CI job that builds `--release` with each feature individually to assert the guard fires.

Update

Client's response

Fixed (different approach) — no single `SigningPolicy` enum; the production posture is compile-time and attested instead: a plain build selects the production signing path (SPV + RGB validation, with the shipped enclave image additionally enabling `vsock` and `evm-rpc`), and no dev feature is on by default (PR #125); each dev feature trips a `compile_error!` in release builds with CI asserting it (PR #75), an unconfigured bridge config refuses to sign (PRs #75, #98), and the pinned chain/contract/asset values are committed into the attestation `user_data`. The individual implicit mode-selections are closed under C-03 (PR #75) and M-05 (PR #102).

Partially fixed at: `bbed82d3`, `4e38f205`, `2a9566d5`, `9248ed70`

Oxorio's response

Remaining issue: The enclave still has no single fail-closed production policy object. The effective security posture is reconstructed at runtime from several independent conditions — build features, the presence or absence of `BridgeConfig` fields, request shape, and the optional chain-verification components — and the shipped image (which enables raw `evm-rpc`) can run without an explicit, attested policy that a verifier can check as a single value. The individual improvements (release guards, all-pins-required config) close specific manifestations but do not establish the unified invariant.

Required remediation: Introduce one explicit `Production` policy object with fail-closed defaults that carries the signing modes, the chain/contract/asset pins, the expected attestation values, and the allowed data sources; commit it into attestation `user_data`; and add release tests that reject every legacy or unsafe build/deployment combination before the enclave becomes active.

Reproduction: Build the enclave from the pinned source with `evm-rpc` enabled and a partially-populated environment, then deploy it. The external attestation still verifies that the correct image is running (matching PCRs), but nothing in the attested `user_data` proves a complete, fail-closed production policy is in force. The enclave keeps accepting signing requests whose validation strength depends on which build features and config fields happen to be present, so two deployments of the same image can enforce materially different checks while attesting identically.

Client's response

Closes the systemic half of audit finding C-01 (issue #154): the enclave now has one explicit security policy, resolved once at boot, that a verifier checks as a single attested value —

instead of reconstructing the posture at runtime from build features, empty BridgeConfig fields, request shape, and which chain-verification components happened to be wired up.

Fixed at: `2ec5bbb`, `f1be252`, `dfc7401`

C-02	`SignRawDigest` signs arbitrary gas-key digests without a transaction-shape allowlist in `server.rs`, `proto/enclave.proto`, `keys.rs`
Severity	**CRITICAL**
Status	• FIXED

> This finding promotes a previously tracked hardening item to a standalone finding because it violates the invariant that every enclave signing endpoint must bind the signature to an enclave-validated intent.

Location

File	Location	Line
server.rs	`handle_sign_raw_digest` signs any 32-byte digest with the gas key	544-569
proto/enclave.proto	`SignRawDigestRequest` carries only a raw 32-byte digest	291-293
keys.rs	`KeyManager::sign_evm_gas_tx` signs the digest with the gas TX key	318-330

Description

In the file `server.rs`, the raw-digest endpoint accepts only one predicate: the digest must be exactly 32 bytes. It then signs that digest with the dedicated EVM gas transaction key:

```
if req.digest.len != 32 {
    return Err(EnclaveError::InvalidRequest(..));
}

let digest: [u8; 32] = req.digest.as_slice.try_into.unwrap;
let signature = state.sign_evm_gas_tx(&digest)?;
```

The request does not carry a typed transaction, chain id, target address, nonce, gas limits, calldata allowlist, or value limit. Therefore the enclave cannot distinguish a legitimate gas transaction digest from an attacker-chosen digest prepared by a compromised listener.

If a compromised listener can reach `SignRawDigest`, it can obtain gas-key signatures over arbitrary Ethereum transaction digests. The immediate impact is draining or misusing the dedicated gas-address funds and, depending on the account's permissions, possibly executing unexpected contract calls from that gas address.

Proof of Concept

1. Prepare any Ethereum transaction controlled by the gas-key account, for example a value transfer or contract call that is outside the expected operational gas-payment flow.
2. Compute its pre-signing transaction digest outside the enclave.
3. Submit that 32-byte digest in `SignRawDigestRequest.digest`.
4. `handle_sign_raw_digest` checks only `digest.len == 32` and calls `state.sign_evm_gas_tx(&digest)`.
5. The enclave returns a valid ECDSA signature for the attacker-chosen transaction digest.

Expected result: the gas-key endpoint should sign only typed, enclave-validated gas transactions that match a strict allowlist. Actual result: any 32-byte digest is signable.

Recommendation

We recommend replacing `SignRawDigest` with a typed gas-transaction endpoint where the enclave receives the transaction itself, not a pre-computed digest, and recomputes the signing hash internally so it can never be tricked into signing a digest over bytes it never inspected. Concretely:

- **What to validate (and derive the digest from).** Accept the RLP-encoded EIP-1559/ legacy transaction, decode it into a typed struct, and validate every field: `chain_id`, `nonce`, destination address, `value`, `gas_limit`, `max_fee` / `max_priority_fee` (or `gas_price`), and the calldata selector. Recompute the signing digest from this typed transaction in-enclave (`keccak256(rlp(tx))`) and sign that — replacing the current "trust the caller's 32-byte digest" path in `keys.rs:318-330`.
- **Where the source of truth comes from.** Pin `chain_id` to `BridgeConfig` (the same attested value used elsewhere) and fail closed on mismatch. Constrain the destination to an allowlist — the bridge / known operational contracts — rather than any address. Bound `value` by an explicit per-transaction cap (a pure gas-payment tx normally has `value == 0`), and restrict calldata to an allowlisted set of selectors the gas account is expected to call. Anything outside these bounds fails closed.
- **Dev exception.** If a raw-digest path must remain for development, gate it behind the same attested non-production `SigningPolicy` introduced in the production-policy finding and make it impossible in release production builds via `compile_error!` / CI guards.

Update

Client's response

Fixed (different approach) — `SignRawDigest` now requires the unsigned gas-tx preimage, RLP-decodes it in-enclave, pins chain-id and destination (`GAS_TX_ALLOWED_TO`, value must be 0, fail-closed when unset) and recomputes the digest itself (PR #81). Instead of a calldata

allowlist, `GAS_TX_ALLOWED_TO` is required by operator configuration to be an EOA — which makes any calldata inert. The enclave enforces the destination pin and `value == 0`; it cannot verify EOA-ness itself (it holds no chain state), so that requirement is documented for the operator.

Partially fixed at: `5e5ed8a1`

Oxorio's response

Remaining issue: The gas-key signing policy is only partially inside the enclave trust boundary. The enclave now derives the digest from a decoded transaction and pins the destination and `value == 0`, but it does not cap the gas limit or fees, does not restrict the calldata to an allowlist, does not itself confirm that the destination is a codeless EOA, and does not commit the destination rule into attestation. The "any calldata is inert" argument rests on an unverified operator assumption (EOA-ness), which the enclave cannot check.

Required remediation: Enforce in-enclave upper bounds on gas and fees, an allowlist of permitted calldata, and a chain/destination rule committed into attestation `user_data`; where safety depends on the destination being a plain wallet, verify the destination's authenticated code state rather than trusting configuration. Add rejection tests for each forbidden transaction shape.

Reproduction: A compromised listener submits an unsigned gas transaction whose destination matches `GAS_TX_ALLOWED_TO` and whose `value == 0`, but with an inflated gas limit / fee (or, if `GAS_TX_ALLOWED_TO` is in fact a contract because the EOA assumption was never enforced, with non-empty calldata). The enclave decodes it, sees the pinned destination and zero value, and signs — allowing the gas account to be drained through fees or driven into an unintended contract call.

Client's response

Fixed as proposed. Gates the gas-key SignRawDigest path with operator-pinned, attested bounds (canonical preimage recomputation, chain/destination pins, zero value, gas/fee caps, and a selector allowlist) so a compromised listener can no longer obtain arbitrary gas-key signatures.

Fixed at: `5e5ed8a`, `091cc8f`

C-03	An empty `rgb_asset_id` bypasses the consignment-to-pinned-asset binding, allowing a foreign-asset burn to authorize a release in `server.rs`, `evm_crosscheck.rs`
Severity	**CRITICAL**
Status	• FIXED

Location

File	Location	Line
server.rs	empty `req.rgb_asset_id` short-circuits the in-enclave `contract_id` check	371-375
evm_crosscheck.rs	`req.rgb_asset_id` is not compared against the pinned asset when empty	219-224

Description

In the file `server.rs`, an empty `req.rgb_asset_id` short-circuits **both** the listener-pin match **and** the in-enclave `contract_id` check. With the asset identity unpinned, a valid burn consignment for a *foreign* RGB asset can be presented to authorize a USDT0 unlock — a listener-triggerable funds-theft path, since the release is no longer tied to the asset the bridge actually custodies.

Recommendation

We recommend requiring the enclave to compare `validated.contract_id` against the pinned `BridgeConfig::rgb_asset_id` and to reject when either side is absent, so an unpinned or mismatched asset can never authorize a release.

Update

Client's response

Fixed as recommended — the validated consignment's `contract_id` is bound to the pinned `RGB_ASSET_ID`, failing closed when either side is absent (PR #75).

Fixed at: `4e38f205`

C-04	The EVM `recipient` in `fundsOut` calldata is not bound to the RGB consignment's destination, allowing release redirection in `evm_crosscheck.rs`, `rgb.rs`
Severity	**CRITICAL**
Status	• ACKNOWLEDGED

Location

File	Location	Line
evm_crosscheck.rs	the `recipient` field in `fundsOut` calldata is signed without a destination cross-check	—
rgb.rs	the burn transition's destination-binding metadata (EVM contract + recipient) is not read	—

Description

In the file `evm_crosscheck.rs`, the EVM recipient carried in the `fundsOut` calldata is neither derived from nor bound to any RGB-side authorized destination. The consignment does not commit to the EVM recipient at all: at the audited commit the IFA schema the enclave understands defines a single burn-transition metadata key, `MS_BURNED_ASSET` (the destroyed amount), and the transition's "destination seals" are *Bitcoin* seals (`txid:vout` or a `utxob:...` commitment), not EVM addresses. As a result `ValidatedConsignment` / `TransitionSummary` (`rgb.rs`) expose the burned amount and Bitcoin seals but no EVM address whatsoever. The EVM `recipient` is therefore a free, unbound calldata field: a compromised backend can present a perfectly valid burn consignment and direct the unlocked funds to any address it chooses, because nothing the enclave validated says where the funds were supposed to go on EVM.

This is not a case of the enclave failing to read an existing binding — the binding does not exist in the committed RGB state. Making the recipient (and, symmetrically, the source) verifiable requires first adding an EVM destination commitment to the schema: a new burn-transition metadata key (EVM contract + recipient) that the burn initiator populates so `rgbstd` validation makes it part of the signed state, which the enclave can then parse the same way it special-cases `MS_BURNED_ASSET`.

Recommendation

We recommend introducing an EVM destination commitment into the RGB burn flow and then enforcing it in the enclave. Concretely: add a burn-transition metadata key (EVM

contract address + recipient, optionally the destination chain id) that the burn initiator must populate, so `rgbstd` validation binds it into the signed consignment state; have the enclave extract that committed destination from the validated transition and cross-check it against the `recipient` in the `fundsOut` calldata being signed, rejecting on mismatch, so the release destination is always the one the RGB side authorized. Apply the same pattern to an authorized source/sender address where the flow needs it (commit it on the originating side, decode it from the validated transition metadata, and assert equality with the calldata `sourceAddress`) so neither end of the transfer can be silently redirected. Until such a commitment exists in the schema, the EVM recipient cannot be validated against any enclave-trusted fact and should be documented as an explicitly unbound field. This is the same lever as the canonical-bridge-intent recommendation in the INFO finding on request/calldata-supplied cross-chain metadata, and the two should be implemented together.

Update

Client's response

Acknowledged (valid — fix scheduled before production) — the recipient cross-check requires destination metadata in the consignment, which is introduced by the new RGB schema designed specifically for the bridge: BFA (Bridged Fungible Asset). The finding is valid; the binding will be implemented before production, once BFA is fully implemented in the RGB protocol. Details tracked in #66.

Oxorio's response

The risk remains open. A valid RGB burn consignment still carries no authenticated EVM recipient, so a compromised backend can present a legitimate burn and direct the unlocked funds to any address of its choosing; the enclave has no RGB-derived value to compare the `fundsOut` recipient against. The client accepts the finding and defers it to the new bridge-specific RGB schema (BFA, tracked in issue #66), which will commit the EVM destination into the signed consignment state. There is no in-enclave compensating control today: the only mitigations are operational (trusting the backend not to redirect releases), which is precisely the trust the enclave is meant to remove. Before production, the enclave must extract the RGB-committed destination from the validated transition and assert it equals the `fundsOut` recipient, failing closed on mismatch; until then a bridge deployment relying on this path is not zero-trust for release redirection.

M-01	`rgb-validation` without `spv`: consignment anchoring trusts only the host-controlled Esplora in `enclave/Cargo.toml`
Severity	**MAJOR**
Status	• FIXED

MAJOR, conditionally CRITICAL: if a `rgb-validation`-without-`spv` build ever reaches production it signs a `fundsOut` release against a non-existent on-chain anchor; rated MAJOR rather than CRITICAL only because the production `build/Dockerfile.enclave` builds `--features vsock,rgb-validation,spv`.

Location

File	Location	Line
Cargo.toml	feature graph (`spv = ["rgb-validation"]`, but `rgb-validation` does not pull `spv`; `default = []`)	92, 98
server.rs	`handle_sign_evm` (the `#[cfg(feature = "spv")]` anchoring block)	342-458
rgb.rs	Esplora resolver (reached through the untrusted host)	251-279
vsock_forwarder.rs	enclave TCP → vsock → host → real Esplora	—

Description

In the file `enclave/Cargo.toml` the feature graph is one-directional: `spv = ["rgb-validation"]`, but `rgb-validation = ["rgb-ops","rgb-consignment"]` does **not** pull `spv`, and `default = []`. In the file `enclave/src/server.rs`, RGB validation (`transfer.validate(&resolver, &config)`) asks the Esplora resolver whether each witness tx is mined and at what height; that resolver reaches Esplora **through the untrusted host** (enclave TCP → vsock → host → real Esplora, see `enclave/src/vsock_forwarder.rs`). The only host-independent anchoring check is the `#[cfg(feature = "spv")]` block in `handle_sign_evm`, which verifies a merkle proof for every witness txid against the enclave's own PoW-verified header chain at depth ≥ 6.

```
## enclave/Cargo.toml
rgb-validation = ["rgb-ops", "rgb-consignment"] # does NOT pull spv
spv = ["rgb-validation"] # one-directional
```

```
// enclave/src/server.rs — empty proofs slip through a no-spv build

#[cfg(not(feature = "spv"))]
if !req.merkle_proofs.is_empty() {
    return Err(EnclaveError::Spv(
        "enclave was not built with --features spv but request carries \
         merkle_proofs; refusing to sign without verification (rebuild \
         with `--features spv,rgb-validation` to enable SPV)"
            .into(),
    ));
}
```

Reproduction:

1. The enclave is built with `--features rgb-validation` (or `--no-default-features --features rgb-validation`, which stays a valid build even with the default feature set disabled) — RGB validation runs **without** SPV.
2. A malicious host runs an Esplora MITM and answers the resolver that an arbitrary witness tx is "mined" at a sufficient height.
3. `transfer.validate` succeeds with no real Bitcoin anchor; `handle_sign_evm` reaches the signing step and the enclave signs a `fundsOut` release.
4. The existing fail-closed guard at `server.rs` does not save this case: it only rejects a no-spv build when the request *carries* `merkle_proofs`, so a no-spv build with an **empty** `merkle_proofs` passes.

The result is an authorized release backed by a fabricated anchor.

Recommendation

We recommend adding a `compile_error!` when `rgb-validation` is set without `spv` (near the top of `enclave/src/lib.rs`) or making `rgb-validation` pull `spv` in `Cargo.toml`; as a runtime backstop, widen the guard at `server.rs` so a no-`spv` build refuses to sign *any* `fundsOut`, not only requests that carry `merkle_proofs`.

Update

Client's response

Fixed as recommended — `spv` is in the default feature set, `rgb-validation`-without-`spv` is a `compile_error!`, and a no-spv build refuses to sign any fundsOut (PR #125). CI lanes asserting the guard landed in PR #144.

Fixed at: `bbed82d3`, `849cff59`

M-02	No binding or uniqueness of the RGB ↔ EVM operation identifiers, enabling double-spend in both bridge directions in `evm_crosscheck.rs`, `psbt_crosscheck.rs`, `server.rs`, `proto/enclave.proto`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
enclave.proto	`SignEvmRequest` carries no `op_id` field on the wire	—
evm_crosscheck.rs	`SignEvm` validates the burn amount and SPV inclusion but never extracts or binds the RGB `op_id` / `burnId`	—
psbt_crosscheck.rs	`SignPsbt` bridge mode signs without recording the consumed `(evm_tx_hash, operation_idx)`	—
server.rs	`burnId` is folded into the signed digest but is never bound to the validated consignment	—

Description

Three distinct identifiers are involved and must not be conflated. The **RGB** `OpId` is the identifier of an RGB state transition (burn/mint) inside the consignment — the spec's intended cross-domain primitive, extracted by the enclave as `all_op_ids` / `TransitionInfo.op_id` (`rgb.rs`). The **EVM** `operationId` is a separate `uint256` supplied by the caller at deposit time (`Bridge.fundsIn(..., operationId, ...)`) and used as the on-chain lock-record key (`fundsInRecords[operationId]`); at deposit time there is no RGB operation yet, so it is a backend-assigned EVM-domain identifier, not an RGB `OpId`. The `burnId` is a caller-supplied `uint256` (carried in the `fundsOut` calldata and merely signed by the enclave) that keys the release-side replay guard (`consumedBurnIds[burnId]`). The spec's design is for one RGB `OpId` to be threaded across all of these, but in the code they are independent fields that are never bound together.

In the files `evm_crosscheck.rs`, `psbt_crosscheck.rs` and `server.rs`, the enclave enforces neither cross-domain identifier binding nor operation-uniqueness in either bridge direction. The RGB `OpId` — the spec's core cross-domain primitive — is absent from `SignEvmRequest` entirely (the request carries `call_data`, `rgb_amount`, `consignment`, …, but no `op_id`): it is never checked against the consignment's transition history and never

bound into the signed calldata or EIP-712 typed data. The only in-memory replay structure is the attestation replay guard, which protects the cloning handshake, not bridge operations.

Two concrete double-spend paths follow:

- **EVM→RGB (`SignPsbt` bridge mode).** The enclave does not record that a finalized deposit reference `(evm_tx_hash, operation_idx)` was already consumed, and there is no on-chain guard on this side. The same deposit reference can therefore be presented repeatedly, each submission producing a fresh signature that spends bridge UTXOs — one deposit, multiple Bitcoin/RGB releases.
- **RGB→EVM (`SignEvm fundsOut`).** Replay protection is delegated entirely to the on-chain `consumedBurnIds[burnId]`. The enclave validates the burn amount and SPV inclusion of the consignment's witness transactions but never extracts `burnId` or binds it to the specific consignment; `burnId` is only folded into the signed digest. The on-chain guard rejects a repeated *same* `burnId`, but not the same underlying RGB burn presented under a *different* `burnId`. A listener can therefore pair one valid `(consignment, SPV proof)` with distinct `burnId` values and obtain multiple valid `fundsOut` signatures, releasing EVM funds more than once for a single RGB burn.

The common root cause is that operation uniqueness and identifier binding are enforced nowhere inside the enclave: they are delegated to the on-chain contract (only for one direction, and only at `burnId` granularity) and to a correct listener. Under the project's zero-trust model neither is a sufficient anchor on its own.

Recommendation

We recommend enforcing operation uniqueness with a durable, cluster-wide anchor and binding the operation identifier to the validated event. Concretely: add an `op_id` field to `SignEvmRequest` / `EnrichedEvmPayload`, cross-check it against the consignment's extracted operation identifiers, and bind it into the signed calldata. For EVM→RGB, reject a request whose deposit reference — chain id, bridge contract, EVM tx hash, operation index, RGB asset id — was already consumed, anchored in durable state (an on-chain marker mirroring `consumedBurnIds`, or a quorum-replicated, rollback-protected store updated under the same critical section as signing). For RGB→EVM, derive and bind `burnId` to the specific validated consignment and SPV-proven witness transaction so the same burn cannot be re-released under a different `burnId`. Treat the listener and any single enclave as untrusted: the uniqueness anchor must be durable and shared across the quorum.

Update

Client's response

Fixed (different approach) — the enclave authors the identifiers instead of comparing them: `burnId` = keccak256 of the validated OpId, `fundsInIds` rewritten from the validated mint transitions, overwritten before signing (PRs #97, #101); in-memory dedup as defence-

in-depth (PR #100). Durable replay uniqueness is anchored on-chain by design (`consumedBurnIds`), so no enclave-side persistent set is needed.

Partially fixed at: `e8aea565`, `3147209d`, `7b675b7a`

Oxorio's response

Remaining issue: Operation identity is still not derived from a single canonical, enclave-validated representation, and there is no durable, cluster-wide exactly-once anchor. Binding `burnId` to the validated OpId is an improvement, but mint-side identifiers come from a parallel flat parser, and the only uniqueness state added by these PRs is an in-memory, per-instance, TTL-bounded dedup that is partly keyed on host-controlled fields. Because the replay key is recorded before signing, a transient failure can consume a key without any signature ever being produced.

Required remediation: Derive one canonical source-operation identifier from the authentic RGB/EVM data and consume it atomically in durable, shared state (an on-chain marker or a quorum-replicated, rollback-protected store) only when an authorized transition is committed; the local reservation must roll back on failure. On-chain `consumedBurnIds` alone covers only one direction and only at `burnId` granularity, so it cannot close EVM→RGB uniqueness or the same-burn-under-different-`burnId` replay.

Reproduction: A correct request passes validation and reserves the in-memory replay key; after an artificially induced transient signing error, retrying the same legitimate operation before the TTL expires is rejected as a replay even though no signature was produced. Conversely, the same volatile dedup is bypassed by an enclave restart, a sibling enclave, TTL expiry, or cache eviction, so one deposit or burn can be signed more than once across the cluster.

Client's response

Fixed — The transient-failure replay issue has been addressed in PR #187.

The replay key is now reserved before signing but committed only after signing succeeds. If signing fails, the reservation is rolled back, allowing the same legitimate operation to be retried. The replay key is also now derived using the enclave-validated funds_in_operation_id rather than the host-controlled operation_idx.

Architectural clarification regarding durable shared replay state — We do not consider persistent, cluster-wide replay storage inside the TEE to be a requirement of this architecture. The enclave is intentionally stateless and does not own protocol-level persistence. Durable replay protection is enforced by the authoritative state layer of each bridge direction:

RGB → EVM: replay protection is enforced by the EVM smart contract through consumedBurnIds. Once a burnId has been consumed, it cannot be used to unlock funds again.

EVM → RGB: replay protection is enforced by RGB's stateful validation and rgb-lib. RGB recipient IDs are single-use, and rgb-lib explicitly rejects reuse with RecipientIDAlreadyUsed: `rgb-lib` `RecipientIDAlreadyUsed`

The enclave's in-memory replay guard is therefore intentionally defence-in-depth, not the authoritative durable uniqueness store. An enclave restart or a sibling enclave may lose the local replay cache, but this does not by itself allow the same operation to produce a second valid protocol-level state transition.

For this reason, introducing persistent or quorum-replicated storage into the TEE would conflict with the stateless enclave design and duplicate replay protection that belongs to the underlying EVM/RGB state machines.

Fixed at: `3147209`, `ce10812`, `d76efc0`

M-03	Signer-set rotation desync: the BTC/RGB multisig and the EVM signer set rotate independently in `MultisigProxy.sol`
Severity	**MAJOR**
Status	• ACKNOWLEDGED

> MAJOR (security + liveness): an enclave removed from the EVM signer set — including a *compromised* one — retains Bitcoin/mint signing power until the BTC descriptor is rotated and all UTXOs migrated.

Location

File	Location	Line
MultisigProxy.sol	`_enclaveSigners` / `enclaveThreshold` have no reference to any BTC signer set	24-25
MultisigProxy.sol	`proposeUpdateEnclaveSigners` rotates only the EVM enclave signer set	355-372
MultisigProxy.sol	constructor rejects only too-large `timelockDuration_`; no minimum floor	167
MultisigProxy.sol	`proposeSetTimelockDuration` queues arbitrary new duration below the maximum	458-473
MultisigProxy.sol	`SetTimelockDuration` execution enforces only the upper bound	828-831
keys.rs	BTC colored derivation path `m/86'/827167'/0'`	179-180
keys.rs	`sign_psbt` signs matching inputs without checking EVM-set membership	335-397

Description

The signer set lives in two unlinked registries. In the contract `MultisigProxy` the EVM `_enclaveSigners` / `enclaveThreshold` (rotated via the timelocked `proposeUpdateEnclaveSigners`) reference no BTC set. In the file `enclave/src/keys.rs` the BTC colored key `m/86'/827167'/0'` signs any input matching its own key in `sign_psbt`, regardless of EVM-set membership — there is no cross-layer revocation. The Bitcoin multisig script (the cosigner pubkeys baked into the address holding the federation UTXOs) is rotated only by building a new address and migrating all UTXOs, a manual off-chain operation. Rotating one layer does not touch the other.

```
// enclave/src/keys.rs — sign_psbt: no cross-layer revocation
// signs any input whose key matches m/86'/827167'/0' regardless of EVM-set membership
```

```
// MultisigProxy.sol (ctor) and (SetTimelockDuration)
if (timelockDuration_ >= MAX_PROPOSAL_LIFETIME) revert TimelockTooLong; // upper bound only
— 0 is accepted
```

Reproduction (security case):

1. An enclave key is compromised and removed from `_enclaveSigners` on EVM via `proposeUpdateEnclaveSigners`.
2. The BTC descriptor is not rotated (or its UTXOs are not yet migrated).
3. The removed/compromised key still matches inputs in `sign_psbt`, so it keeps participating in mint/anchoring signing on Bitcoin.

The liveness mirror: rotating EVM but not BTC can make the BTC threshold unreachable (mint/funds locked on Bitcoin); rotating BTC but not EVM is the reverse. `enclaveThreshold` (EVM) and M (the BTC script) are also set independently.

The EVM rotation path additionally has no minimum timelock floor: `MultisigProxy` enforces only an upper bound (`timelockDuration_ >= MAX_PROPOSAL_LIFETIME` reverts), so the federation may set `timelockDuration = 0` and rotate `_enclaveSigners` in a single block. The spec requires: "any update to the trusted TEE signer set MUST go through the timelocked admin path (e.g. 2 weeks)" — relying on an observation window that a zero-timelock rotation removes entirely, making a hostile EVM rotation both instant *and* unmirrored on BTC.

Recommendation

We recommend binding BTC/RGB signing authority to the same attested signer set inside the enclave, so a key removed from the federation can no longer sign: have `keys.rs::sign_psbt` check the current authorized signing set (delivered as attested policy/config) before signing an input, instead of signing any input that merely matches its own derived key with no cross-layer revocation. Treat the on-chain changes as companion steps, not substitutes: rotate the EVM signer set and the BTC descriptor as one process and add a new `bytes32 public btcDescriptorHash` state variable next to `_enclaveSigners`, updated inside the existing `UpdateEnclaveSigners` execution branch, so a desync between the two registries is observable on-chain; migrate the BTC UTXOs immediately on compromise rather than only rotating on EVM; and add a `MIN_TIMELOCK_DURATION` floor enforced in the constructor and `SetTimelockDuration` so the timelock cannot be set to zero.

Update

Client's response

Acknowledged (no enclave-only fix exists) — the desync is between two rotation processes the enclave cannot observe. The fix couples them at the contract (BTC descriptor

hash published next to the signer set, with a timelock floor), after which the enclave gates on membership; contract-side design comes first.

Oxorio's response

The risk remains and is accepted. Removing a compromised key from the EVM `_enclaveSigners` still does not revoke its Bitcoin/RGB signing power, because `sign_psbt` signs any input matching its derived key with no cross-layer membership check; the reverse desync can make a threshold unreachable and lock funds. There is no enclave-only remediation, since the enclave cannot observe the two independent rotation processes. The compensating control today is operational: rotation is performed as a timelocked procedure, and the client plans to publish the BTC descriptor hash next to the EVM signer set. Before production, EVM signer changes, the BTC descriptor update, and the BTC UTXO migration must be executed as one coordinated procedure, and a `MIN_TIMELOCK_DURATION` floor must prevent a zero-timelock rotation. Full closure requires publishing the descriptor hash on-chain and having the enclave gate signing on the attested authorized set.

M-04	Mint path (`signPsbt`) has no SPV / RGB / nonce / deadline → stale-data minting in `enclave/src/server.rs`
Severity	**MAJOR**
Status	• FIXED

> MAJOR: the mint path authorizes new token issuance, so the absence of any freshness/finality binding is a fund-impacting weakness (supply inflation / double-credit), not a hardening note.

Location

File	Location	Line
server.rs	`handle_sign_psbt`	493-508
psbt_crosscheck.rs	`validate_psbt_request` (forks on `evm_tx_hash`; bridge-mode cross-check, no SPV/RGB/nonce/deadline)	42-86
keys.rs	`sign_psbt` (in vanilla mode signs every bridge-owned input)	—

Description

In the file `enclave/src/validation/psbt_crosscheck.rs`, `validate_psbt_request` forks on `evm_tx_hash`: an **empty** hash selects **VANILLA mode** (a shape-only check that returns `Ok`), a non-empty hash selects **BRIDGE mode** (the 32-byte hash check, `evm_event_valid`, `evm_event_finalized`, and `evm_amount ≥ psbt_output + evm_commission`). Both `evm_event_valid` and `evm_event_finalized` are booleans supplied by the **untrusted listener**. Unlike the EVM→RGB leg (where SPV independently confirms the anchor against the enclave's own PoW-verified header chain), in the file `enclave/src/server.rs` the mint path `handle_sign_psbt` carries no SPV proof, no RGB validation, no nonce and no deadline.

```
// psbt_crosscheck.rs — validate_psbt_request
if req.evm_tx_hash.is_empty {
    return Ok(); // VANILLA: shape-only, signs anyway
}
require(req.evm_event_valid); // untrusted listener boolean
require(req.evm_event_finalized); // untrusted listener boolean
// no nonce, no deadline
```

What "vanilla mode" is, and why it exists. The PSBT signing path is deliberately dual-purpose. Vanilla mode is the branch the enclave takes for *plain-Bitcoin* operations that have no EVM/RGB counterpart — e.g. `create_utxo` and federation housekeeping spends — where there is no `evm_tx_hash` to cross-check against, so `validate_psbt_request` performs only the BIP-174 shape-whitelist (non-empty bytes, parses as a PSBT, ≥1 input) and returns `Ok`. This is the PSBT-side mirror of the EVM leg's `consignment.is_empty` short-circuit, and it is *intentional* behaviour. The danger is structural: the selector between "fully cross-checked bridge mint" and "shape-only vanilla sign" is a single attacker-influenced field — an **empty** `evm_tx_hash`. Because vanilla mode then lets `enclave/src/keys.rs::sign_psbt` sign **every bridge-owned input** (the cross-check is the only thing that would otherwise constrain which inputs are signable), the entire safety of the mint path rests on an out-of-enclave guarantee that *bridge* mints are never submitted with an empty `evm_tx_hash`. Nothing inside the enclave enforces that invariant; it is exactly the open client-tracked concern. A buggy or hostile backend that submits a bridge mint with an empty `evm_tx_hash` silently downgrades it to a shape-only signature.

Even in bridge mode the binding is weak: `evm_event_valid` / `evm_event_finalized` are trusted booleans from the host, and there is no nonce or deadline, so the same `evm_tx_hash` can be replayed and stale/re-orged events can be forced `true`.

Reproduction:

1. The backend submits a `SignPsbt` for a real, finalized deposit with `evm_event_valid = true` / `evm_event_finalized = true`; the quorum mints.
2. The backend re-submits the same `evm_tx_hash` with the same booleans. Nothing in the enclave records that this deposit was already minted, so the quorum mints again — double-credit of one deposit.
3. Alternatively, the backend forces the booleans `true` for an unfinalized or re-orged event; with no independent finality check the enclave mints against data later orphaned.
4. Submitting an **empty** `evm_tx_hash` skips the cross-check entirely (the vanilla bypass): a bridge mint is signed with a shape-only check and `sign_psbt` signs all bridge-owned inputs.

The result is supply inflation / double-credit driven purely by replayed or stale listener booleans, or by a bridge mint mis-routed into the vanilla branch.

Recommendation

We recommend carrying an explicit, signed-over `operation_kind` (`Vanilla` / `BridgeMint`) instead of inferring mode from an empty `evm_tx_hash` and rejecting a `BridgeMint` with an empty hash; verifying the deposit event from a TEE-trusted EVM source instead of the listener booleans (see the bridge-mode listener-flag finding for the concrete in-enclave EVM-proof verification scheme); adding a per-deposit nonce/dedup on the verified `evm_tx_hash` plus a `deadline`; and constraining what vanilla signing may sign through an explicit

enclave-checked allowlist of permitted output scripts and amount caps in `keys.rs::sign_psbt`, rather than signing any input that matches the bridge key.

Update

Client's response

Fixed (different approach) — closed by the signPsbt cluster: in-enclave FundsIn verification with a confirmation-depth floor (PR #136), vanilla-mode bypass removed (PR #102), PSBT identity-bound to the validated consignment (PR #79; mint shape in PR #146), operation dedup (PR #100).

Partially fixed at: `2a2b929f`, `9248ed70`, `43355713`, `ca2e7590`, `7b675b7a`

Oxorio's response

Remaining issue: The mint path is materially stronger, but two invariants are still violated in the shipped release. First, the in-enclave `FundsIn` verification reads the EVM receipt and header over a raw RPC relayed by the untrusted host, so the deposit evidence is not host-independent (this is the same gap tracked in the bridge-mode listener-flag finding). Second, the Inflation amount binding rejects only `asset_output_amount < net_credited`, so an asset output larger than the verified EVM credit is accepted — newly minted value that is not independently authorized. The operation dedup is also volatile and its reservation can block a legitimate retry.

Required remediation: Make the header-and-receipt verification path mandatory in production and remove the raw-RPC fallback; make replay protection transactional and durable; and for Inflation require exact amount equality (`asset_output_amount == net_credited`) or independently prove each output (recipient and allowed change) with an over-mint rejection test.

Reproduction: With the shipped image, a compromised host returns a coherent but fabricated receipt and block header, and the enclave signs a mint against a deposit that was never finalized. Separately, an Inflation request with `net_credited = 900` and `asset_output_amount = 5,000` passes because the check only rejects outputs below the credit, minting 5,000 units of asset for 900 units of verified credit.

Client's response

Fixed.

The remaining points identified in the re-audit have now been addressed.

1. Host-independent EVM verification
   Production bridge enclaves are required to use the Helios-verified EVM path. This is enforced by the production security policy, not merely by deployment convention: a production enclave refuses to boot if the EVM source is RawRpc or Disabled, and requires HeliosVerified together with a pinned weak-subjectivity checkpoint.

Therefore, the raw host-relayed RPC path cannot be used as the trust source by a production bridge signer. The header and receipt evidence used for FundsIn verification is verified through Helios against the enclave-pinned trust root.

2. Replay handling
   The local replay reservation has also been made transactional in PR #187: the canonical, chain-verified funds_in_operation_id is reserved before signing and committed only if signing succeeds. A signing failure rolls the reservation back, so a transient error cannot consume the operation and prevent a legitimate retry.
   As clarified separately, durable replay protection is intentionally enforced by the authoritative protocol state rather than by persistent enclave-local storage. The enclave remains stateless; its in-memory deduplication is defence-in-depth.

3. Inflation amount binding
   The remaining over-mint issue is fixed in PR #181.
   For an RGB Inflation transition, the enclave now requires:
   asset_output_amount == net_credited
   where:
   net_credited = source_amount - source_commission
   The previous one-sided check that only rejected asset_output_amount < net_credited has been removed for the Inflation path. Consequently, an output greater than the verified EVM credit is rejected as well.
   A regression test was added specifically for the reported reproduction: an Inflation request with a verified credit of 1,000 and asset_output_amount = 1,500 is rejected with a mint amount mismatch.
   This directly implements the remediation requested in the report: exact equality between the minted OS_ASSET amount and the independently verified net EVM credit. Therefore, the reported Inflation over-mint vector is closed.

Fixed at: `2a2b929`, `ca2e759`, `9248ed7`, `b37d81a`

M-05	`SignPsbt` vanilla mode bypasses EVM-to-RGB authorization when `evm_tx_hash` is empty in `psbt_crosscheck.rs`, `server.rs`, `keys.rs`
Severity	**MAJOR**
Status	• FIXED

> Rated MAJOR for cross-report consistency: this is the same mint-path bypass described in the related finding "Mint path (`signPsbt`) has no SPV / RGB / nonce / deadline → stale-data minting", rated MAJOR. It escalates to CRITICAL if `SignPsbt` is reachable in production for bridge-controlled UTXOs without a tightly controlled operational `create_utxo` flow; the current enclave code does not enforce that restriction.

Location

File	Location	Line
server.rs	`handle_sign_psbt` validates, then signs the supplied PSBT	493-508
psbt_crosscheck.rs	vanilla mode returns before bridge checks	41-45
keys.rs	`KeyManager::sign_psbt` signs matching P2WSH/Taproot inputs	335-398

Description

In the file `psbt_crosscheck.rs`, an empty `evm_tx_hash` switches `SignPsbt` into vanilla mode:

```
// psbt_crosscheck.rs
if req.evm_tx_hash.is_empty {
    tracing::info!("PSBT signing: vanilla mode (no evm_tx_hash, skipping EVM cross-checks)");
    return Ok();
}
```

Vanilla mode still parses the PSBT and requires at least one input, but it skips every bridge authorization predicate: no EVM event proof, no listener validity/finality checks, no amount check, no token/asset binding, no recipient binding, and no replay context. After this early return, `handle_sign_psbt` immediately calls:

```
let (signed_psbt, inputs_signed) = state.sign_psbt(&req.psbt_bytes)?;
```

`KeyManager::sign_psbt` then signs any P2WSH/Taproot input that matches the enclave's Bitcoin key and script checks. This means a compromised listener can intentionally omit `evm_tx_hash` and ask the enclave to sign a PSBT outside the EVM-to-RGB authorization path.

The code comments say vanilla mode is used for `create_utxo` and other plain BTC operations, but the enclave does not authenticate that caller intent. The mode is selected only by a user-controlled request field (`evm_tx_hash.is_empty`).

If a hostile listener can reach production `SignPsbt`, it can request signatures over arbitrary PSBTs spending bridge-controlled UTXOs, limited by the scripts/keys the enclave is allowed to sign. This can authorize Bitcoin/RGB-side movements without a corresponding EVM deposit.

Proof of Concept

1. Prepare a valid PSBT spending a bridge-controlled UTXO whose input script/key matches the enclave signer. The same PSBT shape used by legitimate bridge PSBT signing is sufficient.
2. Build a `SignPsbtRequest` with `psbt_bytes = serialized_psbt`.
3. Set `evm_tx_hash = []` (empty bytes).
4. Leave the bridge-enrichment fields empty or false: `evm_event_valid = false`, `evm_event_finalized = false`, `evm_amount = 0`, `evm_commission = 0`, `psbt_output_amount = 0`, `rgb_asset_id = ""`.
5. Send the request to `SignPsbt`.
6. `validate_psbt_request` accepts the request in vanilla mode and returns before any EVM/RGB bridge checks.
7. `handle_sign_psbt` calls `state.sign_psbt`, and the enclave signs every matching PSBT input.

Expected result: production bridge PSBT signing should require a proven EVM deposit or an explicitly authenticated operational mode. Actual result: an empty `evm_tx_hash` selects vanilla mode and bypasses the EVM-to-RGB authorization checks.

Recommendation

We recommend making vanilla PSBT signing an explicit, separately authorized mode rather than a default selected by an empty `evm_tx_hash`. Concretely:

- **How the mode is selected.** Stop inferring intent from `evm_tx_hash.is_empty` (`psbt_crosscheck.rs:41-45`). Route plain-BTC operations through a distinct request type / endpoint (e.g. `SignCreateUtxo`) so the caller's intent is explicit and cannot be reached by merely omitting a field on the bridge endpoint.
- **What vanilla mode is allowed to sign.** Bind the `create_utxo` path to a strict, enclave-checked allowlist: the permitted output scripts (e.g. the known bridge change / funding scripts), an amount cap, and the allowed destinations. The enclave should validate,

against the parsed PSBT, that every non-change output matches this allowlist and reject anything else, rather than signing any input that matches its key in `keys.rs:335-398`.

- **Where the authorization comes from.** Gate vanilla mode on the attested `SigningPolicy` from the production-policy finding via its `allow_vanilla_psbt` flag, and commit that flag (plus the allowlist parameters) into attestation `user_data`. For production bridge enclaves where `create_utxo` is not needed, disable vanilla mode entirely so an empty `evm_tx_hash` fails closed instead of unlocking an unauthenticated signing path.

Update

Client's response

Fixed as recommended — the empty-`evm_tx_hash` mode selection is gone: plain-BTC ops use an explicit `SignBtc` request gated by operator pins and restricted to vanilla-account UTXOs; bridge-mode always runs the full cross-checks (PR #102).

Fixed at: `9248ed70`

M-06	`SignPsbt` bridge mode trusts listener-supplied EVM validity and finality flags in `psbt_crosscheck.rs`, `server.rs`, `proto/enclave.proto`
Severity	**MAJOR**
Status	• FIXED

> Scope: this finding is the **bridge-mode** case (`evm_tx_hash` non-empty), where EVM validity/finality is accepted as two listener booleans. It is distinct from — and the concrete root cause of — the broader mint-path finding (no SPV/RGB/nonce/deadline on `SignPsbt`); the empty-hash **vanilla-mode** authorization bypass is tracked as its own finding. The three together form the `SignPsbt` trust-boundary cluster.

Location

File	Location	Line
server.rs	`handle_sign_psbt` calls `validate_psbt_request` before signing	493-508
psbt_crosscheck.rs	`validate_psbt_request` bridge-mode EVM checks	50-84
proto/enclave.proto	`SignPsbtRequest` EVM enrichment fields	171-188

Description

In the file `psbt_crosscheck.rs`, bridge mode starts when `evm_tx_hash` is non-empty. The enclave then checks only the shape of `evm_tx_hash` and two boolean fields provided by the listener:

```
// psbt_crosscheck.rs
if req.evm_tx_hash.len != 32 {
    return Err(EnclaveError::CrossCheck(...));
}

if !req.evm_event_valid {
    return Err(EnclaveError::CrossCheck("EVM event not validated by Listener".into));
}

if !req.evm_event_finalized {
    return Err(EnclaveError::CrossCheck("EVM event not yet finalized".into));
}
```

The enclave does not verify an EVM receipt proof, a block header, a Merkle Patricia proof, a finalized checkpoint, or even the event topics and data that produced the fields in the request. As a result, a compromised listener can supply any 32-byte `evm_tx_hash`, set `evm_event_valid = true` and `evm_event_finalized = true`, and move the request to the amount checks.

This is exactly the trust boundary the enclave is expected to close. The PSBT signer does verify whether the PSBT input is spendable by the enclave key, but it does not prove that the PSBT is authorized by a real finalized EVM deposit. The bridge-mode comment says "Full cross-checks", but the EVM side of the cross-check is an assertion from the listener.

If the listener is compromised, or if its event-validation code is wrong, the enclave can sign a PSBT for an EVM-to-RGB operation that is not backed by a finalized deposit. The impact is stale or fabricated mint / withdrawal authorization on the RGB/Bitcoin side, limited by the UTXOs and signing policy available to the enclave.

This finding is the concrete root cause of the stale-data PSBT-minting finding reported separately in this report: that finding observes the missing SPV/RGB/nonce/deadline on the mint path, and this one pinpoints why bridge mode still passes — the EVM evidence is two trusted listener booleans.

Proof of Concept

1. Prepare a valid PSBT that spends an input controlled by the bridge / enclave key. The PSBT only needs to be syntactically valid and have at least one input; this is the only PSBT shape condition enforced before the bridge-mode checks.
2. Build a `SignPsbtRequest` with a non-empty `evm_tx_hash`, for example 32 arbitrary bytes such as `0xaaaaaaaa...aaaaaaaa`.
3. Set `evm_event_valid = true` and `evm_event_finalized = true`. These values are accepted as facts by the enclave; no receipt proof, block proof, or finalized EVM header is required in the request.
4. Set the amount fields so that the local arithmetic check passes, for example `evm_amount = 1000`, `psbt_output_amount = 900`, `evm_commission = 50`.
5. Send the request to `SignPsbt`. In `server.rs`, `handle_sign_psbt` calls `validate_psbt_request(&req)` and then immediately calls `state.sign_psbt`.
6. The request passes bridge-mode validation even though the enclave has not proven that `evm_tx_hash` corresponds to a real finalized `FundsIn` event.

Expected result: the enclave should require verifiable EVM finality evidence before signing. Actual result: the listener-provided booleans are enough to pass the EVM side of the PSBT bridge-mode check.

Recommendation

We recommend replacing the listener booleans with a self-verified EVM event proof, mirroring the posture the `SignEvm` path already takes for the RGB→EVM direction (it refuses to trust the host-supplied `consignment_valid` flag; raw consignment validation is

run from `server.rs:361-385` via `rgb.rs:251-279`, and SPV inclusion/finality is enforced from `server.rs:422-458` via `spv_crosscheck.rs:68-120`; `evm_crosscheck.rs:105-131` is only the integrity-hash check). The EVM→RGB direction should adopt the same "untrusted evidence, verified in-enclave" model. Concretely:

- **What to derive (not trust).** Stop consuming `evm_event_valid`, `evm_event_finalized`, `evm_amount`, `evm_commission`, and `operation_idx` as listener assertions. Derive each of them — plus the event's recipient, token/RGB asset, source/destination chain id, and bridge contract — directly from the decoded deposit (`FundsIn`) event log. The boolean "valid" collapses into "the proof verified"; "finalized" collapses into "the anchoring header meets the finality rule".
- **How to verify.** The enclave should (1) ABI-decode the deposit event's `topics` / `data` to extract the fields above, (2) verify a Merkle-Patricia inclusion proof of the receipt (or log) against the `receiptsRoot` committed in the EVM block header that the event claims to be in, and (3) verify that header against a finality rule the enclave enforces itself — either a minimum confirmation depth over an in-enclave EVM header chain or a finalized-checkpoint / sync-committee proof. The `chain_id` and bridge contract MUST be pinned against `BridgeConfig` and fail closed on mismatch, exactly as `validate_evm_request` already does (`evm_crosscheck.rs:185-200`).
- **Where the evidence comes from.** The listener/parent supplies the raw inputs — the receipt/log bytes, the MPT proof nodes, the block header, and the finality proof — as the request payload, but they are treated strictly as untrusted evidence that the enclave re-verifies cryptographically. The trust anchor is pinned inside the enclave (a known-good EVM finalized checkpoint / light-client sync-committee root, committed the same way the SPV checkpoint is), so a compromised listener cannot forge a header the enclave will accept. This is the direct analog of the existing `MerkleProofEntry` + SPV header-chain mechanism (`proto/enclave.proto:146-162`) used on the Bitcoin side, and it requires the enclave to track EVM finality the way it already tracks the Bitcoin header chain.

Update

Client's response

Fixed as recommended — the listener validity/finality booleans are gone. Bridge-mode signPsbt now verifies the `FundsIn` event in-enclave before signing: the receipt is fetched over the enclave's own RPC client, the log is decoded and pinned to the configured bridge contract (a wrong or ambiguous log is rejected), a confirmation-depth floor is enforced, and every predicate fails closed; a build without this verification refuses the bridge path entirely (PR #136). To make the RPC itself trustless — the host relays it — the enclave also integrates a Helios light client that cryptographically verifies the RPC against a pinned checkpoint (PR #136). Helios is implemented and validated but not yet enabled by default in the shipped enclave image: it is being brought into production once the operational and infrastructure maturity to run it is in place.

Partially fixed at: `2a2b929f`

Oxorio's response

Remaining issue: Replacing the listener booleans with in-enclave decoding and a confirmation-depth floor is a real improvement, but the evidence is still host-forgeable in the shipped release. The production image enables raw `evm-rpc`, and Helios remains optional and falls back to raw RPC when its endpoint is absent, so the receipt and header the enclave "verifies" are relayed by the untrusted host. A compromised host can return an internally coherent but fabricated receipt and block header, which the enclave will accept.

Required remediation: Make Helios (or an equivalent authenticated header/proof verifier) mandatory in the production EIF, pin its finalized checkpoint and mode into attestation `user_data`, and refuse to operate on the bridge path when only raw RPC is available.

Reproduction: Run the shipped image (raw `evm-rpc`, Helios disabled). Present a bridge-mode `SignPsbt` referencing an `evm_tx_hash` that has no real finalized `FundsIn`. The host returns a coherent forged receipt whose log decodes to the pinned bridge contract with a satisfying confirmation depth; the enclave decodes it, passes every predicate, and signs a mint/withdrawal not backed by a finalized deposit.

Client's response

A release rgb-validation build now fails the boot gate unless FundsIn deposits are verified via the Helios light client. This closes the asymmetry with the Bitcoin side, which already requires SPV: production must not fall back to the host-relayed RawRpc source.
The attested posture committed Helios mode but not the weak-subjectivity checkpoint it trust-roots on, so a verifier could confirm an enclave ran Helios but not which chain it synced from — an operator could point it at an attacker-chosen checkpoint invisibly. Bind the checkpoint into attestation user_data so the trust root is verifiable, and refuse to boot a production Helios enclave that pins none.

Fixed at: `f1be252`, `7b395f6`, `2a2b929`

M-07	The enclave serves connections serially with no read or idle timeout in `main.rs`, `server.rs`, `framing.rs`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
main.rs	accept loop handles one connection at a time, inline	148-156
server.rs	single-threaded `handle_connection` / `process_connection`, no timeout plumbing	51-67
framing.rs	`read_message` blocks in `read_exact` with no deadline	10-29

Description

In the file `main.rs`, the enclave accepts and serves connections one at a time, calling `handle_connection` directly inside the accept loop:

```
for stream in listener.incoming {
    match stream {
        Ok(stream) => server::handle_connection(stream, &ctx),
        Err(e) => tracing::error!("accept error: {}", e),
    }
}
```

In the file `framing.rs`, `read_message` reads the length prefix and then the body with blocking `read_exact`, and no read, idle, or total-request timeout is configured anywhere on the connection:

```
stream.read_exact(&mut len_buf)?;
let len = u32::from_le_bytes(len_buf);
...
let mut buf = vec![0u8; len as usize];
stream.read_exact(&mut buf)?;
```

Because handling is serial, a single connection occupies the enclave for the whole duration of its request. A caller that connects and then sends nothing, sends only a partial length-

prefixed body, or dribbles bytes slowly leaves `read_exact` blocked indefinitely, and every other client is stalled behind it. The same serial design means a single maximally expensive request (for example a worst-case set of Merkle proofs, see the per-field input-amplification finding) blocks all other work while it runs. No authentication, proof-of-work, or valid payload is required to trigger the stall, which makes this the cheapest denial-of-service path against the enclave; it applies on every network including mainnet and a single idle socket can take the signing service offline, which is why it is rated MAJOR rather than a network-dependent warning.

Proof of Concept

1. Open a connection to the enclave's listening endpoint.
2. Send a 4-byte length prefix announcing a non-zero body, then send no further bytes (or send the body one byte at a time with long pauses).
3. The enclave's `read_message` blocks in `read_exact` waiting for the rest of the body.
4. Because the accept loop is serial, no other connection is serviced while this one is parked.
5. Hold the connection open. The signing service is unavailable for the duration with a single idle socket and no resources spent by the attacker.

Expected result: a stalled or slow connection should time out and be dropped without affecting other clients. Actual result: one parked connection blocks the entire enclave indefinitely.

Recommendation

We recommend enforcing per-connection deadlines and decoupling slow connections from overall availability. Concretely, set a read/idle timeout and a maximum total-request duration on each accepted stream (`set_read_timeout` plus a deadline around `read_message`) and drop connections that exceed them; reject a length prefix whose body does not arrive within the deadline. In addition, serve connections with bounded concurrency (a small worker pool with a connection cap) rather than strictly serially, so one slow or expensive request cannot stall all others, while still capping total resource use.

Update

Client's response

Fixed as recommended — bounded worker pool with per-connection read/idle and total-request timeouts (PR #117).

Fixed at: `67c7b622`

M-08	Cloning authorization rests on a static shared secret while reproducible PCRs cannot distinguish a legitimate enclave from an attacker's identical image in `server.rs`, `attestation.rs`, `cloning.rs`, `main.rs`
Severity	**MAJOR**
Status	• ACKNOWLEDGED

Location

File	Location	Line
server.rs	`handle_get_clone` identity / PCR / digest checks before sealing the seed	777-827
attestation.rs	peers are verified against the enclave's *own* PCRs	71-88
cloning.rs	`HMAC(cloning_secret, encryption_pubkey)` is the authorization proof	108-123
main.rs	`UTEXO_CLONING_SECRET` is a static env var on the untrusted parent	58-64
enclave.proto	`InitiateCloningRequest.cloning_secret` is a plaintext `string` request field	251-254
server.rs	`handle_initiate_cloning` takes the secret in cleartext from the request and computes the HMAC digest	708-729

Description

In the file `server.rs`, the donor side of the cloning handshake decides whether to seal its seed to a requester using two checks: the requester's attestation PCRs must equal the donor's own PCRs, and an HMAC digest must verify against the configured cloning secret:

```
// 2. Verify the requester attestation chain + PCRs.
let expected_pcrs = attestation::get_own_pcrs?;
let verified =
    attestation::verify_peer_attestation(&req.requester_attestation, &expected_pcrs, None)?;
...
// 6. Digest authenticity: HMAC(donor_secret, encryption_pubkey) must match.
state.with_donor_cloning_secret(|secret| {
    if !cloning::verify_cloning_digest(secret, &req_encryption_pk, &req_digest) {
        return Err(EnclaveError::DigestMismatch);
    }
    Ok()
})?;
```

The PCR check uses the enclave's *own* measurements as the expected value for the peer (`attestation.rs`, `get_own_pcrs`). PCR0/1/2 measure the enclave image, and the image is open source and reproducible, so anyone can build the same image and obtain identical PCRs. The PCR check therefore confirms only that the peer runs the same code — it cannot distinguish an enclave operated by the legitimate operator from an enclave an attacker built from the public source.

To make this concrete: PCR0/1/2 are the AWS Nitro platform measurements taken at enclave launch — PCR0 is the hash of the entire enclave image (EIF), PCR1 covers the kernel and bootstrap, and PCR2 covers the application. All three are derived deterministically from the image contents, so two builds of the same source produce byte-identical values, and the enclave compares only these three (`read_own_pcrs` returns `ExpectedPcrs::new(pcr0, pcr1, pcr2)`). They therefore attest *what code is running*, not *who is running it* — nothing in the compared measurement set is tied to the operator.

That leaves the cloning secret as the only gate that actually authorizes seed extraction. `UTEXO_CLONING_SECRET` is a single, static, cluster-wide value supplied to the enclave as an environment variable on the untrusted parent (`main.rs`). The handshake has no rate-limiting, no operator-in-the-loop approval, no per-clone authorization, and no revocation or rotation binding.

The same static secret is also handed to the requester role over the RPC surface in cleartext. `InitiateCloningRequest.cloning_secret` is a plaintext `string` field (`proto/enclave.proto`), and `handle_initiate_cloning` reads `req.cloning_secret` directly to compute `HMAC(cloning_secret, encryption_pubkey)` (`server.rs`). This is asymmetric with the donor role, which receives the secret out-of-band as an environment variable and never on the wire: the requester instead obtains the same cluster-wide secret as a runtime request field that the untrusted parent constructs and relays, so the value is exposed to the parent process (its memory, logs/tracing, and any relay hop). Because the secret is static and the digest binds only to the ephemeral X25519 pubkey, anyone who observes it once can recompute a valid digest for any pubkey and clone any donor. Note this is parent / host-level exposure rather than a classic on-path network MITM — over vsock the untrusted parent is the channel.

As a result, a one-time disclosure of `UTEXO_CLONING_SECRET` (from the host environment, process listing, configuration management, or any operational leak) lets anyone build the public enclave image, present a valid NSM attestation with matching PCRs, prove knowledge of the secret, and have a donor seal the full seed to an attacker-controlled X25519 key. Because the seed derives every key (EVM bridge, EVM gas, BTC, RGB taproot) and is shared across every instance that runs it, this is full key compromise for that signing identity across the cluster, not a single enclave instance.

To be explicit about the impact model: the attacker never needs their own enclave to be registered as a bridge signer or added on-chain anywhere. Cloning does **not** mint a new, unauthorized signer whose EVM address the federation would have to approve — it copies

the donor's existing seed, from which `from_seed` deterministically re-derives the very EVM/BTC/RGB private keys whose addresses are *already* in `_enclaveSigners`. Possession of the seed is therefore possession of the live signing identity, so the attacker can produce signatures the existing on-chain configuration already accepts; there is no separate "trust the attacker's enclave" step that the operator could withhold.

How many clones are required depends on the enclave signer model on-chain. If the bridge trusts a single shared-seed cluster identity (the "one seed, replicated cluster-wide" model this codebase implements, threshold effectively 1 over that key), a single successful clone is immediately sufficient. If instead the on-chain `enclaveThreshold` is `M`-of-`N` over *distinct* enclave seeds (the reference deployment's 2-of-3 enclave set), one clone yields only one of the `M` already-authorized keys, so forging a `fundsOut` release requires cloning `M` distinct donors. That remains practical precisely because `UTEXO_CLONING_SECRET` is a single, static, cluster-wide value: leaking it once lets the attacker clone as many distinct donors as the threshold needs. Either way the decisive property is the same — every cloned key is already authorized on-chain, so no federation approval gates the attack.

Proof of Concept

1. Obtain `UTEXO_CLONING_SECRET`. It is static and shared cluster-wide and lives as a plaintext env var on the untrusted parent, so a single operational leak (host compromise, log, config store, process environment) is sufficient.
2. Build the enclave image at the deployed revision from the public source. The resulting PCR0/1/2 match the production enclaves bit-for-bit.
3. Run the attacker image as a cloning requester: it generates an ephemeral X25519 keypair, obtains a real NSM attestation binding that pubkey, and computes `HMAC(UTEXO_CLONING_SECRET, pubkey)`.
4. Drive the handshake to a donor enclave (`InitiateCloning` then a relayed `GetClone`).
5. The donor verifies PCRs (match — same image), the pubkey binding, and the HMAC digest (matches — secret known), then seals its seed to the attacker's pubkey.
6. The attacker completes `SetClone`, decrypts the seed, and now holds the cluster's full key material.

Expected result: seed extraction should require an authorization that an attacker cannot satisfy merely by running the public image and learning one static secret. Actual result: matching PCRs plus knowledge of the shared secret are sufficient.

Recommendation

We recommend binding cloning authorization to an attested operator identity rather than to a static shared secret that a reproducible image cannot encode. Concretely:

- **Bind authorization to an attested commitment.** Commit an operator-controlled authorization public key (or a per-cluster identity) into the attestation `user_data` / a

verifier-visible commitment, and require the requester to prove control of that key, so that running the public image is not by itself sufficient.

- **Add an approval and rate-limiting layer.** Require operator-in-the-loop approval or a quorum signature for each `GetClone`, rate-limit the donor's cloning endpoint, and support rotation and revocation of the cloning authority.
- **Keep the distinguishing secret off the untrusted host.** Since reproducible PCRs cannot carry operator identity, any distinguishing secret must be high-entropy and never present in plaintext on the parent — for example released into the enclave only through an attestation-gated KMS path — and it must be rotatable without redeploying every enclave. As an immediate step, remove `cloning_secret` from the runtime RPC surface so the parent cannot pass it as a plaintext `InitiateCloningRequest` field; moving it to an environment variable (as on the donor side) only removes the request-wire leak and is not by itself a sufficient boundary, since the parent/ops-controlled environment can still expose the value. Note that merely "sending a proof of knowledge instead of the secret" does not help while generating that proof still requires the raw shared secret to be placed inside the requester via the untrusted parent. Prefer replacing the shared secret with a one-time operator-signed cloning authorization, bound to the requester attestation, the requester's ephemeral X25519 public key, the intended donor identity, and a nonce/session id with an expiry, so it cannot be replayed or redirected to a different enclave.

> Related defence-in-depth (previously client-tracked): the `cloning_secret` is not cross-bound to the image measurement. Extending PCR2 (or an equivalent measured slot) with the cloning secret at init, and verifying that binding on the peer side during the handshake, additionally defends against a leaked secret being used against a different image and against a substituted image cloning with a known secret.

Update

Client's response

Acknowledged (deferred) — binding the secret into PCR2 makes the image measurement secret-dependent, which conflicts with the reproducible-PCR pipeline (I-16); cloning already requires a mutually attested, PCR-checked handshake. Scheduled with the cluster-clone architecture (#73).

Oxorio's response

The risk remains and is accepted. Matching PCRs prove only that the peer runs the same open-source, reproducible image — not that it belongs to the legitimate operator — so the cluster-wide static `UTEXO_CLONING_SECRET` is the sole gate on seed extraction. Anyone who learns that secret once (it is a plaintext env var on the untrusted parent and is also passed to the requester as a plaintext request field) can build the public image, present a valid NSM attestation with matching PCRs, prove knowledge of the secret, and have a donor seal the full seed — full cluster key compromise. The compensating controls are the mutual attestation and PCR equality, which restrict cloning to the approved code image, plus operational protection of the secret. Before production, cloning must be restricted to

approved PCRs with the secret protected, rotated, and its exposure minimized (fewest donors, least access); the target design must replace the static shared secret with an attested operator identity or a one-time operator-signed cloning authorization (bound to the requester attestation, ephemeral X25519 key, donor identity, and an expiry) that never places a distinguishing secret on the untrusted host.

M-09	Inability to pass `EVM` request validation under a specific enclave configuration in `evm_crosscheck.rs`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
`evm_crosscheck.rs`	-	`175`
`evm_crosscheck.rs`	-	`178`
`evm_crosscheck.rs`	-	`193`
`evm_crosscheck.rs`	-	`199`
`evm_crosscheck.rs`	-	`212`

Description

At the referenced locations, a number of conditions are checked that may lead to errors under different `bridge_config` configurations. In `bridge_config`, it is possible to set `chain_id`, `bridge_contract`, or `rgb_asset_id` to empty or zero values, while the enclave will still be considered configured:

```
pub fn is_configured(&self) -> bool {
    self.chain_id != 0 || self.bridge_contract != [0u8; 20] || !self.rgb_asset_id.is_empty
}
```

This leads to the following inconsistencies:

`1)` In the case of `bridge_config.chain_id = 0`, `EVM` request validation will never succeed:

```
if req.chain_id == 0 {
    return Err(EnclaveError::CrossCheck("chain_id must be > 0".into));
}
//...

if bridge_config.is_configured {
    if req.chain_id != bridge_config.chain_id {
```

```
            return Err(EnclaveError::CrossCheck(format!(
                "chain_id mismatch: request {} != pinned {}",
                req.chain_id, bridge_config.chain_id
            )));
        }
        //...
    }
```

2) The case where `bridge_contract == [0u8; 20]` means that the enclave considers an `EVM` request for a contract with the zero address to be valid:

```
    if req.proxy_contract.len != 20 {
        return Err(EnclaveError::CrossCheck(format!(
            "proxy_contract must be 20 bytes, got {}",
            req.proxy_contract.len
        )));
    }
    //...

    if bridge_config.is_configured {
        //...
        if req.proxy_contract.as_slice != bridge_config.bridge_contract {
            return Err(EnclaveError::CrossCheck(format!(
                "proxy_contract mismatch: request {} != pinned {}",
                hex::encode(&req.proxy_contract),
                hex::encode(bridge_config.bridge_contract)
            )));
        }
        //...
    }
```

3) The error message explicitly states that all `3` variables must be set in the configuration. However, in practice, apart from `bridge_config.rgb_asset_id`, it is sufficient to set only `bridge_config.chain_id` for the check against `0` to pass (see the first item):

```
    if bridge_config.rgb_asset_id.is_empty {
        return Err(EnclaveError::CrossCheck(
            "bridge config pinned chain/contract but RGB_ASSET_ID is empty — \
             set all three env vars or none"
                .into,
        ));
    }
```

Recommendation

We recommend replacing the OR-based `is_configured()` with a fail-closed `is_fully_configured()` that requires all three of `(chain_id, bridge_contract, rgb_asset_id)` to be non-zero/non-empty before the enclave validates or signs any bridge request, so a partially populated `bridge_config` can never be treated as configured. This is the configuration-level instance of the explicit attested `SigningPolicy` proposed in the production-policy finding and should be implemented together with it.

Update

Client's response

Fixed as recommended — `is_configured()` requires all of chain_id, bridge_contract and rgb_asset_id (PR #98).

Fixed at: `2a9566d5`

3.3 WARNING

W-01	Malformed ABI tail in `calldata`: no dynamic offset/ bounds validation in `evm_crosscheck.rs`
Severity	**WARNING**
Status	• FIXED

> WARNING (DoS / revert load; not direct theft — the critical numeric fields are still bound).

Location

File	Location	Line
evm_crosscheck.rs	reads only fixed calldata byte offsets (legacy: `amount` at byte 68 / `commission` at byte 100; mint-burn: `amount` at byte 36); the dynamic ABI tail `string/bytes` is not validated	—

Description

In the file `enclave/src/validation/evm_crosscheck.rs` the cross-check validates the critical numeric slots at fixed calldata byte offsets (legacy 6-arg: `amount` at byte 68, `commission` at byte 100; mint-burn 8-arg: `amount` at byte 36) but never validates the ABI head's dynamic offsets or the tail (`sourceAddress`, `proof`, `settlementData`).

```
// evm_crosscheck.rs — fixed offsets read, dynamic tail unchecked
// legacy 6-arg fundsOut: amount at byte offset 68, transactionId ("commission") at byte
offset 100
let cd_amount = extract_uint256_as_u64(&req.call_data, 68)?;
let cd_commission = extract_uint256_as_u64(&req.call_data, 100)?;
// mint/burn 8-arg fundsOut: amount at MINTBURN_AMOUNT_OFFSET (byte offset 36); no
commission slot
let calldata_amount = extract_uint256_as_u64(&req.call_data, MINTBURN_AMOUNT_OFFSET)?;
// sourceAddress / proof / settlementData (dynamic) — offsets & lengths NOT validated
```

Reproduction:

1. The backend submits calldata whose fixed slots look valid while the dynamic tail is malformed (broken offsets, or no tail at all).

2. The numeric checks pass, but downstream parsing reverts or the enclave signs over an unparsed tail (which also carries `fundsInIds` — see the operation-identifier uniqueness finding).

Classic spoofing of the critical numeric fields is already cut; the residual is a revert/DoS surface plus the signed-but-unparsed tail.

Recommendation

We recommend parsing and bounds-checking the full ABI (head offsets in-range and monotonic, tail lengths inside the buffer) in `validate_evm_request` before signing — the same parse needed for the operation-identifier uniqueness finding — and adding a contract-derived calldata fixture test.

Update

Client's response

Fixed (different approach) — the full calldata is ABI-decoded (head and dynamic tails, bounds-checked) before signing (PR #128), with per-field size caps (PR #118) and canonical-encoding enforcement rejecting trailing junk and overlapping tails (PR #145).

Fixed at: `38c46e90`, `ef9aca2d`, `0beefb68`

W-02	Cross-layer selector drift: enclave whitelist wider than the on-chain allowlist in `evm_crosscheck.rs`
Severity	**WARNING**
Status	• FIXED

WARNING (operational / fail-on-execution, not direct theft).

Location

File	Location	Line
evm_crosscheck.rs	whitelists both `0x1ad880b2` legacy 6-arg and `0xcddb768` mint/burn 8-arg	13-37
MultisigProxy.sol	default `teeAllowedCalls` = 8-arg only	184-187
Bridge.sol	only the 8-arg `fundsOut` exists	168

Description

In the file `enclave/src/validation/evm_crosscheck.rs` the enclave whitelists both the `0x1ad880b2` legacy 6-arg selector and the `0xcddb768` mint/burn 8-arg selector. In the contract `MultisigProxy` the default `teeAllowedCalls` is the 8-arg only, and in the contract `Bridge` only the 8-arg `fundsOut` exists.

```
// evm_crosscheck.rs
pub const FUNDS_OUT_SELECTOR_POOLS_LEGACY: [u8; 4] = [0x1a, 0xd8, 0x80, 0xb2]; // 6-arg
legacy
pub const FUNDS_OUT_SELECTOR_MINTBURN: [u8; 4] = [0xcc, 0xdd, 0xb7, 0x68]; // 8-arg
```

Reproduction:

1. The backend asks the enclave to sign the legacy `0x1ad880b2` operation; the enclave deems it valid and signs.
2. On-chain there is no 6-arg `fundsOut` and no allowlist entry, so execution reverts with `CallNotAllowed`.

"What the enclave deems valid" is a strict superset of "what is executable on-chain."

The drift also runs the other way. The enclave whitelist is limited to the two `fundsOut` selectors and contains no entry for the LayerZero adapter's outbound `UtexoLZAdapter.sendOut`, while `MultisigProxy` notes that adapter calls such as `sendOut` are "added later" to `teeAllowedCalls` (`MultisigProxy.sol:182`). If the outbound RGB →

non-Arbitrum flow requires the enclave to sign a `sendOut` (or `fundsOut` + `sendOut`) operation, the enclave cannot produce that signature at all, so the LayerZero path silently fails to operate even though the contracts allow it. In other words the two sets can be inconsistent in *either* direction: enclave-signable-but-not-executable (legacy selector) and on-chain-allowed-but-not-enclave-signable (`sendOut`).

Recommendation

We recommend keeping the enclave-signable selector set and the on-chain `teeAllowedCalls` set exactly consistent — neither a superset nor a subset of the other. Concretely: drop the legacy `0x1ad880b2` from `FUNDS_OUT_SELECTORS` (`evm_crosscheck.rs`) unless it is actually deployed and allowlisted on-chain; and, if the LayerZero outbound flow is in scope, add the `UtexoLZAdapter.sendOut` selector to the enclave whitelist (with its own validation rules) and to `teeAllowedCalls`, so the LZ binding can execute. Add a regression test asserting a bijection between enclave-signable selectors and `teeAllowedCalls` entries (every enclave-signable selector is allowlisted on-chain, and every allowlisted bridge selector the enclave is expected to sign is present in the whitelist), so future selectors cannot drift apart in either direction.

Update

Client's response

Fixed — the legacy selector was removed; the enclave whitelist is exactly the single `fundsOut` selector the contract allowlists, `0xcddb768` (#78), pinned by a known-answer test (PR #145).

Fixed at: `9174d8fa`, `0beefb68`

W-03	Cross-layer batch signature drift: `executeBatch` exists on-chain, enclave signs single-call only in `enclave/src/signing/evm.rs`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
evm.rs	`sign_request_digest` builds a single-call digest only	—
enclave.proto	`SignEvmRequest` carries a single `call_data` blob, no batch type	—
MultisigProxy.sol	`executeBatch` + `BridgeBatchOperation` / `_buildBatchDigest` (non-canonical `address[]`) vs `_hashAddressArray`	939-960, 1002-1008

Description

In the files `enclave/src/signing/evm.rs` and `proto/enclave.proto` the enclave API/digest builder produces only a single-call digest: `sign_request_digest` hashes one `call_data` blob (there is no batch type in the proto). In the contract `MultisigProxy` the on-chain `executeBatch` accepts a `BridgeBatchOperation` with an independent `batchNonce` and a `_buildBatchDigest`.

```
// MultisigProxy.sol — _buildBatchDigest (NON-canonical for address[])
keccak256(abi.encodePacked(targets)), // 20 bytes/element
// — correct helper, used elsewhere but NOT here:
// words[i] = bytes32(uint256(uint160(arr[i]))); // 32-byte pad (canonical)
```

Reproduction:

1. A caller attempts `executeBatch`; the only signatures the enclave can produce are single-call digests.
2. A single-call signature is invalid for the batch digest, so the batch surface is unreachable through the normal `SignEvm` flow.

There is a second defect in the on-chain batch digest itself: `_buildBatchDigest` hashes `targets` as `keccak256(abi.encodePacked(targets))` (`MultisigProxy.sol`) — a 20-byte-per-element packing — instead of the canonical 32-byte-per-element encoding, while the contract's own correct helper `_hashAddressArray` is used in

`proposeUpdateEnclaveSigners` / `proposeUpdateFederationSigners` but **not** here. (`bytes32[]` / `uint256[]` packings are coincidentally canonical; only `address[]` is affected.) The spec requires: "the signed payload MUST be identical across quorum members"; if the enclave ever grows a batch path built on a standard EIP-712 library it will compute a different digest than the contract, which also blocks the spec's multi-vendor-TEE roadmap and creates a wallet-UI spoof window.

Recommendation

We recommend either adding a batch-signing path to the enclave or removing the on-chain `executeBatch` / `_buildBatchDigest` (or documenting that the production TEE signs only `MultisigProxy.execute`); and regardless, fixing `_buildBatchDigest` to use the existing `_hashAddressArray(targets)` so the `address[]` encoding is canonical EIP-712.

Update

Client's response

Fixed — the on-chain generic `execute` / `executeBatch` path and its `_buildBatchDigest` were removed and replaced with typed, single-purpose TEE entry points (a typed `fundsOut` and a typed LZ `fundsOut`), each with its own EIP-712 typehash; the enclave authors exactly those single-call typed digests. With no `executeBatch` / `BridgeBatchOperation` / `_buildBatchDigest` left on-chain, the two layers now expose the same single-call action set and the non-canonical `address[]` batch encoding is gone (PR #73).

Fixed at: `316168fd`

W-04	DoS-ordering: expensive RGB validation runs before the cheap `consignment_hash` check in `enclave/src/server.rs`
Severity	**WARNING**
Status	• FIXED

> WARNING (availability / DoS; not direct theft).

Location

File	Location	Line
server.rs	`handle_sign_evm` calls `validate_consignment` first	—
rgb.rs	rgbstd validate + blocking Esplora	251-279
evm_crosscheck.rs	cheap `keccak256(consignment) == consignment_hash`	115-131

Description

In the file `enclave/src/server.rs`, `handle_sign_evm` calls `validate_consignment` (the expensive RGB deserialize, parser walk and blocking Esplora HTTP, see `enclave/src/validation/rgb.rs`) before the near-free tamper-detection check `keccak256(consignment) == consignment_hash` in `enclave/src/validation/evm_crosscheck.rs`.

```
// server.rs — handle_sign_evm (current order)
let validated = validate_consignment(&req.consignment)?; // expensive: rgbstd + blocking
Esplora
// only later (evm_crosscheck.rs):
if keccak256(&req.consignment) != req.consignment_hash {... } // cheap — runs too late
```

Reproduction:

1. The backend streams `SignEvm` requests with arbitrary consignment bytes and a deliberately wrong/empty `consignment_hash`.
2. Each request forces a full expensive RGB cycle before the cheap keccak rejects it.

On the single-threaded signing path this is a CPU/network amplification channel.

Recommendation

We recommend running the cheap checks first in `handle_sign_evm`: non-empty `consignment`/`consignment_hash` and `keccak256(consignment) == consignment_hash` (`evm_crosscheck.rs`) before the expensive `validate_consignment`, so a wrong hash never triggers an RGB cycle (pairs with the blocking Esplora HTTP finding's timeout).

Update

Client's response

Fixed as recommended — the `consignment_hash` integrity check runs before rgbstd validation on both paths (PR #128).

Fixed at: `38c46e90`

W-05	The EVM-lock-record predicate (the spec's same-`OpId` obligation) is delegated to the untrusted Listener in `evm_crosscheck.rs`
Severity	**WARNING**
Status	• ACKNOWLEDGED

WARNING (spec-conformance / trust-boundary; security): Listener compromise is required and the on-chain `FundsInNotFound` revert backstops the *fake-*`operationId` case, so on its own it is MEDIUM-class; it escalates to the MAJOR operation-identifier uniqueness scenario for the *real-but-unrelated-*`operationId` case.

Location

File	Location	Line
evm_crosscheck.rs	`validate_evm_request` — checks selector, consignment hash, amount, chain/contract pin, deadline; **no** lock-record check	60-236
RgbSettlementModule.sol	on-chain backstop (`FundsInNotFound`)	—

Description

The spec places a positive obligation on the TEE: "the corresponding EVM-side lock record exists for the same OpId." In the code this EVM-side lock record is keyed by the deposit `operationId` — the `uint256` supplied at `Bridge.fundsIn(..., operationId, ...)` and stored as `fundsInRecords[operationId]` — which is a distinct, backend-assigned identifier and not the RGB `OpId` itself; the spec's intent is that they be the same value threaded across domains, but the code never binds them. In the file `enclave/src/validation/evm_crosscheck.rs`, `validate_evm_request` verifies the calldata shape, the consignment integrity hash, the amount binding, the pinned chain/contract and the deadline — but it never independently verifies that the deposit `operationId`(s) the release settles correspond to real EVM-side `FundsIn` events. (The `SignEvmRequest` does not even carry an `operationId`; the deposit ids it settles are the `fundsInIds[]` encoded in the `fundsOut` calldata `settlementData`.) That check is performed (if at all) by the off-chain Listener, which sits **outside** the TEE trust boundary. The spec states: "a TEE moves trust away from the host operator … toward the correctness of the enclave code"; pushing this predicate onto the host inverts the trust model the enclave exists to enforce.

```
// evm_crosscheck.rs — validate_evm_request checks:
// selector · consignment hash · amount · chain_id/proxy_contract · deadline
// — NO independent "FundsIn(operationId) exists on EVM" check
```

Reproduction:

1. A buggy/hostile Listener presents a `fundsOut` release whose `settlementData` references a deposit `operationId` with no real EVM `FundsIn` lock.
2. The enclave signs (it never checked).
3. For a fabricated deposit `operationId`, the on-chain `FundsInNotFound` revert in `ethereum/src/settlement/RgbSettlementModule.sol` saves the case; for a *real-but-unrelated* deposit `operationId` (someone else's deposit) nothing on-chain catches it — that is the operation-identifier uniqueness path.

Recommendation

We recommend enforcing the lock-record predicate inside a trust boundary instead of delegating it to the listener. There are two ways to do this, in order of preference:

- **Preferred (zero-trust, in-enclave).** In `validate_evm_request`, parse the deposit `operationId`(s) the release settles (the `fundsInIds[]` carried in the `fundsOut` calldata `settlementData`), re-derive and verify each EVM-side `FundsIn(operationId, …)` record from a TEE-trusted EVM source — i.e. an event/receipt inclusion proof verified against an enclave-anchored EVM header chain, the same posture the RGB→EVM leg already uses for SPV — and fail closed if any referenced lock record does not exist. This keeps the check on the side of the trust model the enclave exists to enforce.
- **Lighter on-chain backstop.** If in-enclave EVM verification is too heavy for now, harden the existing on-chain check in `RgbSettlementModule.beforeFundsOut`. Today that function requires each visited id to resolve to a real `FundsIn` record while it consumes records until the release amount is covered, but it breaks once `remaining == 0`; any trailing ids in `fundsInIds[]` are not validated or consumed. What is missing is rejecting any release whose referenced set is not fully validated (no unconsumed trailing ids — see the `beforeFundsOut` settlement-loop finding in the Solidity report), instead of relying on the listener to have matched the deposit `operationId`. Do not conflate the release identifier with the deposit ids: the `fundsOut` release is keyed by `ctx.burnId` (there is no `operationId` on `FundsOutContext`), while `fundsInIds[]` are the distinct deposit `operationId`s it settles, so the binding must be expressed as set-membership-and-consumption of real deposits, not identifier equality.

The fake-`operationId` case (no real `FundsIn`) is already backstopped by the on-chain `FundsInNotFound` revert; the addition above is what closes the real-but-unrelated-`operationId` case, and it should be implemented together with the operation-identifier uniqueness finding so a single `FundsIn` cannot be settled by more than one `fundsOut`.

Update

Client's response

Fixed (different approach) — holds by construction: `fundsInIds` / `burnId` are enclave-authored from the validated consignment (PR #97), and every bridge-mode mint is signed only after in-enclave FundsIn verification (PR #136), so the signed identifiers always trace back to enclave-checked lock events.

Partially fixed at: `e8aea565`, `2a2b929f`

Oxorio's response

Remaining issue: the enclave still does not enforce a single durable, enclave-verified binding between an EVM lock record and the release that settles it. The lock-record predicate depends on FundsIn evidence relayed by the untrusted host (see the bridge-mode listener-flag finding), and the local replay/dedup protection can be sidestepped by an enclave restart, a sibling enclave in the cluster, TTL expiry, or cache eviction. Authoring the identifiers in-enclave prevents fabricated ids but does not by itself prove that each referenced deposit `operationId` corresponds to a real, not-yet-consumed EVM `FundsIn`.

Required remediation: derive the lock identifier from a canonical, authentic EVM event/receipt verified against an enclave-anchored EVM header chain (the same zero-trust posture the RGB→EVM leg uses for SPV), and require its durable on-chain consumption before a release is authorized, so a real-but-unrelated `operationId` cannot be reused across instances or restarts. Add cross-instance and post-restart replay tests.

Reproduction: the fabricated-`operationId` case is caught on-chain by `FundsInNotFound`, but a real-but-unrelated `operationId` (someone else's deposit) still passes the enclave, and the in-memory dedup that would otherwise catch a repeat is defeated by the volatile-dedup bypass described in the operation-identifier uniqueness finding (restart, sibling enclave, or TTL/eviction). This is the same reproduction as that MAJOR finding's release leg.

Client's response

Fixed / architectural clarification — durable replay protection is enforced by the authoritative protocol state, not by enclave-local persistence.

For the **RGB → EVM** flow, replay protection is enforced on-chain through `consumedBurnIds`. The enclave derives `burnId` from the validated RGB consignment, so replaying the same burn after an enclave restart, against another enclave instance, or after local dedup expiration produces the same `burnId` and is rejected by the EVM contract.

For the **EVM → RGB** flow, durable replay protection is enforced by the RGB state itself. RGB is stateful, and `rgb-lib` rejects reuse of an already-used recipient ID (`RecipientIDAlreadyUsed`). Therefore, replaying the same mint operation cannot create another valid RGB state transition using the same recipient.

W-06	`psbt_output_amount` is supplied by the listener instead of being derived from the PSBT in `psbt_crosscheck.rs`, `proto/enclave.proto`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
psbt_crosscheck.rs	amount consistency check	72-84
proto/enclave.proto	`psbt_output_amount` field	185-187

Description

In the file `psbt_crosscheck.rs`, the amount check compares three request fields:

```
let required = req
    .psbt_output_amount
    .checked_add(req.evm_commission)
    .ok_or_else(...)?;

if req.evm_amount < required {
    return Err(EnclaveError::CrossCheck(...));
}
```

The enclave parses `req.psbt_bytes` only to confirm that it is a valid PSBT with at least one input. It does not compute the output value from `psbt.unsigned_tx.output` and it does not identify which outputs are user outputs versus change. Therefore the listener controls both sides of the economic comparison: it supplies the claimed `psbt_output_amount`, and it supplies `evm_amount` / `evm_commission`.

A compromised listener can understate `psbt_output_amount` while providing a PSBT whose actual non-change output value is higher. The current code would accept the request if the stated values satisfy `evm_amount >= psbt_output_amount + evm_commission`.

The enclave can sign a PSBT whose real output value is not covered by the EVM deposit amount represented in the request. This creates an under-collateralized EVM-to-RGB operation if the PSBT spends bridge-controlled UTXOs.

Proof of Concept

1. Build a valid PSBT with at least one input and at least one output. Make the real user-facing output value higher than the amount that will be declared in the request. For example, make the PSBT output pay `1_500` units.
2. Serialize this PSBT into `req.psbt_bytes`.
3. Set bridge-mode fields so the request reaches the amount check: `evm_tx_hash = 32 bytes`, `evm_event_valid = true`, and `evm_event_finalized = true`.
4. Understate the listener-supplied `psbt_output_amount`; for example set `psbt_output_amount = 900`, `evm_commission = 50`, and `evm_amount = 1000`.
5. Call `validate_psbt_request`.
6. The code checks only `1000 >= 900 + 50` and accepts the request. It never sums `psbt.unsigned_tx.output` and never compares the real PSBT output value (`1_500`) to the stated `psbt_output_amount` (`900`).

Expected result: the enclave should derive the PSBT output amount from the parsed PSBT and compare it to the proven EVM deposit amount. Actual result: the listener can choose the amount used in the check.

Recommendation

We recommend deriving `psbt_output_amount` inside the enclave from the parsed PSBT instead of accepting it from the listener. Concretely:

- **What to derive.** The total value paid to non-change outputs. Compute it in-enclave by iterating `psbt.unsigned_tx.output`, classifying each output as user-facing or change, and summing the user-facing values — the same parsed PSBT the enclave already deserializes in `psbt_crosscheck.rs:33-39`.
- **How to classify change vs user output.** Define an explicit change-output policy, two acceptable forms: (a) the request explicitly marks which output indices are change and the enclave validates that each marked output pays a known bridge-controlled script; or (b) the enclave derives the expected user output script/value from the proven EVM event (see the `SignPsbt` bridge-mode listener-flag finding for how to verify and decode that EVM event in-enclave) and treats every other output as change back to a known bridge script. Any output that is neither a validated user output nor a recognized bridge change script fails closed.
- **Where the source of truth is.** The amount comes only from the parsed PSBT and the change policy — never from a standalone request scalar. The listener-supplied `psbt_output_amount` field should be removed, or kept only as a redundant value that must equal the enclave-derived amount (mismatch fails closed).

Update

Client's response

Fixed (different approach) — the amount is no longer trusted from the listener. The PSBT is now cryptographically bound to the validated RGB consignment, and the credited amount must be covered by that consignment's own validated outputs (PRs #79, #146). The `psbt_output_amount` request field now only carries a redundant value that must equal the consignment-derived amount; it can no longer authorize a larger release on its own, because the consignment — not the field — is the amount authority.

Partially fixed at: `43355713`, `ca2e7590`

Oxorio's response

Remaining issue: the amount authority moved from the listener field to the validated consignment, which closes the original understate-the-field attack, but the enclave still does not prove that the specific authorized asset output (federation or recipient) exactly equals the verified EVM credit. The inflation path deliberately allows a total asset output greater than `net_credited` (rejecting only `asset_output_amount < net_credited`), so an identity-bound PSBT can still carry an excess asset output if the surplus is not classified as legitimate change.

Required remediation: identify and bind each authorized asset output individually. For the inflation shape, require `asset_output_amount == net_credited`, or independently prove which outputs are the recipient output and which are change back to a known bridge script, and fail closed on any surplus. Add an over-mint rejection test.

Reproduction: with a verified EVM credit of `900`, submit an identity-bound PSBT whose asset output is `5000`. The check only rejects `asset_output_amount < net_credited`, so `5000 ≥ 900` passes and the enclave signs a PSBT that mints more than the credit authorizes.

Client's response

Fixed — the remaining issue was the same Inflation over-mint condition tracked separately. PR #181 changes the Inflation amount binding from a one-sided lower bound to exact equality (asset_output_amount == net_credited) and adds an explicit over-mint rejection test. An Inflation output greater than the independently verified EVM credit is therefore rejected.

Fixed at: `ca2e759`, `b37d81a`

W-07	`SignPsbt` returns a successful response even when no inputs were signed in `server.rs`, `keys.rs`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
server.rs	`handle_sign_psbt` returns `SignedPsbtResponse` regardless of `inputs_signed`	498-508
keys.rs	`KeyManager::sign_psbt` returns `signed_count` and permits zero	335-398

Description

In the file `server.rs`, `handle_sign_psbt` forwards the `inputs_signed` count to the caller but does not require it to be non-zero:

```
let (signed_psbt, inputs_signed) = state.sign_psbt(&req.psbt_bytes)?;

Ok(EnclaveResponse {
    response: Some(Response::SignedPsbt(SignedPsbtResponse {
        signed_psbt,
        inputs_signed: inputs_signed as u32,
    })),
})
```

The key manager intentionally returns `Ok((signed_bytes, 0))` when no input belongs to the enclave signer. The unit tests also lock in this behavior:

```
let (_, count) = km.sign_psbt(&psbt_bytes).unwrap;
assert_eq!(count, 0);
```

This is safe if every caller treats `inputs_signed = 0` as a failed signing attempt. However, the enclave API itself returns a normal successful response. A backend or multi-enclave aggregator can therefore confuse "the enclave returned a response" with "the enclave contributed a signature", causing false-positive signing status, stuck flows, or incorrect quorum accounting.

This is primarily an availability and integration-safety issue. It can cause stuck PSBT flows, incorrect quorum accounting, or misleading operational status. In a multi-signer flow, it increases the chance that the backend believes it has collected enough enclave responses while the PSBT still lacks the required signatures.

Proof of Concept

1. Build a syntactically valid PSBT whose inputs do not contain the enclave key in the verified P2WSH witness script or Taproot leaf.
2. Send the PSBT in a `SignPsbtRequest` that passes `validate_psbt_request` (vanilla mode with empty `evm_tx_hash` is enough, or bridge mode with valid listener-supplied fields).
3. `KeyManager::sign_psbt` iterates through the inputs, signs none of them, and returns `signed_count = 0`.
4. `handle_sign_psbt` wraps the PSBT into `SignedPsbtResponse` and returns success to the caller.
5. A caller that checks only RPC success, and not `inputs_signed`, may treat this as a valid signer contribution.

Expected result: a PSBT signing request that signs no inputs should fail at the enclave API boundary, unless there is an explicit "dry-run / inspect" mode. Actual result: the request succeeds with `inputs_signed = 0`.

Recommendation

We recommend returning an error when `inputs_signed == 0` in production signing mode. If a zero-signature result is useful for diagnostics, expose it through a separate explicit inspection endpoint rather than the signing endpoint.

Update

Client's response

Fixed as recommended — signPsbt rejects when `inputs_signed == 0` (PR #98).

Fixed at: `2a9566d5`

W-08	The in-memory SPV header chain grows without an upper bound in `chain.rs`, `server.rs`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
chain.rs	`HeaderChain` stores `headers` / `hashes` with no length cap	70-80
chain.rs	forward extension appends without an upper bound (only reorgs are depth-bounded)	12-25
server.rs	`SubmitHeaders` is reachable repeatedly and feeds the chain	110-130

Description

In the file `chain.rs`, the enclave keeps the validated Bitcoin header chain entirely in memory and appends to it on every accepted batch:

```
pub struct HeaderChain {
    network: Network,
    checkpoint: Checkpoint,
    headers: Vec<Header>,
    hashes: Vec<BlockHash>,
}
```

A `MAX_REORG_DEPTH` bound exists for the reorg case, but the forward-extension case (`start_height == tip + 1`) appends with no upper bound, and there is no maximum chain length or pruning anywhere. Each accepted header costs roughly 80 bytes plus a 32-byte cached hash and is retained for the lifetime of the enclave, which has no persistent storage to spill to. Because `SubmitHeaders` is an unauthenticated, repeatable request from the untrusted listener, the chain length is controlled by the caller.

On Bitcoin mainnet this is hard to exploit: `submit_headers` validates proof-of-work and `nBits` for every header, so an attacker cannot cheaply mint headers. On signet, regtest, or testnet, proof-of-work is trivial, so a hostile listener can generate a very large number of valid low-difficulty headers and push them in `≤ 4 MB` batches (about 52k headers per batch) across repeated `SubmitHeaders` calls until the enclave exhausts memory and is killed, taking down the signing service.

Proof of Concept

1. Run the enclave configured for a low-difficulty network (signet / regtest / testnet).
2. Generate a long run of valid headers extending the current tip; on these networks the proof-of-work target is trivial, so producing them is cheap.
3. Pack the headers into `SubmitHeadersRequest` batches just under the 4 MB framing limit.
4. Send the batches in sequence; each one extends the tip and appends to `headers` / `hashes`.
5. Repeat. The in-memory chain grows linearly with the number of submitted headers, with no cap or pruning.
6. Continue until the enclave's resident memory is exhausted and the process is terminated.

Expected result: the enclave should retain only the headers it needs and reject or prune unbounded growth. Actual result: the chain grows without limit until memory is exhausted.

Recommendation

We recommend bounding the retained header set to the window the enclave actually needs. Concretely, keep only headers at or above `tip - (MAX_REORG_DEPTH + max_confirmations)` and prune everything below that as the tip advances, so memory is constant regardless of how many headers are submitted. In addition, `SubmitHeaders` should be rate-limited with a per-request batch/count cap and — only if an attested caller-authentication mechanism is introduced — bound to that authenticated identity, so an untrusted caller cannot drive header ingestion at will. The checkpoint anchor already lets the enclave discard old headers safely, since confirmation counting never needs the full history.

Update

Client's response

Fixed (different approach) — first bounded by a sliding window (PR #104), then replaced by full retention from the checkpoint with a fixed fail-closed cap, so deep RGB anchors stay verifiable while memory stays bounded (PR #131).

Fixed at: `d2c57d62`

W-09	Ineffective `typesystem` check during `consignment` validation in `rgb.rs`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
rgb.rs	-	263-268

Description

In `rgb.rs#L263-L268`, within the `validate_consignment` function, `trusted_typesystem` is set for subsequent `consignment` validation:

```
let config = ValidationConfig {
    chain_net: self.chain_net,
    trusted_typesystem: transfer.types.clone(),
    build_opouts_dag: true,
    ..Default::default()
};
```

However, the `trusted_typesystem` values are taken from the `consignment` being validated itself. As a result, the types will always be compared against themselves, and this check will always pass successfully.

Recommendation

We recommend explicitly pinning in the enclave a set of allowed trusted `typesystems` to be used for `consignment` validation.

Update

Client's response

Fixed as recommended — the trusted type system is pinned from the canonical `rgb-schemas` crate by schema id; consignment-supplied types are no longer load-bearing (PR #98).

Fixed at: `2a9566d5`

W-10	The attestation certificate-chain verifier omits `BasicConstraints` / `KeyUsage` / `pathLen` checks in `attestation-verify/src/lib.rs`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
lib.rs	`verify_certificate_chain` validates signatures and the pinned AWS root only	—

Description

In the file `attestation-verify/src/lib.rs`, `verify_certificate_chain` validates the certificate signatures and the byte-pinned AWS root but omits the X.509 `BasicConstraints` (CA flag / `pathLen`) and `KeyUsage` checks. Without those constraints a non-CA leaf could in principle be used to sign further certificates in a crafted chain. The verifier is shared by the cloning and attested-public-key flows, so the gap affects both.

Recommendation

We recommend enforcing `BasicConstraints` (CA flag and `pathLen`) and `KeyUsage` at each level of the chain in the shared verifier, and adding crafted-chain rejection tests.

Update

Client's response

Fixed as recommended — BasicConstraints/KeyUsage/pathLen enforced at every level of the chain, plus COSE strictness (PR #126).

Fixed at: `cada7355`

W-11	Header-staleness and request-deadline checks rely on a potentially host-settable enclave clock in `server.rs`, `spv`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
server.rs	request-deadline checks use `SystemTime::now`	—
spv	header-staleness checks use `SystemTime::now`	—

Description

In the file `server.rs` (and the SPV staleness path), the header-staleness and request-deadline gates rely on `SystemTime::now` inside the enclave. If the Nitro enclave wall clock were host-influenced, a malicious host could move time to defeat the staleness gate (accepting a stale header chain) or the deadline gate (reviving an expired request).

Recommendation

We recommend sourcing time for security-relevant checks from an attestable / monotonic source, or documenting the platform guarantee that makes `SystemTime::now` safe here, so the assumption behind the staleness and deadline checks is explicit.

Related platform-trust assumption — randomness. The same "is this input really enclave-controlled?" question applies to the entropy used for key/seed and nonce generation: it draws from `getrandom(2)` / `OsRng` *inside* the enclave (`server.rs:177`, `keys.rs::generate`, `cloning.rs`), which on AWS Nitro is seeded by the NSM hardware RNG rather than the untrusted parent. Subverting it would make every derived key predictable, but that requires breaking the NSM/Nitro platform, which is already inside the trust base the design assumes. We recommend documenting this assumption explicitly and asserting that the entropy path stays NSM-backed (no fallback to a weak or host-influenced PRNG), so the dependency is auditable rather than implicit.

Update

Client's response

Fixed as recommended — via the recommendation's documentation option: the wall-clock and entropy trust model (hypervisor kvm-clock, the three consuming gates, the NSM-attested-time upgrade path) is documented in the repository's trust-model specification (PR #145).

Fixed at: `0beefb68`

W-12	The cloning replay guard is count-capped and rejects new handshakes when full, enabling an availability DoS in `state.rs`, `server.rs`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
state.rs	`check_and_record` caps the guard at 10k and hard-rejects new entries on overflow	67-82
server.rs	`handle_get_clone` consumes replay-guard capacity for each accepted attestation nonce	802
server.rs	`handle_set_clone` consumes replay-guard capacity for each accepted attestation nonce	875

Description

In the file `state.rs`, the cloning replay guard (`NonceReplayGuard`) is bounded by a fixed entry count (capped at 10k) and hard-rejects new handshakes when full. Because cloning is parent-initiated, a misbehaving parent or any caller able to reach the cloning RPC through the parent can flood the guard with distinct handshakes to exhaust the cap and permanently block legitimate cloning — a cloning-availability DoS (no funds or secret leak). The concrete unauthenticated exploit path that makes this cheap to trigger is described in the companion finding (record-before-authentication + arbitrary-nonce attestation oracle).

Recommendation

We recommend bounding the guard by time (evicting stale entries) or using a sliding window rather than a pure count cap, so a flood cannot permanently wedge cloning, and testing the exhaustion path.

Update

Client's response

Fixed as recommended — the count cap was replaced by TTL + oldest-first eviction with regression tests (PR #80).

Fixed at: `cea1dcbd`

W-13	The replay guard records nonces before authenticity checks, and `handle_get_attested_public_key` is an arbitrary-nonce attestation oracle, enabling a secret-less cloning DoS in `server.rs`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
server.rs	`handle_get_attested_public_key` issues a valid NSM attestation over any caller-supplied `nonce`	303-323
attestation.rs	`verify_peer_attestation(..., None)` verifies the document, PCRs, nonce presence, and public-key presence, but does not match the nonce to an expected caller value	92-107
server.rs	`handle_get_clone` records the nonce (L802) before the pubkey/digest/donor-secret checks (L807-L822)	796-827
server.rs	`handle_set_clone` records the nonce (L875) before the pubkey binding / seed decrypt	864-875

Description

The count cap described in the companion finding (count-capped replay guard) can be exhausted by an attacker that holds **no** cloning secret and is any caller able to reach the cloning RPC through the parent, because of two code properties:

- `handle_get_attested_public_key` mints a valid NSM attestation over **any** caller-supplied `nonce` (`server.rs`), and `verify_peer_attestation(..., None)` verifies the document, PCRs, nonce presence, and public-key presence, but it does not compare the nonce to an expected caller-generated value. Freshness is delegated to the replay guard. An attacker can therefore obtain an unlimited number of valid attestations carrying distinct nonces from the donor's *own* endpoint (its attestations carry the donor's PCRs, which is exactly what the donor compares against).
- In `handle_get_clone`, `state.replay_guard.check_and_record(nonce)` runs **before** the requester-pubkey binding, digest binding, and donor-secret HMAC checks. In `handle_set_clone`, it runs **before** the donor-pubkey binding and seed / identity checks. So a handshake that will ultimately be rejected (wrong pubkey, wrong digest, no knowledge of the cloning secret, or invalid encrypted seed) still records its nonce first.

Combined, the attacker calls `get_attested_public_key(nonce_i)` for many distinct nonces, feeds each attestation into `get_clone` / `set_clone`, and each request records its nonce and then fails the later checks. After 10k such bogus handshakes the guard is full and every subsequent legitimate cloning handshake fails at `check_and_record` with "replay guard full; refusing new attestations" until the enclave is restarted.

Recommendation

We recommend recording the nonce only **after** the authenticity and binding checks pass — in `handle_get_clone`, move `check_and_record` below the pubkey, digest, and donor-secret verification; in `handle_set_clone`, move it below the donor-pubkey binding and seed / identity checks (or record provisionally and roll back on failure) — so that unauthenticated or otherwise rejected handshakes never consume guard capacity. Combine this with the time-based / sliding-window eviction recommended in the companion count-cap finding, and add a test asserting that invalid attestations, wrong pubkeys, wrong digests, and failed seed / identity checks do not consume replay capacity.

Update

Client's response

Fixed as recommended — nonces are recorded only after all authenticity and binding checks in both cloning handlers (PR #129).

Partially fixed at: `35dabe2f`

Oxorio's response

Remaining issue: the attacker-driven secret-less DoS is closed, and one narrow self-inflicted edge remains. Both cloning handlers now record the nonce only after the peer attestation chain and PCRs are verified, and `handle_set_clone` runs the `Cloning → Active` transition (`complete_cloning`) after the replay check, not before — so an unauthenticated caller can no longer pre-consume nonces, and a rejected replay no longer mutates state. `handle_get_attested_public_key` still attests any caller-supplied nonce, but it no longer feeds the replay guard, so it can no longer be used to exhaust cloning capacity. The residual is that in `handle_set_clone` the nonce is recorded before `complete_cloning` and is not rolled back if that step fails: the seed decrypt, `KeyManager` derivation, or identity check can fail after the nonce is already consumed. A legitimate party whose handshake fails there cannot retry with the same donor attestation — the reused nonce is rejected as a replay — and must obtain a fresh donor attestation carrying a new nonce.

Required remediation: reserve the nonce provisionally and roll it back on failure, or record it inside the same critical section that commits `complete_cloning`, so a failed completion leaves no consumed nonce. Add a test asserting that a `SetClone` which fails at seed-

decrypt or identity leaves both the enclave state unchanged and the nonce un-consumed, so an immediate retry with the same attestation still succeeds.

Reproduction: submit a `SetClone` with a valid, freshly attested donor nonce but an encrypted seed that fails the identity check (`km.evm_address() != session.cluster_public_key`). The nonce is recorded, then `complete_cloning` returns `IdentityMismatch`; the enclave state correctly stays in `Cloning`, but the nonce is now consumed, so resubmitting the corrected handshake with the same donor attestation is rejected as a replay and only succeeds once a fresh donor attestation (new nonce) is issued.

Client's response

Fixed as recommended — in `handle_set_clone`, the donor nonce is recorded only after `complete_cloning` succeeds. Seed decryption, `KeyManager` derivation, identity verification, and the `Cloning → Active` transition therefore complete before the nonce is consumed.

If any of these steps fail, `complete_cloning` returns an error, the enclave remains in `Cloning`, and the nonce is not recorded. The same donor attestation can therefore be retried without being rejected as a replay.

PR #177 adds the requested regression test: a `SetClone` is intentionally failed during seed decryption, the state is verified to remain unchanged, and retrying with the **same donor attestation / same nonce** succeeds.

This closes the remaining self-inflicted replay/availability edge described in the re-audit.

Fixed at: `35dabe2`, `043856b`

W-14	The mainnet SPV checkpoint is not aligned to a retarget boundary, wedging the chain at the first boundary above it in `spv/checkpoint.rs`, `spv/chain.rs`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
checkpoint.rs	the pinned mainnet checkpoint height is not a multiple of `2016`	—
chain.rs	the non-PoW short-circuit only covers signet/regtest	236-240

Description

In the file `spv/checkpoint.rs`, the pinned mainnet checkpoint sits at a height that is not aligned to a difficulty-retarget boundary (not a multiple of `2016`). The non-PoW short-circuit in `spv/chain.rs` only protects signet/regtest, so on mainnet the retarget-difficulty lookup will wedge the chain at the first retarget boundary above the checkpoint once the listener syncs that high — a latent liveness break for the SPV header chain.

Recommendation

We recommend replacing the mainnet checkpoint with a boundary-aligned height (the next multiple of `2016`, or a higher already-finalized boundary) carrying its real `(hash, bits, time)`, and asserting the boundary-alignment invariant at checkpoint-load time so a future non-aligned checkpoint fails fast before mainnet sync reaches the boundary.

Update

Client's response

Fixed as recommended — the mainnet checkpoint is re-aligned to a retarget boundary (PR #104).

Fixed at: `76c62569`

3.4 INFO

I-01	`SignRawMessage` signs arbitrary bytes with no domain / version / nonce in `enclave/src/server.rs`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
server.rs	`handle_sign_raw_message` (Active in dispatch; dispatch table)	510-541
server.rs	inline EIP-191 prefix + `state.sign_evm` (main `evm_secret` key)	518-529

Description

In the file `enclave/src/server.rs`, `handle_sign_raw_message` validates only that `message` is non-empty and always applies the EIP-191 prefix. It is safe from EVM tx-replay (EIP-191 ≠ tx encoding), but with no domain/version/nonce in the payload a weak backend protocol could enable application-level replay/misuse. The path may also be dead surface (the "fundsIn authorization" route is not exercised on-chain).

Verified: the handler is at `server.rs`, **Active** in the dispatch table (alongside `SignEvm`/`SignPsbt`/`SignRawDigest`), and signs with the **main** `evm_secret` **key** (`m/44'/60'/0'/0/0` — the same key that signs `fundsOut` EIP-712 digests, *not* the isolated gas key used by `SignRawDigest`). An EIP-191 digest begins `0x19 'E'` (`0x19 0x45`), an EIP-712 `fundsOut` digest begins `0x19 0x01`, and any tx envelope is legacy-RLP (`≥ 0xc0`) or typed (`0x01..0x7f`) — none can alias, so a raw-message signature can never be replayed as a `fundsOut` authorization or a transaction. The residual is purely application-level, and its intended consumer — an off-chain `fundsIn` authorization — is not wired on-chain (there is no `fundsIn` handler), so the path also looks like dead surface.

```
// enclave/src/server.rs — handle_sign_raw_message
if req.message.is_empty() { return Err(...); } // only validation
// EIP-191: keccak256("\x19Ethereum Signed Message:\n" || len || message)
let mut hasher = Keccak256::new();
hasher.update(b"\x19Ethereum Signed Message:\n");
```

```
hasher.update(req.message.len().to_string().as_bytes());
hasher.update(&req.message);
let hash: [u8; 32] = hasher.finalize().into();
let signature = state.sign_evm(&hash)?; // MAIN fundsOut key, not the gas key
```

Recommendation

We recommend first determining whether `handle_sign_raw_message` is reachable in production. If dead, remove the handler and its `proto/enclave.proto` dispatch entry so the enclave exposes no arbitrary-bytes signing surface. If live, wrap `message` in a domain-separated schema hashed inside the enclave (a fixed domain/version tag, the operation type, and a monotonic per-purpose nonce) and reject anything that does not parse.

Update

Client's response

Acknowledged (deferred) — the handler signs EIP-191-prefixed messages only, structurally unusable as a transaction or bridge-operation signature, and has no known consumer; removal (preferred) vs adding a domain/nonce schema is a pending product decision (#124).

Oxorio's response

The risk remains and is accepted. The endpoint still signs arbitrary non-empty bytes with the main EVM key under an EIP-191 prefix, with no application domain, version, or nonce, so an application-level replay or misuse is still possible for any backend protocol that relies on it. The compensating controls are strong: an EIP-191 digest cannot alias an EIP-712 `fundsOut` digest or any transaction envelope, so a raw-message signature can never be replayed as a bridge authorization or a transaction, and the endpoint has no known production consumer. Before production, the enclave should either remove this handler entirely (preferred, since it is unused) or, if it must stay, wrap the message in a fixed domain/version/operation/nonce schema hashed inside the enclave and reject anything that does not parse.

I-02	`dev-mode` panic-path on short calldata in `enclave/src/server.rs`
Severity	**INFO**
Status	• FIXED

> INFO (availability / test-hardening; not a prod theft risk).

Location

File	Location	Line
server.rs	`handle_sign_evm`	—
evm.rs	`sign_request_digest` uses `assert!(len >= 4)`	—

Description

In the file `enclave/src/server.rs`, `dev-mode` skips `validate_evm_request` (which includes `call_data.len >= 4`), while in the file `enclave/src/signing/evm.rs`, `sign_request_digest` still uses `assert!(len >= 4)`. A short calldata therefore panics the handler in dev/test builds instead of returning a clean error.

```
// enclave/src/signing/evm.rs — sign_request_digest
assert!(call_data.len >= 4); // panics in dev-mode (validate_evm_request is skipped)
```

Recommendation

We recommend replacing the `assert!` in `sign_request_digest` with a fallible check that returns an error, and moving the `call_data.len >= 4` guard out of the `dev-mode`-gated path so it runs on every build.

Update

Client's response

Fixed as recommended — `sign_request_digest` is fallible; short calldata returns an error in every build instead of aborting the enclave (PR #145).

Fixed at: `0beefb68`

I-03	Blocking Esplora HTTP on the signing path through the untrusted host in `enclave/src/validation/rgb.rs`
Severity	**INFO**
Status	• ACKNOWLEDGED

> INFO (availability). Adjacent to the DoS-ordering finding (expensive RGB validation before the cheap hash check).

Location

File	Location	Line
rgb.rs	`esplora_blocking` + `transfer.validate`	251-279
vsock_forwarder.rs	Esplora reached through the untrusted host	—

Description

In the file `enclave/src/validation/rgb.rs`, RGB validation issues **blocking** HTTP requests to Esplora (`esplora_blocking` + `transfer.validate`), reached through the untrusted host (`enclave/src/vsock_forwarder.rs`), on the synchronous signing path. A slow or unresponsive host — or one deliberately stalling responses — can stall sign-request handling. This is distinct from the DoS-ordering finding (which is about ordering); here it is the blocking network call itself, with no explicit timeout/isolation.

```
// enclave/src/validation/rgb.rs
let resolver = AnyResolver::esplora_blocking(Builder::new(esplora_url))?; // no timeout
transfer.validate(&resolver, &config) // blocking HTTP via vsock→host
```

Recommendation

We recommend setting an explicit connect/read timeout on the `esplora_blocking` client in `validate_consignment` so a stalling host fails fast, and running the resolver off the synchronous signing path (pairs with the DoS-ordering finding).

Update

Client's response

Fixed as recommended — the blocking Esplora calls on the signing path are bounded by a 30s compile-time timeout (PR #145).

Partially fixed at: `0beefb68`

Oxorio's response

Remaining issue: a compile-time timeout was added, which stops a single stalled request from blocking forever, but validation still performs a synchronous network round-trip through the untrusted host directly on the signing thread. The host can repeatedly delay every request up to the timeout and, on the serial/bounded signing path, tie up the available workers.

Required remediation: use a typed Esplora client with a pinned destination and limits on connection, response size, and total request count, and fetch the required network data ahead of time and asynchronously — or otherwise isolate it from the signing thread pool — so the signing path no longer performs blocking host I/O inline.

Reproduction: a hostile host answers each Esplora query only just before the 30-second timeout expires. Because the resolver runs inline on the signing path, every in-flight sign request is stalled for up to that timeout, and repeated slow responses keep the bounded worker pool saturated, degrading availability of the signing service. The fix bounds each call but does not remove the inline blocking dependency on the untrusted host.

Client's response

Partially fixed / risk substantially bounded — PR #182 replaces the generic Esplora forwarder with a dedicated typed and destination-pinned client.

The production client:

- connects only to the boot-pinned parent vsock destination;
- exposes only the specific Esplora endpoints required by RGB validation;
- applies an absolute connect/read deadline to every request;
- performs no automatic retries/backoff;
- enforces a maximum response size; and
- parses bounded HTTP responses explicitly.

This removes the previous amplification case where a malicious host could combine the blocking timeout with `esplora-client` retries and keep a request occupied significantly longer than the configured timeout. A single Esplora fetch is now strictly bounded and fails closed.

The remaining architectural point is that RGB validation still performs bounded synchronous Esplora I/O during consignment validation. PR #182 therefore closes the unbounded/retry-amplified host-I/O issue and implements the requested typed/pinned and response-limited transport, but does not by itself move Esplora fetching completely off the signing execution path.

I-04	Per-field input sizes are bounded only by generic framing, enabling CPU/memory amplification in `spv_crosscheck.rs`, `evm_crosscheck.rs`, `framing.rs`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
framing.rs	only a single global `MAX_MESSAGE_SIZE` bounds the whole request	—
spv_crosscheck.rs	`merkle_path` depth and the number of `merkle_proofs` are unbounded	—
evm_crosscheck.rs	`calldata` and `proof` lengths have no semantic per-field cap	—

Description

In the file `framing.rs`, request size is limited only by a single global `MAX_MESSAGE_SIZE`; there are no semantic, per-field caps. As a result several inputs that drive expensive in-enclave computation are effectively unbounded: in `spv_crosscheck.rs` the `merkle_path` depth and the number of `merkle_proofs` can be made very large, and in `evm_crosscheck.rs` the `calldata` and `proof` lengths are bounded only by the global frame size. Because the server processes requests serially, a single crafted request with a deep Merkle path or many proofs forces a large amount of hashing/parsing work before any cheap check can reject it, amplifying CPU (and memory) cost out of proportion to the request and degrading availability for legitimate requests.

Recommendation

We recommend adding semantic, per-field size limits that reflect the protocol's real bounds — a maximum `merkle_path` depth, a maximum number of `merkle_proofs`, and maximum `calldata` / `proof` lengths — and validating them before any expensive parsing or hashing. The cheapest checks should run first so an oversized or malformed request is rejected before the enclave commits CPU to it.

Update

Client's response

Fixed as recommended — semantic per-field caps: Merkle path depth ≤ 32, fundsOut calldata ≤ 64 KiB, enforced before any expensive work (PR #118).

Partially fixed at: `ef9aca2d`

Oxorio's response

Remaining issue: the added per-field caps (Merkle path depth ≤ 32, `fundsOut` calldata ≤ 64 KiB) bound individual fields, but the number of proofs, their aggregate byte size, the RGB consignment size, and the remaining dynamic proof/settlement fields are still unbounded. A request can therefore stay under every individual cap while still forcing a disproportionate amount of aggregate parsing and hashing on the serial signing path.

Required remediation: before any expensive parsing, bound the number of proofs, the total encoded proof bytes, the RGB consignment size, and the settlement-data size, and enforce an overall per-request compute budget; add boundary and rejection tests for these aggregate limits.

Reproduction: craft a request that respects the per-field caps (each Merkle path ≤ 32 deep, calldata ≤ 64 KiB) but carries a very large number of proofs and/or an oversized consignment. Because there is no aggregate cap, the enclave performs multiplied hashing/parsing work before any cheap check can reject it, amplifying CPU and memory cost and degrading availability.

Client's response

Fixed as recommended
The generic 4 MB wire-frame cap and the existing per-field caps left the aggregate parse/hash budget unbounded: a request could stay under every per-field limit yet still force a large rgbstd consignment parse or proof-verification loop on the serial signing path. Enforce aggregate caps before any expensive parsing:

consignment size (1 MiB), on both the source and send-RGB destination paths;
number of Merkle proofs (256);
aggregate txid + Merkle-path bytes (128 KiB), bounding hashing work independently of the per-proof depth cap.
Real swap consignments are a few KB, so the caps sit far above legitimate traffic. Adds boundary and rejection tests.

Fixed at: `9163d10`

I-05 COSE verifier accepts DER signatures and does not check `alg == ES384` in `attestation-verify/src/lib.rs`

Severity	**INFO**
Status	• FIXED

> INFO (spec-deviation / hardening; **not exploitable** — the key is pinned to P-384).

Location

File	Location	Line
lib.rs	`parse_cose_ecdsa_signature`	360-372
lib.rs	`verify_real_document`	258-288

Description

In the file `attestation-verify/src/lib.rs`, `parse_cose_ecdsa_signature` accepts both raw `r||s` (96 bytes) and DER-encoded signatures, and `verify_real_document` does not validate the `alg` field of the protected header against `ES384`. RFC 8152 mandates raw `r||s`. There is no exploit: the verifier hardcodes a `p384` `VerifyingKey` / `Signature`, and the payload is bound by AWS's COSE signature, so DER malleability only yields a different encoding of the *same* document. This is pure strictness/hardening.

```
// attestation-verify/src/lib.rs
// parse_cose_ecdsa_signature: accepts raw 96-byte r||s OR DER
// verify_real_document : protected-header `alg` not compared to ES384 (-35)
```

Recommendation

We recommend accepting only the raw **96-byte** `r||s` (P-384 = 48+48) via `p384::ecdsa::Signature::from_slice` and dropping the DER branch (`parse_cose_ecdsa_signature`), and asserting the protected-header `alg` equals **ES384 = -35** (the IANA COSE value, RFC 9053 §2.1; `coset::iana::Algorithm::ES384`) in `verify_real_document`. The `p384` and `coset` crates needed for both are already in the dependency tree.

Update

Client's response

Fixed as recommended — COSE alg pinned to ES384; only raw 96-byte r||s signatures accepted, DER rejected (PR #126).

Fixed at: `cada7355`

I-06	Amount-binding is non-strict (`≤`) rather than the spec's strict `==` in `evm_crosscheck.rs`
Severity	**INFO**
Status	• ACKNOWLEDGED

INFO/WARN.

Location

File	Location	Line
evm_crosscheck.rs	`burned < calldata_amount` rejected, i.e. `≤` allowed	290

Description

In the file `enclave/src/validation/evm_crosscheck.rs` the cross-check rejects only `burned < calldata_amount`, i.e. it permits `calldata_amount ≤ burned`. The spec requires a strict equality: "the burn amount equals the amount requested for unlock". The non-strict comparison enables under-release / over-burn and, combined with the missing RGB `OpId` binding (the operation-identifier uniqueness finding), amplifies replay.

```
// enclave/src/validation/evm_crosscheck.rs
if burned < calldata_amount { reject } // only '<' rejected → '<=' allowed
```

Recommendation

We recommend tightening the check at `evm_crosscheck.rs` from `burned < calldata_amount` to `burned != calldata_amount` so only strict equality passes, landing it together with the RGB `OpId` binding (the operation-identifier uniqueness finding). For the pools (`TS_TRANSFER`) shape, the enclave already requires `last.total_output_amount` to be at least `calldata_amount + calldata_commission`, but this is only a coarse total-output binding. Tighten it by identifying the output(s) paying to the federation's multisig seal in the consignment's last transition and requiring that federation-bound sum to cover the calldata amount.

Update

Client's response

Acknowledged (deferred) — strict `==` needs a design decision per leg: the transfer leg must sum only the outputs paying the federation multisig seal (seal source of truth undecided, #58), and the burn leg needs commission semantics settled. The current `≤` bind with enclave-read amounts already prevents over-release (#124).

Oxorio's response

The risk remains and is accepted. The check still permits `calldata_amount ≤ burned`, so over-burn / under-release and a surplus output are not strictly excluded; strict equality is deferred because each leg needs a design decision first (the transfer leg must sum only the outputs paying the federation multisig seal, whose source of truth is undecided, and the burn leg needs its commission semantics settled). The compensating control is that the amount is read by the enclave from validated data and the `≤` bind already prevents over-release. Before production, any surplus must be independently proven to be authorized change, and processes that depend on an exact amount must not rely on a "no more than" check; the target fix is `burned != calldata_amount` combined with per-output federation-seal binding.

I-07	BIP-86 account extended private keys are not zeroize-on-drop in `enclave/src/keys.rs`
Severity	**INFO**
Status	• ACKNOWLEDGED

> INFO / defense-in-depth.

Location

File	Location	Line
keys.rs	struct `KeyManager` fields `account_xpriv_vanilla` / `account_xpriv_colored`	55, 57
keys.rs	local `master: Xpriv` in `from_seed`	99

Description

> **What "zeroize" means.** Zeroizing a secret means overwriting its bytes in memory (with zeros) the moment the value is dropped, so the key material does not linger in freed heap/stack memory where a later memory-disclosure bug, a core dump, or allocator reuse could expose it. In Rust `secrecy::SecretBox` and `zeroize::Zeroizing` do this automatically on `Drop`; a plain field is just freed, leaving its bytes intact until something else overwrites them.

In the file `enclave/src/keys.rs`, `KeyManager` is careful to wrap the raw secrets — `seed`, `evm_secret`, `evm_gas_tx_secret`, `btc_secret` — in `secrecy::SecretBox`, which zeroizes on drop:

```
pub struct KeyManager {
    seed: SecretBox<[u8; 64]>,
    evm_secret: SecretBox<[u8; 32]>,
    evm_gas_tx_secret: SecretBox<[u8; 32]>,
    btc_secret: SecretBox<[u8; 32]>,
```

But the two BIP-86 account extended private keys are stored as plain fields:

```
    account_xpriv_vanilla: Xpriv,
    account_xpub_vanilla: Xpub,
    account_xpriv_colored: Xpriv,
    account_xpub_colored: Xpub,
```

`bitcoin::bip32::Xpriv` (chain code + `secp256k1::SecretKey`) does **not** implement `ZeroizeOnDrop`. These two account keys can re-derive every child signing key for the Taproot/RGB spend paths and live for the entire enclave session. The function-local `master Xpriv` in `from_seed` (line 99), which is seed-equivalent, is likewise dropped without an explicit wipe. On drop, neither is wiped by an approved API, so residual key bytes persist in heap/stack memory until the allocator reuses them.

It is distinct from the existing (`cloning_secret` not committed into PCR2), which concerns a different object. The enclave memory is inside the AWS Nitro attested boundary (the primary confidentiality control); confidence is `likely` at source level.

Recommendation

We recommend bringing the two account xprivs under the same zeroize discipline already applied to `seed` / `evm_secret` / `btc_secret` (both wrapper types — `secrecy::SecretBox` and `zeroize::Zeroizing` — are already in the dependency tree). Since `Xpriv` does **not** implement `Zeroize` (so `Zeroizing<Xpriv>` will not compile), either store the raw 32-byte private key + chain code in `SecretBox<[u8; 32]>` and rebuild a transient `Xpriv` on demand, or add an `impl Drop for KeyManager` that calls `non_secure_erase` on both keys' `private_key` and overwrites the chain-code bytes:

```
impl Drop for KeyManager {
    fn drop(&mut self) {
        self.account_xpriv_vanilla.private_key.non_secure_erase(); // secp256k1 API
        self.account_xpriv_colored.private_key.non_secure_erase();
        // also overwrite the chain-code bytes
    }
}
```

Apply the same `non_secure_erase` to the local `master` in `from_seed` right after the account keys are derived. The AWS Nitro attested memory boundary remains the primary confidentiality control; this is defense-in-depth against residual key bytes lingering until the allocator reuses them.

Update

Client's response

Acknowledged (deferred) — zeroizing the derived xprivs while the master seed stays resident (required by the cloning protocol) would not shrink the real key-material lifetime; scheduled with the seed-residency work (I-13, #124).

Oxorio's response

The risk remains and is accepted. The two BIP-86 account xprivs and the function-local `master` remain plain `Xpriv` fields without zeroize-on-drop, so their bytes can linger in freed memory. The client's point is valid: while the master seed itself must stay resident for the cloning protocol (tracked as the seed-residency finding), wiping only the derived xprivs would not meaningfully shrink the exposure window. The compensating control is the AWS Nitro attested memory boundary — the primary confidentiality control — and the fact that the raw seed and the main derived secrets already use `SecretBox`. Before production, key-material residency should be minimized as part of the seed-residency redesign, bringing the account xprivs and `master` under an explicit zeroize discipline at the same time.

I-08	The Esplora TCP forwarder exposes a generic local egress path inside the enclave in `main.rs`, `vsock_forwarder.rs`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
main.rs	starts local Esplora forwarder on port `3443`	66-81
vsock_forwarder.rs	binds `127.0.0.1:{local_port}` and forwards to host vsock	17-95

Description

In the files `main.rs` and `vsock_forwarder.rs`, production builds with `vsock` on Linux start a local TCP listener and forward every connection to the parent instance over vsock:

```
// main.rs
if let Err(e) = utexo_bridge_enclave::vsock_forwarder::start_forwarder(3443, vsock_port) {
    tracing::error!("failed to start vsock forwarder: {e}");
}

// vsock_forwarder.rs
let listener = TcpListener::bind(format!("127.0.0.1:{local_port}"))?;
...
let vsock = match VsockStream::connect_with_cid_port(PARENT_CID, vsock_port) {
    Ok(s) => s,
    Err(e) => { /* logged, connection skipped */ continue; }
};
```

This is intended for Esplora access and binds only to loopback, so it is not directly reachable from outside the enclave. However, it is still a generic egress primitive for any code running inside the enclave process boundary. If a future dependency bug, plugin, or additional process can open a local TCP connection, it can reach the host-controlled Esplora forwarder.

This does not by itself bypass validation. It increases the blast radius of an in-enclave bug by giving code inside the enclave a reusable host egress path. The security impact is mainly defense-in-depth and boundary clarity.

Proof of Concept

1. Build/run the enclave with `vsock` on Linux so the production forwarder path is enabled.
2. Start the enclave. In `main.rs`, it starts `vsock_forwarder::start_forwarder(3443, vsock_port)`.
3. From code running inside the enclave process boundary, open a TCP connection to `127.0.0.1:3443`.
4. Send bytes over that connection.
5. The forwarder accepts the local TCP stream and opens a vsock connection to the parent instance (`CID 3`, configured `vsock_port`).
6. The host-side `vsock-proxy` then relays those bytes to the external Esplora endpoint.

Expected result: only the RGB validator should be able to perform the specific Esplora queries needed for validation. Actual result: the forwarder is a generic local TCP egress path for any code that can connect to loopback inside the enclave.

Recommendation

We recommend narrowing the forwarder from a generic egress primitive to a typed, purpose-built client. Concretely:

- **What to expose.** Replace the loopback `TcpListener` that forwards every connection (`vsock_forwarder.rs:17-95`) with a small typed Esplora client that offers only the specific calls the RGB resolver actually makes (e.g. fetch transaction, fetch Merkle proof, fetch tip) and constructs the requests itself, so arbitrary host-bound traffic cannot be tunneled through it.
- **How to constrain it.** Keep the client private to the RGB resolver code path rather than binding a process-wide port; if a socket is unavoidable, restrict accepted requests to an allowlist of Esplora method + path shapes and reject anything else.
- **Trust-boundary note.** Document explicitly that `127.0.0.1:3443` and everything reachable through it is a host-controlled, untrusted boundary — data returned over it is evidence to be verified (SPV / consignment validation), never trusted input.

Update

Client's response

Fixed — the forwarder is documented as an untrusted, host-controlled egress boundary (PR #127), its destination is boot-pinned and never request-controlled, and the Esplora calls it carries are timeout-bounded (PR #145).

Partially fixed at: `a2d6a0e1`, `0beefb68`

Oxorio's response

Remaining issue: the trust boundary is now clearly documented and the forwarder's destination is boot-pinned rather than request-selected, but the loopback TCP forwarder

itself remains a generic, process-wide egress primitive. The documentation constrains neither the accepted protocol shapes nor the returned data for arbitrary code running inside the enclave process, so the reusable host egress path still exists.

Required remediation: replace the generic forwarder with a small typed, destination-pinned Esplora client that exposes only the specific calls the RGB resolver makes, keep it private to that code path rather than binding a process-wide port, include the destination/protocol policy in attestation, and forbid arbitrary destination or protocol selection in production.

Reproduction: with the production `vsock` build running, any code inside the enclave process boundary opens a TCP connection to `127.0.0.1:3443` and sends bytes; the forwarder relays them over vsock to the host, which proxies them out. The fix documents that this is an untrusted boundary but does not remove the generic egress path, so a future dependency bug or additional in-process code still gets a reusable host-bound channel. This is the same PoC as the original finding.

Client's response

Fixed as recommended — PR #182 removes the generic Esplora loopback forwarder and replaces it with a typed, destination-pinned client owned by the RGB validator.

In production, the enclave no longer binds `127.0.0.1:3443` for Esplora access. The RGB validator connects directly to the parent over the fixed Nitro parent CID and exposes only the specific Esplora operations required for validation:

- `/block-height/0`
- `/tx/{txid}/raw`
- `/tx/{txid}/status`
- `/fee-estimates`

There is no public arbitrary-path API, so other in-enclave code can no longer open the former loopback port and tunnel arbitrary host-bound traffic through the Esplora path.

The client also enforces bounded responses, absolute request deadlines, and no automatic retries. The protocol restrictions and fixed production transport are part of the measured enclave code and are therefore covered by the enclave's PCR attestation.

The host-side Esplora endpoint remains an explicitly untrusted relay; data returned through it is treated only as evidence and is independently validated through the enclave's RGB/SPV validation path.

This removes the generic process-wide Esplora egress primitive described in the finding and closes the original reproduction.

Fixed at: `a2d6a0e`, `0beefb6`, `7ad12f7`

I-09	`consignment_hash` is an integrity check, not authorization, in `evm_crosscheck.rs`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
evm_crosscheck.rs	`keccak256(consignment) == consignment_hash` check	115-131

Description

In the file `evm_crosscheck.rs`, the function `validate_evm_request` checks that the request's raw consignment bytes match the request's `consignment_hash`:

```
let computed = Keccak256::digest(&req.consignment);
if &computed[..] != req.consignment_hash.as_slice {
    return Err(EnclaveError::CrossCheck(...));
}
```

This is useful tamper detection for the wire copy, but it is not an authorization predicate by itself. A listener that controls the request can choose both `consignment` and `consignment_hash`. The security boundary must therefore be the independent RGB validation, SPV anchoring, and binding of validated facts (`contract_id`, amount, recipient, operation id) to the EVM authorization.

The check is not harmful, but it can be misread as proving more than integrity. Treating `consignment_hash` as authorization would obscure the real required controls.

Recommendation

We recommend documenting the check as integrity-only and keeping authorization language tied to RGB validation, SPV, and typed intent binding. If the hash is needed for external audit trails, bind it into a canonical signed bridge intent together with the validated operation fields.

Update

Client's response

Fixed as recommended — both check sites document INTEGRITY, NOT AUTHORIZATION, with the actual authorization chain spelled out (PR #127).

Fixed at: `a2d6a0e1`

I-10	Selector-specific EVM validators rely on caller ordering instead of failing closed in `evm_crosscheck.rs`, `server.rs`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
server.rs	calls both selector-specific validators after the main EVM request check	402-416
evm_crosscheck.rs	`validate_funds_out_burn` no-op for non-mintburn selectors	267-269
evm_crosscheck.rs	`validate_funds_out_transfer` no-op for non-legacy selectors	328-330

Description

In the files `evm_crosscheck.rs` and `server.rs`, the functions `validate_funds_out_burn` and `validate_funds_out_transfer` are safe in the current call graph because `handle_sign_evm` first runs the main request validation and then calls both selector-specific checks. Each specific function returns `Ok()` for selectors that do not belong to it.

This is a hidden ordering invariant: the safety of these functions depends on callers always running the selector whitelist first. A future refactor that calls a selector-specific validator directly could receive a silent success for an unsupported selector.

No current bypass was identified in the audited call graph. The concern is refactor safety and making validation APIs fail-closed by construction.

Recommendation

We recommend representing validated calldata as a typed enum after the selector whitelist, for example `FundsOutCall::MintBurn` / `FundsOutCall::PoolsLegacy`, and passing that typed value to the selector-specific checks. Alternatively, make the selector-specific functions return an explicit "wrong selector" error unless the caller passes a typed dispatch context.

Update

Client's response

Fixed as recommended — selector-specific validators fail closed on a non-`fundsOut` selector instead of silently passing (PR #127).

Fixed at: `a2d6a0e1`

I-11	Cross-chain metadata is request/calldata-supplied rather than derived from enclave-validated facts in `evm_crosscheck.rs`, `server.rs`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
evm_crosscheck.rs	mint/burn calldata layout includes `sourceChainId` / `destinationChainId`	13-25
evm_crosscheck.rs	amount-only mint/burn binding	263-297
server.rs	SPV checks `validated.chain_net` separately from calldata metadata	422-458

Description

In the files `evm_crosscheck.rs` and `server.rs`, the mint/burn `fundsOut` calldata contains cross-chain metadata such as `sourceChainId` and `destinationChainId`, but the enclave's EVM cross-check primarily binds amount, selector, domain chain, proxy contract, consignment validity, and SPV inclusion. It does not build a single typed object that derives the calldata chain metadata from `validated.chain_net` and the pinned EVM domain.

This is related to the previously reported identifier-decoupling class: the request contains several identifiers and chain labels, but the enclave does not canonicalize them into one bridge intent before signing.

It is mainly a specification and maintainability gap, but it can amplify higher-severity binding mistakes when combined with asset, recipient, or operation-id decoupling.

Two fields deserve a specific note. The `sourceAddress` (originating EVM sender) is not cross-checked today and, like the `recipient`, cannot be validated from the calldata alone — it would have to be carried in the burn-transition metadata of the consignment and read by the enclave (the same mechanism proposed for the recipient-binding finding), since the validated consignment currently exposes no EVM address. A full cryptographic *proof* of the source (e.g. proving which EVM account emitted the originating event) is harder and is out of scope for this finding; the practical, in-model improvement is to bind `sourceAddress` against a consignment-carried authorized value where the schema provides one, and otherwise to document it as an explicitly unbound field.

Recommendation

We recommend assembling a single canonical bridge intent before signing and deriving every field of it from one explicit source of truth, then signing over that intent rather than over loosely cross-checked scalars. Concretely:

- **What the intent must contain.** Source network, destination chain id, bridge contract, RGB asset id, amount, recipient, operation id (the bridge `burnId`), and consignment hash — one typed object, fully populated, with no field left as a free-standing request scalar.
- **Where each field is derived from.** Pin destination `chain_id`, bridge contract, and RGB asset id to `BridgeConfig` (the boot-time values already committed into attestation `user_data`); derive source network and asset/amount from the in-enclave-validated consignment (`validated.chain_net` and its burn-asset metadata, the same source used in `validate_funds_out_burn`); derive recipient, amount, operation id, `sourceChainId`, and `destinationChainId` from a full ABI decode of the `fundsOut` calldata (the mint/burn layout at `evm_crosscheck.rs:19-25`), not from fixed-offset scalar reads or duplicated request fields such as `calldata_amount`. The consignment hash stays `keccak256(consignment)` as today.
- **How to verify.** For each field, assert that the calldata-decoded value equals its pinned/validated source of truth and fail closed on any mismatch — e.g. calldata `destinationChainId == bridge_config.chain_id`, calldata `target == bridge_config.bridge_contract`, calldata `amount == consignment burn amount`. Validate calldata `sourceChainId` / `destinationChainId` against the configured route table rather than against `validated.chain_net`, and *separately* confirm that the consignment's `validated.chain_net` matches the Bitcoin network expected for that route: the two live in different domains — an EVM-style route/chain id is not the RGB Bitcoin network string and must not be equated directly. This collapses the several independent identifiers and chain labels into one cross-validated bridge intent so a mismatch between any two sources is rejected before signing instead of being silently accepted.
- **Source and recipient addresses.** Include `recipient` and, where the RGB schema carries it, `sourceAddress` in the intent and bind each to a consignment-derived authorized value rather than trusting the calldata scalar (shared with the recipient-binding finding). If the schema does not yet carry an authorized source address, treat `sourceAddress` as an explicitly unbound field and document that a strict cryptographic proof of the source is outside the current trust model, so the gap is recorded rather than silently assumed safe.

Update

Client's response

Acknowledged (deferred) — the security-relevant metadata is already bounded: chain id and proxy contract are pinned to the attested config, and the operation identifiers are

enclave-authored (PR #97). The remaining informational fields (`sourceChainId`/`sourceAddress`) await the canonical bridge-intent design (#124).

Oxorio's response

The risk remains and is accepted. The security-critical metadata is already constrained — the destination chain id and proxy contract are pinned to the attested config and the operation identifiers are enclave-authored — so the exposure is now mainly a specification and maintainability gap rather than a direct binding flaw. What is still outstanding is that the remaining cross-chain fields (`sourceChainId`, `sourceAddress`, and related labels) are not yet collapsed into a single canonical bridge intent derived entirely from validated facts, which awaits the canonical bridge-intent design (and, for `sourceAddress`, the new RGB schema that would carry an authorized value). Before production, metadata inconsistent with the attested route policy should be rejected, and once the schema supports it, recipient and source address should be derived from authentic consignment metadata and carried in one typed intent.

I-12	`SignEvm` validates selected scalar fields instead of parsing the full `fundsOut` calldata into a typed intent in `evm_crosscheck.rs`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
evm_crosscheck.rs	selector whitelist and fixed-offset scalar extraction	60-172
evm_crosscheck.rs	mint/burn amount extraction from byte offset	289-293
evm_crosscheck.rs	legacy amount / commission extraction from byte offsets	351-360

Description

In the file `evm_crosscheck.rs`, the EVM cross-check reads a small set of fixed scalar slots from calldata, such as `amount` and legacy `commission`, but it does not parse the whole ABI payload into a typed `fundsOut` structure. Fields such as dynamic `proof`, `sourceAddress`, `settlementData`, and `fundsInIds` remain outside a typed enclave-side intent.

This report already treats malformed ABI tail and identifier decoupling as stronger, separate findings. This INFO item records the underlying engineering pattern: validation is field-by-field rather than typed-intent based.

Scalar checks can be correct for the fields they read, but partial parsing makes it easier for future fields to remain unbound or to be signed without being understood by the enclave.

Recommendation

We recommend decoding the entire `fundsOut` calldata into a typed structure per selector instead of reading a few fixed scalar offsets. Concretely:

- **What to decode.** For each whitelisted selector decode the full ABI tuple, not just `amount` / `commission`: for the mint/burn shape (`evm_crosscheck.rs:19-25`) that means `recipient`, `amount`, `burnId`, `sourceChainId`, `destinationChainId`, `sourceAddress`, the dynamic `proof = abi.encode(blockHeight, commitmentHash)`, and `settlementData = abi.encode(uint256[] fundsInIds)`; for the legacy pools shape, the full 6-arg tuple. No field that the contract commits to should remain outside the typed object.

- **How to decode safely.** Use contract-derived ABI decoding (generated types or a fixture pinned to the deployed contract ABI) that validates head/tail offsets and dynamic-array lengths, so a malformed or attacker-padded tail fails closed during decode rather than being silently ignored by fixed-offset reads (`evm_crosscheck.rs:289-293`, `351-360`).
- **Where it feeds.** The resulting typed object must be the single input to the cross-checks and to hashing/signing, and it should be the same object the canonical bridge-intent finding (cross-chain metadata derived from enclave-validated facts) is built from. Anything unparsed, out-of-bounds, or inconsistent with the decoded length fails closed before signing.

Update

Client's response

Fixed (different approach) — the validation entry ABI-decodes the full typed `fundsOut` call and cross-checks the decoded amount against the declared scalar (PR #128), with canonical-encoding enforcement and known-answer tests locking the rewrite offsets to the ABI layout (PR #145).

Partially fixed at: `38c46e90`, `0beefb68`

Oxorio's response

Remaining issue: the entry point now performs a full, canonical ABI decode of the `fundsOut` call — which closes the malformed-tail parsing concern — but the decoded result is still reduced to the amount for validation. The recipient, burn id, source/destination chain ids, source address, proof, and settlement data are decoded and then discarded rather than being carried through as one typed intent, so structurally valid calldata can still contain security-relevant fields that the enclave signs without cross-checking against trusted facts.

Required remediation: carry a single typed intent through every stage — decode, cross-check, authorization, identifier rewrite, and signing — and bind each security-relevant field to attested configuration or to authentic chain/RGB facts, so no decoded field is dropped before signing.

Reproduction: submit a structurally valid `fundsOut` call whose decoded recipient, chain ids, or source address are inconsistent with the validated consignment/attested config but whose amount matches. Because validation collapses to the amount check, the enclave signs it even though the other decoded fields were never cross-checked.

Client's response

Fixed (different approach) in PR #191.

The `fundsOut` calldata is no longer decoded and reduced to `amount`. It is decoded once into a canonical `FundsOutParams` object, and that same typed intent is carried through validation, RGB/EVM cross-checks, authorization, and EIP-712 signing.

The signature is built directly from this typed object and commits to every `fundsOut` field: `recipient`, `amount`, `burnId`, `sourceChainId`, `destinationChainId`, `sourceAddress`, `proof`, and `settlementData`. No decoded field is discarded before signing.

Fields for which the enclave has an independent trusted authority are additionally bound to that authority:

- `amount` → validated RGB consignment;
- `destinationChainId` → attested `EVM_CHAIN_ID`;
- `proof` → independently maintained in-enclave Bitcoin header chain.

`burnId` and `settlementData` are bridge-derived identifiers whose authoritative validity is enforced by the EVM bridge contract, while `recipient` / `sourceAddress` are part of the authorized release intent rather than values reconstructed from the RGB consignment. They are nevertheless carried unchanged in the single typed intent and cryptographically committed by the enclave signature.

Therefore the previous condition where validation collapsed the decoded calldata to `amount` and later signed the remaining opaque fields no longer exists.

Fixed at: `a06e956`

I-13	The master seed is held in memory for the whole enclave lifetime and replicated cluster-wide, maximizing key-compromise blast radius in `keys.rs`, `state.rs`, `server.rs`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
keys.rs	`KeyManager` retains the raw seed alongside every derived secret	42-61
keys.rs	`expose_seed` returns the raw 64-byte seed	246-248
server.rs	the donor exposes the seed to seal it during cloning	829-831

Description

In the file `keys.rs`, the `KeyManager` keeps the raw 64-byte BIP-39 seed resident in memory (`SecretBox<[u8; 64]>`) for the entire lifetime of the enclave, in addition to the per-purpose derived secrets:

```
pub struct KeyManager {
    seed: SecretBox<[u8; 64]>,
    evm_secret: SecretBox<[u8; 32]>,
    evm_gas_tx_secret: SecretBox<[u8; 32]>,
    btc_secret: SecretBox<[u8; 32]>,
    ...
}
```

The seed is retained because the donor side of cloning must re-seal it on demand (`server.rs`, `with_seed` → `encrypt_seed_for_peer`), so `expose_seed` keeps it available throughout the process lifetime rather than only during an active signing or cloning operation. The same seed is then replicated to every cloned enclave in the cluster by design.

Two properties follow. First, the seed's residency window is the whole enclave lifetime, which is wider than necessary for routine signing (which only needs the derived secrets). Second, because one seed reconstructs every key and is shared cluster-wide, a memory-disclosure bug in any single enclave compromises the keys of the entire cluster, not just that

instance. This is not directly exploitable on its own; it increases the blast radius of any other memory-safety or extraction issue (including the cloning-authorization finding).

Recommendation

We recommend minimizing seed residency and reducing the shared blast radius. Concretely, keep only the derived signing secrets resident on the hot path and avoid holding the raw seed for the whole lifetime — for example reconstruct or unseal it only for the duration of an active cloning handshake and zeroize it immediately afterwards, and drop it entirely when cloning is disabled by policy. For a more fundamental reduction, consider per-enclave key shares or threshold signing so that no single seed in any one enclave reconstructs all cluster keys.

Update

Client's response

Acknowledged (architectural) — the cloning protocol requires the donor to re-seal the seed on demand, which forces residency; changing that is a recovery-architecture redesign (threshold keys or per-request unsealing), scheduled with the cluster-clone work (#124).

Oxorio's response

The risk remains and is accepted. The raw 64-byte seed still stays resident for the whole enclave lifetime because the donor side of cloning must re-seal it on demand, and the same seed is replicated cluster-wide, so a memory-disclosure bug in any one enclave would expose the keys of the entire cluster. This is not directly exploitable on its own; it widens the blast radius of any other memory-safety or extraction issue. The compensating controls are that the seed is held in `SecretBox`, enclave memory sits inside the Nitro attested boundary, and cloning requires attestation plus the cloning secret. Reducing residency requires a recovery-architecture redesign (per-request unsealing or threshold/shared keys), which is deferred to the cluster-clone work; before production, cloning access and the number of enclave copies should be minimized and the target architecture should ensure no single enclave reconstructs all cluster keys.

I-14	Inability of the enclave to operate with tokens that use the standard `decimals` value of `18` in `evm_crosscheck.rs`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
evm_crosscheck.rs	-	383

Description

In `evm_crosscheck.rs#L383`, within the `extract_uint256_as_u64` function, a `uint256` value is accepted only if it fits into `u64`. The parser rejects any amount whose high 24 bytes are non-zero, so any value greater than `2^64 - 1` fails validation:

```
// High 24 bytes must be zero for value to fit in u64
if slot[..24].iter.any(|&b| b != 0) {
    return Err(EnclaveError::CrossCheck(
        "uint256 value exceeds u64 range".into,
    ));
}
```

The `extract_uint256_as_u64` function is used to convert the `amount` of `EVM` tokens from the calldata. However, the standard `ERC20` token uses a default `decimals` value of `18`.

For example, if the `DAI` token (`decimals = 18`) is used as `TOKEN` in the bridge contract, any transfer amount above approximately `18.44` whole tokens (≈ $18.44 for a $1-pegged token) will cause an enclave error due to exceeding the `u64` range. The `18.44` figure comes from the `u64` ceiling itself: `2^64 - 1 ≈ 1.845 × 10^19` base units, which at `18` decimals is `1.845 × 10^19 / 10^18 ≈ 18.44` whole tokens.

Recommendation

We recommend either documenting an explicitly supported token `decimals` value and a maximum transfer amount in the specification, or — preferably — handling the amount as a full `U256` (or a wider integer) in `evm_crosscheck.rs`, so standard 18-decimal tokens can be

bridged without hitting the current `u64` ceiling. Whichever path is chosen, preserve fail-closed parsing for out-of-range values.

Update

Client's response

Acknowledged (scope decision) — the bridged asset fits in `u64` at its own precision, and RGB itself is `u64`-bounded, so a wider range cannot be represented end-to-end; amounts above `u64` are rejected fail-closed and never truncated (#124).

Oxorio's response

The risk remains and is accepted as a scope decision. `extract_uint256_as_u64` still rejects any `uint256` whose high 24 bytes are non-zero, so an 18-decimal token would fail validation above roughly `18.44` whole tokens. The client's rationale is that the bridged RGB asset is itself `u64`-bounded, so a wider range cannot be represented end-to-end, and the important safety property holds: out-of-range values fail closed and are never truncated, so no incorrect (wrapped) amount can be signed. Before production, the production configuration and specification should pin the supported asset precision and maximum transfer amount so operators do not deploy a token whose real transfer sizes exceed the representable range.

I-15	Suppression of warnings when fresh transactions are present in `consignment` in `rgb.rs`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
rgb.rs	`ValidationConfig` assembled without `safe_height`	263-268
rgb.rs	`validate` result bound to `_valid` and discarded	272

Description

In the `validate_consignment` function of `impl RgbValidator`, `consignment` validation is performed. As part of the validation parameters, a `ValidationConfig` structure is assembled, in which the `safe_height` field may be specified. This field determines the block height, and the most recent blocks are considered unsafe.

However, this value is not set, meaning `safe_height = None`. Even if this value were set, the warning that would appear during validation would still be ignored, since the validation result `_valid` is never used:

```
let _valid = transfer.validate(&resolver, &config).map_err(|e| {/*... */})?;
```

As a result, after resolution in `Esplora`, transactions that have the status `Tentative`, `Ignored`, or `Archived` will be accepted as valid without any warning being handled.

At the same time, later, during processing of the request to sign `EVM` calldata, a check will be performed in `spv`. This check includes the requirement that blocks reach the `SPV_MIN_CONFIRMATIONS` height.

In other words, currently only the `spv` check validates the block height, and in the event of compromise of the `HeadChain`, it may be possible to pass a `consignment` with forged `Tentative` transactions into the enclave, and such a `consignment` may be accepted.

Recommendation

We recommend failing closed on a non-final validation outcome instead of discarding it: inspect the `transfer.validate` result rather than binding it to `_valid`, and reject the consignment when validation reports warnings or when any witness transaction is in a non-final state (`Tentative`, `Ignored`, `Archived`). In addition, set the `safe_height` field in `ValidationConfig` so recent/unsafe blocks are flagged during validation. Since `Esplora` resolution occurs before the heavy `spv` proof parsing, an earlier block-height/status check can reject the request immediately and avoid the expensive computation.

Update

Client's response

Fixed (different approach) — the `validate()` status is inspected and non-mined witnesses are surfaced and rejected on the fundsOut direction (PR #118).

Fixed at: `ef9aca2d`

I-16	No reproducible-build pipeline exists for independently deriving the expected PCR measurements in `build/`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
build-enclave.sh	enclave EIF build script — not deterministic / no published PCR derivation	—
Dockerfile.enclave	non-reproducible image base for the enclave build	—

Description

In the build tooling, there is no reproducible-build pipeline that lets an external party rebuild `enclave-signer` from source and obtain PCR0/PCR1 values matching the published attestation of the running enclave. Attestation can prove a key was generated inside an enclave with specific PCRs, but without reproducible builds no one can independently verify that the public source corresponds to those PCRs — weakening the trust argument that the deployed image is the audited code.

Recommendation

We recommend setting up a deterministic (e.g. Nix-flake-based) build so that cloning the repository and running a single build command yields bit-for-bit identical EIF and PCR0/PCR1 values, and publishing those values alongside the attestation for independent verification.

Update

Client's response

Fixed as recommended — a secret-free EIF build with a pinned Rust toolchain, pinned `nitro-cli`, and dependencies locked by `Cargo.lock`; a single build command emits `PCR.json`; PCR measurements are published in CI; at deploy time the PCRs are checked against this manifest (PR #113, PR #96). The only remaining step toward full bit-for-bit external reproducibility is digest-pinning the base images (`FROM …@sha256:`) instead of by tag.

Partially fixed at: `359dae47`

Oxorio's response

Remaining issue: The image and its PCR values are still verified from the same release source, and PR #96 (the reviewed deterministic-build proposal) is not merged while PR #113 only checks build artifacts against PCR data produced by that same source. Mutable base-image tags and package resolution remain build inputs, so an external party still cannot independently rebuild the EIF from public source and reproduce the attested PCR0/PCR1, which is exactly the trust argument attestation is meant to support.

Required remediation: Merge a reviewed, deterministic build, digest-pin every base image and tool (`FROM …@sha256:`), publish signed provenance and PCR values, and then independently rebuild the EIF and compare the measurements against the attested production release.

Reproduction: Clone the repository at the deployed revision and run the build; because base-image tags and package resolution are not digest-pinned, the resulting EIF and PCR0/PCR1 are not guaranteed to match the production attestation bit-for-bit, so an external verifier cannot prove the deployed image corresponds to the public source. This is a build/verification gap rather than a runtime exploit; PR #96 remains unmerged and PR #113 provides deployment integrity against same-source PCR data only.

Client's response

The build pipeline now includes the reproducibility foundations: a secret-free EIF build with a pinned Rust toolchain, pinned `nitro-cli`, and dependencies locked by `Cargo.lock --locked`; deterministic build timestamps; a single build command that emits `PCR.json`; PCR measurements published in CI; and a deploy-time check of the PCRs against this manifest. The base-image digest-pinning (`FROM …@sha256:`) is added in the reproducible-build-pins change; the remaining clean-room independent rebuild will be performed before the production release.

4. APPENDIX

4.1 SECURITY ASSESSMENT METHODOLOGY

Oxorio's smart contract security audit methodology is designed to ensure the security, reliability, and compliance of smart contracts throughout their development lifecycle. Our process integrates the Smart Contract Security Verification Standard (SCSVS) with our advanced techniques to address complex security challenges. For a detailed look at our approach, please refer to the full version of our methodology. Here is a concise overview of our auditing process:

1. Project Architecture Review

All necessary information about the smart contract is gathered, including its intended functionality and dependencies. This stage sets the foundation by reviewing documentation, business logic, and initial code analysis.

2. Vulnerability Assessment

This phase involves a deep dive into the smart contract's code to identify security vulnerabilities. Rigorous testing and review processes are applied to ensure robustness against potential attacks.

This stage is focused on identifying specific vulnerabilities within the smart contract code. It involves scanning and testing the code for known security weaknesses and patterns that could potentially be exploited by malicious actors.

3. Security Model Evaluation

The smart contract's architecture is assessed to ensure it aligns with security best practices and does not introduce potential vulnerabilities. This includes reviewing how the contract integrates with external systems, its compliance with security best practices, and whether the overall design supports a secure operational environment.

This phase involves a analysis of the project's documentation, the consistency of business logic as documented versus implemented in the code, and any assumptions made during the design and development phases. It assesses if the contract's architectural design adequately addresses potential threats and integrates necessary security controls.

4. Cross-Verification by Multiple Auditors

Typically, the project is assessed by multiple auditors to ensure a diverse range of insights and thorough coverage. Findings from individual auditors are cross-checked to verify accuracy and completeness.

5. Report Consolidation

Findings from all auditors are consolidated into a single, comprehensive audit report. This report outlines potential vulnerabilities, areas for improvement, and an overall assessment of the smart contract's security posture.

6. Reaudit of Revised Submissions

Post-review modifications made by the client are reassessed to ensure that all previously identified issues have been adequately addressed. This stage helps validate the effectiveness of the fixes applied.

7. Final Audit Report Publication

The final version of the audit report is delivered to the client and published on Oxorio's official website. This report includes detailed findings, recommendations for improvement, and an executive summary of the codebase's security status.

4.2 CODEBASE QUALITY ASSESSMENT REFERENCE

The tables below describe the codebase quality assessment categories and rating criteria used in this report.

Category	Description
Access Control	Evaluates the effectiveness of mechanisms controlling access to ensure only authorized entities can execute specific actions, critical for maintaining system integrity and preventing unauthorized use.
Arithmetic	Focuses on the correct implementation of arithmetic operations to prevent vulnerabilities like overflows and underflows, ensuring that mathematical operations are both logically and semantically accurate.
Complexity	Assesses code organization and function clarity to confirm that functions and modules are organized for ease of understanding and maintenance, thereby reducing unnecessary complexity and enhancing readability.
Data Validation	Assesses the robustness of input validation to prevent common vulnerabilities like overflow, invalid addresses, and other malicious input exploits.
Decentralization	Reviews the implementation of decentralized governance structures to mitigate insider threats and ensure effective risk management during contract upgrades.
Documentation	Reviews the comprehensiveness and clarity of code documentation to ensure that it provides adequate guidance for understanding, maintaining, and securely operating the codebase.
External Dependencies	Evaluates the extent to which the codebase depends on external protocols, oracles, or services. It identifies risks posed by these dependencies, such as compromised data integrity, cascading failures, or reliance on centralized entities. The assessment checks if these external integrations have appropriate fallback mechanisms or redundancy to mitigate risks and protect the protocol's functionality.
Error Handling	Reviews the methods used to handle exceptions and errors, ensuring that failures are managed gracefully and securely.
Logging and Monitoring	Evaluates the use of event auditing and logging to ensure effective tracking of critical system interactions and detect potential anomalies.
Low-Level Calls	Reviews the use of low-level constructs like inline assembly, raw `call` or `delegatecall`, ensuring they are justified, carefully implemented, and do not compromise contract security.

Category	Description
Testing and Verification	Reviews the implementation of unit tests and integration tests to verify that codebase has comprehensive test coverage and reliable mechanisms to catch potential issues.

4.2.1 Rating Criteria

Rating	Description
Excellent	The system is flawless and surpasses standard industry best practices.
Good	Only minor issues were detected; overall, the system adheres to established best practices.
Fair	Issues were identified that could potentially compromise system integrity.
Poor	Numerous issues were identified that compromise system integrity.
Absent	A critical component is absent, severely compromising system safety.
Not Applicable	This category does not apply to the current evaluation.

4.3 FINDINGS CLASSIFICATION REFERENCE

4.3.1 Severity Level Reference

The following severity levels were assigned to the issues described in the report:

Title	Description
CRITICAL	Issues that pose immediate and significant risks, potentially leading to asset theft, inaccessible funds, unauthorized transactions, or other substantial financial losses. These vulnerabilities represent serious flaws that could be exploited to compromise or control the entire contract. They require immediate attention and remediation to secure the system and prevent further exploitation.
MAJOR	Issues that could cause a significant failure in the contract's functionality, potentially necessitating manual intervention to modify or replace the contract. These vulnerabilities may result in data corruption, malfunctioning logic, or prolonged downtime, requiring substantial operational changes to restore normal performance. While these issues do not immediately lead to financial losses, they compromise the reliability and security of the contract, demanding prioritized attention and remediation.
WARNING	Issues that might disrupt the contract's intended logic, affecting its correct functioning or making it vulnerable to Denial of Service (DDoS) attacks. These problems may result in the unintended triggering of conditions, edge cases, or interactions that could degrade the user experience or impede specific operations. While they do not pose immediate critical risks, they could impact contract reliability and require attention to prevent future vulnerabilities or disruptions.
INFO	Issues that do not impact the security of the project but are reported to the client's team for improvement. They include recommendations related to code quality, gas optimization, and other minor adjustments that could enhance the project's overall performance and maintainability.

4.3.2 Status Level Reference

Based on the feedback received from the client's team regarding the list of findings discovered by the contractor, the following statuses were assigned to the findings:

Title	Description
NEW	Waiting for the project team's feedback.

Title	Description
FIXED	Recommended fixes have been applied to the project code and the identified issue no longer affects the project's security.
ACKNOWLEDGED	The project team is aware of this finding and acknowledges the associated risks. This finding may affect the overall security of the project; however, based on the risk assessment, the team will decide whether to address it or leave it unchanged.
NO ISSUE	Finding does not affect the overall security of the project and does not violate the logic of its work.

4.4 ABOUT OXORIO

OXORIO is a blockchain security firm that specializes in smart contracts, zk-SNARK solutions, and security consulting. With a decade of blockchain development and five years in smart contract auditing, our expert team delivers premier security services for projects at any stage of maturity and development.

Since 2021, we've conducted key security audits for notable DeFi projects like Lido, 1Inch, Rarible, and deBridge, prioritizing excellence and long-term client relationships. Our co-founders, recognized by the Ethereum and Web3 Foundations, lead our continuous research to address new threats in the blockchain industry. Committed to the industry's trust and advancement, we contribute significantly to security standards and practices through our research and education work.

Our contacts:

- oxor.io
- ping@oxor.io
- Github
- Linkedin
- Twitter

THANK YOU FOR CHOOSING

