

UTEXO BRIDGE BURN SIGNER SECURITY AUDIT REPORT

1

EXECUTIVE SUMMARY

1.1 EXECUTIVE SUMMARY

This document presents the security audit conducted by Oxorio for UTEXO Protocol's UTEXO Bridge burn 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 burn signer is the enclave image that checks an RGB burn and signs the EVM release. `MultisigProxy` accepts that signature on `fundsOutCall` and `1zFundsOutCall`, and `Bridge.fundsOut` pays it. The same zero-trust model as the Bridge audit applies: no off-chain actor is assumed correct unless an on-chain check enforces it. On this image the release still depends on the enclave for several binds the contract does not repeat.

The audit process involved a comprehensive approach, including manual code review of the burn path against the August 12, 2026 enclave and Solidity reports and the September 23, 2026 Bridge re-audit. The audit covered 61 Rust files, encompassing 15,359 total lines (11,014 lines of code). The proofs of concept were traced at enclave [dd55d5f](#). Solidity [c2a5f213](#) and the LayerZero adapter [be01e61](#) are the contracts the findings were traced against. They are not part of this file scope. This report does not re-score every finding of those earlier reports. Where a finding reopens or raises an earlier item, the finding itself says which one. The remediations were read at enclave [f658a45](#) and bridge [e49adf7](#). For an in-depth explanation of the security audit methodology, please refer to the [Security Assessment Methodology](#) section of this document.

This report contains 3 major findings, 5 warnings, and 5 informational findings. There is no critical finding.

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 Burn Signer, as of enclave `dev@f658a45` and bridge `dev@e49adf7`, 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

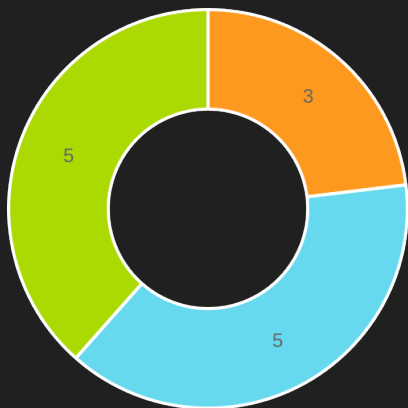
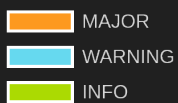
The table below provides a comprehensive summary of the audit findings, categorizing each by status and severity level. For a detailed description of the severity levels and statuses of findings, see the [Findings Classification Reference](#) section.

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

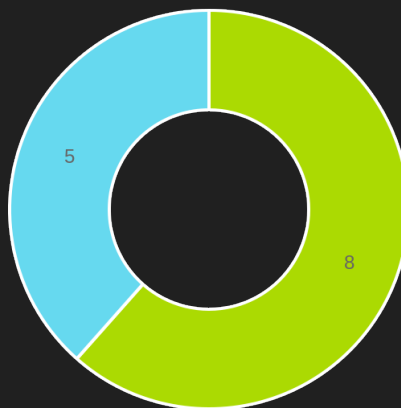
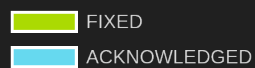
Of the 13 findings, 8 are FIXED and 5 are ACKNOWLEDGED. None remain open.

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

Severity	TOTAL	NEW	FIXED	ACKNOWLEDGED	NO ISSUE
CRITICAL	0	0	0	0	0
MAJOR	3	0	3	0	0
WARNING	5	0	3	2	0
INFO	5	0	2	3	0
TOTAL	13	0	8	5	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	6
2.1. DISCLAIMER	9
2.2. PROJECT BRIEF	10
2.3. PROJECT TIMELINE	11
2.4. AUDITED FILES	13
2.5. PROJECT OVERVIEW	17
2.6. CODEBASE QUALITY ASSESSMENT	18
2.7. FINDINGS BREAKDOWN BY FILE	20
2.8. CONCLUSION	21
3. FINDINGS REPORT	22
3.1. CRITICAL	23
3.2. MAJOR	24
M-01 Outbound IzFundsOutCall does not bind dstEid to destinationChainId, so a release can be delivered to a chain the burn did not name in MultisigProxy.sol, UtexoLZAdapter.sol, validation.rs	24
M-02 The mandatory Helios EVM source introduced for M-06 in the previous reports was removed, so the deposit set inside burnId again depends on a trusted RPC in policy.rs, events.rs, Bridge.sol	28
M-03 A burn of units minted after 62 deposits cannot be released on this implementation, because settlement must cite every mint in that unit's history and settlementData is capped at 4096 bytes in crosscheck.rs, bfa.rs, Bridge.sol	32
3.3. WARNING	37
W-01 minAmountLD is not taken from the burned amount in signing.rs	37

W-02 The LayerZero native fee is not tied to one release in gas_tx.rs	40
W-03 extraOptions are signed without being parsed in signing.rs	43
W-04 A signet burn can be signed for a production EVM domain in spv_crosscheck.rs and validation.rs	45
W-05 The signature stays valid after the bridge implementation changes in policy.rs and MultisigProxy.sol	49
3.4. INFO	52
I-01 The enclave does not apply the one-day deadline cap of MultisigProxy in validation.rs	52
I-02 The signed nonce is taken from the request rather than from teeNonce in signers.rs	54
I-03 The header-chain lock is not held across the signature in signers.rs	56
I-04 EVM confirmation depth is measured against a head from the same provider that returned the receipt in events.rs	58
I-05 Enclave finality thresholds are not aligned with the RGBVerifier thresholds in crosscheck.rs and RGBVerifier.sol	60
4. APPENDIX	62
4.1. SECURITY ASSESSMENT METHODOLOGY	63
4.2. CODEBASE QUALITY ASSESSMENT REFERENCE	65
Rating Criteria	66
4.3. FINDINGS CLASSIFICATION REFERENCE	67
Severity Level Reference	67
Status Level Reference	67
4.4. ABOUT OXORIO	69

2.1 DISCLAIMER

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

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 Burn Signer
Category	TEE signer (AWS Nitro Enclave)
Website	https://utexo.com
Documentation	https://docs.utexo.com
Repository	enclave-signer
Initial Commit	dd55d5f (October 5, 2026)
Diff	2df6d50...dd55d5f , from the August 12, 2026 enclave report
Final Commit	Enclave f658a45 (October 16, 2026). Bridge e49adf7 (October 14, 2026)
Bridge repository	bridge-smart-contracts , outside this file scope
Bridge commit	c2a5f213 (October 1, 2026), Solidity 0.8.35
Adapter repository	usdt0-smart-contracts , outside this file scope
Adapter commit	be01e61 (October 5, 2026)
Build	<code>--no-default-features --features vsock,rgb,burn-signer</code>
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 their integration into <code>dev</code> .
August 12, 2026	Submission of the final audit report at <code>ea5de9b4</code> .
September 17, 2026	Re-audit of the Bridge delta from <code>ea5de9b4</code> to <code>95d8e566</code> initiated.
September 22, 2026	Submission of the re-audit report at <code>95d8e566</code> .
September 29, 2026	Client began remediation of the 15 findings.
October 1, 2026	Remediation submitted at <code>c2a5f213</code> .
October 1, 2026	Re-audit of that remediation.
October 2, 2026	Submission of the final re-audit report incorporating the verified fixes.
October 5, 2026	The client fixed the burn-signer scope: enclave commit <code>dd55d5f</code> , directory <code>enclave/</code> , build <code>--no-default-features --features vsock,rgb,burn-signer</code> .
October 6, 2026	The client granted access to the private RGB dependency repositories. The audit team initiated work on the burn signer.

Date	Event
October 14, 2026	Submission of the preliminary audit report.
October 16, 2026	Client submitted the remediations. Enclave f658a45 , bridge e49adf7 .
October 22, 2026	Re-audit of those remediations.
October 23, 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.

The build is `--no-default-features --features vsock,rgb,burn-signer`. Files that this build does not compile are not in the table: the mint-signer seed path (`kms/seed_persistence.rs`), the Concordium module, the send/receive flow (`flow/swap.rs`), and the EVM-to-RGB PSBT path (`psbt_validation.rs`, `psbt_fee_size.rs`, `btc_crosscheck.rs`, `btc_ownership.rs`, `invoice.rs`). Test files are not in the table. Solidity and the adapter are outside this file scope.

	File	Lines	Blanks	Comments	Code	Complexity
1	enclave/src/config.rs	1023	77	177	769	9%
2	enclave/src/networks/evm/events.rs	893	60	194	639	6%
3	enclave/src/server/sign.rs	728	52	142	534	5%
4	enclave/src/networks/evm/crosscheck.rs	648	49	165	434	9%
5	enclave/src/networks/rgb/spv_crosscheck.rs	616	63	119	434	7%
6	enclave/src/server/endpoints.rs	489	30	25	434	7%
7	enclave/src/cloning.rs	535	63	64	408	2%
8	enclave/src/state/enclave.rs	550	53	120	377	6%
9	enclave/src/networks/evm/gas_tx.rs	525	38	116	371	19%
10	enclave/src/bootstrap.rs	441	23	61	357	10%
11	enclave/src/networks/evm/validation.rs	487	35	121	331	14%
12	enclave/src/keys.rs	433	55	51	327	6%
13	enclave/src/server/bfa.rs	389	38	59	292	4%
14	enclave/src/policy.rs	380	17	101	262	20%

	File	Lines	Blanks	Comments	Code	Complexity
15	enclave/src/server/sign/bridge_operation_retry.rs	290	23	15	252	1%
16	enclave/src/networks/rgb/spv/chain.rs	391	45	105	241	13%
17	enclave/src/server/keys.rs	291	19	31	241	4%
18	enclave/src/networks/rgb/validation/consignment.rs	329	25	66	238	6%
19	enclave/src/networks/rgb/spv/checkpoint.rs	364	24	118	222	8%
20	enclave/src/server/dispatch.rs	237	5	17	215	6%
21	enclave/src/networks/rgb/validation/consensus.rs	261	19	54	188	7%
22	enclave/src/server/signers.rs	276	31	57	188	7%
23	enclave/src/server/cloning.rs	268	30	52	186	6%
24	enclave/src/networks/route.rs	247	19	56	172	3%
25	enclave/src/attestation.rs	213	25	20	168	5%
26	enclave/src/networks/evm/signing.rs	231	28	39	164	5%
27	enclave/src/networks/rgb/signing/taproot.rs	204	19	25	160	8%
28	enclave/src/conn.rs	206	21	33	152	5%
29	enclave/src/networks/rgb/spv/validation.rs	217	25	42	150	5%
30	enclave/src/networks/rgb/route.rs	199	19	31	149	5%
31	enclave/src/state/replay_guard.rs	202	25	44	133	13%
32	enclave/src/networks/rgb/spv/merkle.rs	195	22	50	123	7%

	File	Lines	Blanks	Comments	Code	Complexity
33	enclave/src/main.rs	157	17	22	118	7%
34	enclave/src/networks/ rgb/validation/bfa.rs	200	16	67	117	10%
35	enclave/src/server/ wire.rs	142	19	11	112	4%
36	enclave/src/networks/ rgb/validation/source.rs	142	10	27	105	8%
37	enclave/src/framing.rs	133	19	16	98	5%
38	enclave/src/networks/ rgb/validation/indexer.rs	154	10	48	96	6%
39	enclave/src/server/ context.rs	144	9	41	94	0%
40	enclave/src/error.rs	120	25	2	93	2%
41	enclave/src/lib.rs	130	11	26	93	0%
42	enclave/src/ vsock_forwarder.rs	109	10	17	82	12%
43	enclave/src/networks/ rgb/spv/types.rs	109	20	11	78	3%
44	enclave/src/networks/ rgb/flow/mint_burn.rs	126	8	42	76	12%
45	enclave/src/server/ rate_limit.rs	104	11	18	75	4%
46	enclave/src/clocksync.rs	102	12	23	67	6%
47	enclave/src/networks/ rgb/validation/ asset_bind.rs	91	6	19	66	15%
48	enclave/src/server/ health.rs	89	11	14	64	5%
49	enclave/src/server/spv.rs	69	10	6	53	0%
50	enclave/src/networks/ rgb/validation/types.rs	155	5	103	47	2%
51	enclave/src/state/ cloning_session.rs	57	8	14	35	3%
52	enclave/src/networks/ rgb/mod.rs	46	2	19	25	0%

	File	Lines	Blanks	Comments	Code	Complexity
53	enclave/src/server/mod.rs	27	2	5	20	0%
54	enclave/src/networks/rgb/validation/mod.rs	36	3	14	19	0%
55	enclave/src/networks/rgb/validation/schema.rs	39	3	18	18	11%
56	enclave/src/networks/rgb/spv/mod.rs	26	2	10	14	0%
57	enclave/src/networks/evm/mod.rs	15	1	2	12	0%
58	enclave/src/networks/mod.rs	18	2	7	9	0%
59	enclave/src/networks/rgb/flow/mod.rs	35	2	25	8	0%
60	enclave/src/state/mod.rs	17	3	6	8	0%
61	enclave/src/networks/rgb/signing/mod.rs	9	1	7	1	0%
	Total	15359	1335	3010	11014	7%

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 burn signer is the enclave half of an RGB to EVM release. A user burns RGB units first. The listener then sends the consignment and the `fundsOut` or `lzFundsOut` calldata to the enclave. The enclave checks the consignment, the Bitcoin inclusion of its witnesses, and the EVM deposits behind the mint ancestry, and it signs an EIP-712 digest for `MultisigProxy`. The contract checks the quorum and calls `Bridge.fundsOut`, which debits `lockedLiquidity`, charges commission, and asks the route's settlement module to accept `settlementData`. On the LayerZero route, `lzFundsOutCall` pays the adapter and the adapter delivers to `dstEid`.

`settlementData` has to name the deposits behind that burn. At the audited pin, `burnId` includes the settlement bytes. The enclave is what ties the signed fields to the burn, because the contract does not repeat every bind. The three major findings are the places where that tie was missing or could not be executed: the LayerZero destination was not the chain the burn was commissioned for, the deposit set inside `burnId` depended on the RPC operator after Helios was removed, and a burn whose mint history was longer than 62 deposits could not be settled under the 4096-byte cap. The remediations re-checked for this report are enclave [f658a45](#) and bridge [e49adf7](#). Each finding records what changed.

The audited build is [enclave-signer](#) at [dd55d5f8](#), directory `enclave/`.

Repository	enclave-signer
Commit	dd55d5f8
Directory	<code>enclave/</code>
Build	<code>--no-default-features --features vsock,rgb,burn-signer</code>
Features	<code>vsock, rgb, burn-signer</code>

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	The enclave quorum and the proxy nonce still gate a release. A V1 burn now commits the LayerZero chain, endpoint and recipient. The signature still does not name the bridge implementation. The attested policy now commits the Bitcoin network. Related: implementation bind.	Fair
Arithmetic	Solidity 0.8 checked arithmetic is unchanged. No finding is an overflow or a rounding error. The outflow settlement cap is an exact byte bound, <code>128 + 64 * n</code> , now 90,000 bytes.	Good
Complexity	The burn path is one pipeline from consignment to digest. The new surface relative to the August enclave is the LayerZero signature and the ancestry settlement bind, not a second governance system.	Good
Data Validation	<code>extraOptions</code> are limited to one gas-only <code>lzReceive</code> option. <code>minAmountLD</code> must be above zero and no higher than the burned amount. A new mint's settlement cites the deposits that burn spends, not every earlier mint.	Good
Decentralization	Outflow caps from the Bridge audit still bound a single release. <code>burnId</code> no longer hashes <code>settlementData</code> . FundsIn receipts on the mint path still come from the pinned RPC, and a bridge upgrade still leaves the signatures of the last day valid. Related: implementation bind.	Fair
Documentation	<code>docs/burn-destination.md</code> and <code>docs/mint-flow.md</code> record the destination record and the mint-right rules checked in this re-audit. <code>docs/tee-spec.md</code> records the pinned RPC as a design assumption. The spec table that says the LayerZero route skips the settlement bind is behind the code, which runs the bind on both routes.	Good

Category	Assessment	Result
External Dependencies	Deposit receipts come from a pinned TLS endpoint. Helios is not in the burn-signer build. LayerZero options are decoded before signing. The native fee is still not tied to one release. Related: native fee.	Fair
Error Handling	Failed checks return no signature. The contract uses custom errors, including SettlementDataTooLong for a payload over 90,000 bytes. A release the enclave refuses is not signed.	Good
Logging and Monitoring	No finding is a missing event. Contract events on the release path are the ones reviewed in the Bridge audit.	Good
Low-Level Calls	The gas key signs the transaction that submits a release. Native value is allowed for 1zFundsOutCall up to a per-transaction ceiling and is not tied to one release. Related: LayerZero native fee.	Fair
Testing and Verification	The codebase includes tests for the burn-signing flow. The signing checks and remediations were also assessed through source-code review.	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
ethereum/src/MultisigProxy.sol	6	0	1	3	2
enclave/src/networks/evm/crosscheck.rs	3	0	2	0	1
enclave/src/networks/evm/signing.rs	3	0	0	3	0
enclave/src/networks/evm/validation.rs	3	0	1	1	1
enclave/src/policy.rs	3	0	1	2	0
ethereum/src/Bridge.sol	3	0	3	0	0
enclave/src/networks/evm/events.rs	2	0	1	0	1
enclave/src/networks/rgb/spv_crosscheck.rs	2	0	0	1	1
enclave/src/server/bfa.rs	2	0	1	1	0
enclave/src/server/signers.rs	2	0	0	0	2
enclave/Cargo.toml	1	0	1	0	0
enclave/src/networks/evm/gas_tx.rs	1	0	0	1	0
enclave/src/networks/rgb/validation/bfa.rs	1	0	1	0	0
enclave/src/server/sign.rs	1	0	1	0	0
ethereum/src/UtexolZAdapter.sol	1	0	1	0	0
ethereum/src/UtexolZAdapter.sol	1	0	0	1	0
ethereum/src/Settlement/RgbSettlementModule.sol	1	0	1	0	0
ethereum/src/verifiers/RGBVerifier.sol	1	0	0	0	1

2.8 CONCLUSION

A comprehensive audit was conducted on 61 Rust files, revealing no critical and 3 major issues, along with 5 warnings and 5 informational notes. At the audited pin, `lzFundsOutCall` did not bind `dstEid` to `destinationChainId`, the deposit set inside `burnId` depended on the RPC operator after the Helios requirement from the previous reports was removed, and a burn whose mint history was longer than 62 deposits could not be settled under the 4096-byte cap. Two of the major findings reopen or raise an item from the August 12, 2026 reports, and each finding says which. This report does not re-score that finding list. Bridge `c2a5f213` and adapter `be01e61` are the contracts the findings were traced against, and they are outside the file scope.

Following our initial audit, UTEXO Protocol addressed the identified issues. The LayerZero delivery chain is now the chain the burn commits to. `burnId` no longer hashes `settlementData`, and the enclave derives each mint's deposit id without an RPC read. A new mint consumes its mint right, and a mint parent is refused, so a new mint is not part of a later mint's history. The outflow cap is 90,000 bytes. `minAmountLD` cannot be zero or above the burned amount. `extraOptions` are limited to one gas-only `lzReceive` option. The attested policy commits the Bitcoin network and refuses a mainnet EVM pin on any other Bitcoin network. The one-day deadline is enforced in the enclave, and the header-chain lock is held until the key signs.

Five findings stay accepted residuals. The gas-key native fee is not tied to one release. The signature domain is still the proxy address. The signed nonce is still the request nonce. EVM finality still comes from the one pinned RPC. The enclave does not apply the relay freshness and gap rules of `RGBVerifier`.

As a result, the project has passed our audit. Our auditors have verified that the UTEXO Bridge Burn Signer, as of enclave `dev@f658a45` and bridge `dev@e49adf7`, operates as intended within the defined scope, based on the information and code provided at the time of evaluation. Of the 13 findings, 8 are verified fixed and 5 are acknowledged as accepted residual risks. None remain open.

3 FINDINGS REPORT

3.2 MAJOR

M-01

Outbound `lzFundsOutCall` does not bind `dstEid` to `destinationChainId`, so a release can be delivered to a chain the burn did not name in `MultisigProxy.sol`, `UtexoLZAdapter.sol`, `validation.rs`

Severity **MAJOR**

Status • FIXED

Location

File	Location	Line
MultisigProxy.sol	contract <code>MultisigProxy</code> > function <code>lzFundsOutCall</code>	450-484
Bridge.sol	contract <code>Bridge</code> > function <code>fundsOut</code> (commission and route by <code>destinationChainId</code>)	566-568
UtexoLZAdapter.sol	contract <code>UtexoLZAdapter</code> > function <code>sendOut</code>	397-419
validation.rs	function <code>validate_destination</code>	373-387
crosscheck.rs	function <code>validate_funds_out_burn_recipient</code>	72-75

Description

In the function `lzFundsOutCall` of `MultisigProxy`, the signed `dstEid` is not checked against `destinationChainId`.

The final Solidity report of August 12, 2026 already recorded this, as ACKNOWLEDGED: "destinationChainId is not constrained to `block.chainid` on `fundsOut` in `Bridge.sol`" (WARNING). The report covers initial commit [80485109](#) (May 21, 2026) and final commit [ea5de9b4](#) (August 10, 2026). `Bridge.fundsOut` checked only that `destinationChainId` was not zero. It did not check which chain would receive the tokens. In the report's PoC, a signed `fundsOut` with `destinationChainId` = 99999 on a chain whose `block.chainid` is 31337 released the tokens. The report concluded that "the correctness of the routing check is delegated entirely to the off-chain backend".

The client declined a check that the destination equals the bridge chain, because a LayerZero release legitimately leaves for another chain, and stated what destination correctness then relies on:

`destinationChainId` correctness ... relies on the off-chain backend / federation signing process

Oxorio accepted the risk with that comment and added that, before production, a forwarded release should be validated "against the route configuration or the paired `dstEid` mapping". That check is still absent at `c2a5f213`.

The enclave audited in August could not have made the check. Its baseline [2df6d50](#) (August 10, 2026) contains no `TeeLzFundsOut`, and neither do the remediation heads that report lists (`091cc8ff` July 31, `b37d81a6` August 2, `d76efc0b` August 7, `a06e9565` August 10). There was no LayerZero signing path, so there was no comparison of the commission chain with the delivery chain.

On August 18, commit [c8633d8](#) (#172) the enclave started signing `TeeLzFundsOut`: take the tokens from the pool and send them through LayerZero to another chain. The check named in that ACKNOWLEDGED item was not added to this path. On August 20, [8db9250](#) (#200) requires only that `destinationChainId` is non-zero and not the home chain. `dstEid`, the chain LayerZero actually delivers to, is not checked. On October 5, [6ed655b](#) (#275) binds the burn recipient and states the gap in the code:

```
// crosscheck.rs:72-75
/// `lzFundsOut` LayerZero `recipient`. `dstEid` is not bound: the burn does not
/// name a chain.
```

The contract function itself is older than that signature. `lzFundsOutCall` is already in the August report's final commit `ea5de9b4`, and absent from `80485109`. On the history that reaches `c2a5f213` it first appears in [6597381](#) (June 30, 2026, #68). That report attributes the typed entrypoint to [316168fd](#) (June 25, 2026, #73). That commit is not an ancestor of `ea5de9b4` or `c2a5f213`. What appeared on August 18 is the enclave signature, and that signature does not check the network.

The federation signer set only governs configuration through timelocked proposals and does not sign releases. The off-chain signer the August comment counted on is therefore the enclave, and the enclave does not check the network. This is why the accepted WARNING is raised to MAJOR here.

One signed call carries two destination fields, and nothing compares them. The burn records the amount and the recipient, not a chain, so the enclave has nothing to compare the backend's choice against. Neither field is part of `burnId`, so the contract cannot tell that the tokens went somewhere else:

```
// Bridge.sol, fundsOut – commission and route follow destinationChainId
uint256 delivered = _releaseToAdapter(params, adapter);

// MultisigProxy.sol, lzFundsOutCall – delivery follows dstEid
ILZAdapterSendOut(adapter).sendOut{value: msg.value}(
    params.dstEid, params.recipient, delivered, params.minAmountLD, params.extraOptions
);
```

A network that does not exist cannot be reached. An unknown `destinationChainId` reverts with `RouteNotEnabled`, and an unknown `dstEid` reverts with `OftrouteNotConfigured` in the adapter. The call is atomic, so the burn stays unconsumed and can be resubmitted. Any chain the adapter serves is reachable. The adapter uses one registry, `oftByEid`, for inbound and outbound, so every deposit source chain, including Tron if it is enabled, is also a payout destination. The tokens then arrive on chain B at the same address, the commission of chain A is charged, and `burnId` is consumed, so the burn cannot be redirected afterwards. A smart-contract wallet, multisig or exchange deposit address at the same address on B may not belong to the user. A non-EVM chain reads the 20 bytes in its own address format.

The backend gains nothing: the recipient is fixed and the pool is debited once. The loss is that the tokens go to another configured chain the user did not choose.

Proof of Concept

1. A user burns RGB tokens. The burn metadata names the recipient and the amount. It does not name a destination network.
2. The listener builds the enclave request for the LayerZero route. It sets `destinationChainId` to network A and `dstEid` to the LayerZero endpoint of network B. The adapter serves both networks, and neither is the home chain.
3. The enclave validates the consignment, the burned amount and the recipient. For the destination it checks only that `destinationChainId` is non-zero and not the home chain. It compares `dstEid` with nothing, and signs the release for the pair A and B.
4. The signed call is submitted. The contract takes the route and the commission of network A, consumes `burnId`, and the adapter sends the tokens to network B.
5. The user receives the tokens on network B, at the same address, and pays the commission of network A. The burn cannot be sent to network A afterwards, because its `burnId` is already consumed.

If the listener instead sets `dstEid` to a network the adapter does not serve, the enclave still signs, but the adapter rejects the transfer and the burn stays unused.

The steps were traced in the enclave at `dd55d5f` and in the contracts at `c2a5f213` and `be01e61`.

Recommendation

We recommend the check that was missing both from that ACKNOWLEDGED item in the previous reports and from the path added on August 18: bind the delivery chain to the release chain on-chain, in the same way `lzCompose` already checks `eidToChainId[srcEid] == sourceChainId` on the inbound leg. Pass `destinationChainId` to `sendOut` and check it against a governed outbound mapping:

```
// UtxoLZAdapter.sendOut
if (outboundChainIdOf[dstEid] != destinationChainId) revert
DestinationChainIdMismatch(dstEid, destinationChainId);
```

`eidToChainId` is set only together with a trusted entrypoint, so outbound-only destinations need their own entry. As defense in depth, pin the allowed `(destinationChainId, dstEid)` pairs in the attested enclave policy.

Update

Client's response

Fixed in the enclave in commit 082e2a5, merged to dev on 2026-10-16. The 32-byte burn destination is now versioned. A V1 burn commits to the EIP-712 hash of `(destinationChainId, dstEid, recipient)`, and the enclave refuses to sign unless the calldata `destinationChainId`, `dstEid` and recipient all equal that record (`crosscheck.rs`, `validate_funds_out_burn_recipient`). A legacy V0 burn names no chain, so the enclave refuses it on the LayerZero route. The proto, listener and backend changes that carry the record are merged, and the format is described in `docs/burn-destination.md`. The on-chain outbound mapping check in `UtxoLZAdapter.sendOut` and an attested `(destinationChainId, dstEid)` pin are not implemented.

Oxorio's response

Verified at enclave [f658a45](#). `resolve` in `burn_destination.rs` treats a burn whose high 12 bytes are zero as a legacy burn and returns no `dstEid`. `validate_funds_out_burn_recipient` then refuses that burn on `lzFundsOut`, because the calldata `dstEid` is not empty. A V1 record must hash to the burn's `burnReason`, and the calldata `destinationChainId`, `dstEid` and recipient must equal that record. The listener can no longer choose a delivery chain the burn did not commit to. The enclave now binds the signed release to the destination committed by the burn, addressing the reported misdelivery scenario.

Fixed at: [082e2a57](#)

M-02

The mandatory Helios EVM source introduced for M-06 in the previous reports was removed, so the deposit set inside `burnId` again depends on a trusted RPC in `policy.rs`, `events.rs`, `Bridge.sol`

Severity **MAJOR**Status ● FIXED

Location

File	Location	Line
policy.rs	function <code>ProductionPolicy::check_invariants</code>	306-335
events.rs	function <code>verify_rgb_funds_in</code>	501
Cargo.toml	features <code>helios</code> , <code>burn-signer</code>	170-194
Bridge.sol	contract <code>Bridge</code> > function <code>fundsOut</code> (replay guard)	523-524
Bridge.sol	contract <code>Bridge</code> > function <code>_deriveBurnIdFromFields</code>	1191-1212
RgbSettlementModule.sol	contract <code>RgbSettlementModule</code> > function <code>beforeFundsOut</code>	198-218

Description

In the function `verify_rgb_funds_in` of `events.rs`, the enclave takes `(operationId, netAmount)` for every mint of the burn from an EVM receipt returned by its RPC. These pairs become `settlementData`, which is hashed into `burnId`. `burnId` is the only replay guard:

```
// Bridge.sol, _deriveBurnIdFromFields and fundsOut
keccak256(abi.encode(BURN_TYPEHASH, ..., _hashCalldataBytes(settlementData),
sourceBurnTxId));
if (consumedBurnIds[fundsOutParams.burnId]) revert
BurnIdAlreadyConsumed(fundsOutParams.burnId);
consumedBurnIds[fundsOutParams.burnId] = true;
```

`RgbSettlementModule.beforeFundsOut` is `view`. It checks only that each cited record exists and matches its amount and chain. A different deposit set for the same burn gives a different `burnId`, and that `burnId` pays a second time.

The final reports of August 12, 2026 covered this trust:

- ♦ **Solidity M-01** (ACKNOWLEDGED), "burnId is the sole replay guard", was accepted because "double-spend prevention for a burn relies on enclave correctness".
- ♦ **Enclave M-06** (FIXED) closed the enclave side by making Helios mandatory: "a release rgb-validation build now fails the boot gate unless FundsIn deposits are verified via the Helios light client" ([f1be252](#), [7b395f6](#)).

The M-06 fix was removed on August 28, 2026 in [35efb9f0](#) (#199):

```
// policy.rs, ProductionPolicy::check_invariants

-         if self.evm_source != EvmDataSource::HeliosVerified {
-             return Err(format!(
-                 "production policy must verify EVM FundsIn via the trustless Helios path,
not \
-                 {:?}. Build with `--features helios` and set HELIOS_EXECUTION_RPC.",
-                 self.evm_source
-             ));
-         }
+         // The EVM source is not gated: Helios has no Arbitrum light client, so
+         // an L2 image runs on host-relayed RPC.
```

The reason is real: the Bridge runs on Arbitrum, and `HeliosEvmClient::new` accepts only `mainnet`, `sepolia` and `holesky`. Between August 28 and October 1 production accepted raw host-relayed RPC. [#256](#) then refused `RawRpc` again but accepts `PinnedTlsRpc`. The host can no longer forge a receipt, but the RPC operator is trusted, as `events.rs` states: "The RPC operator stays trusted". At the audited commit `helios` is "Kept compilable but unused; not in production" and is not part of `burn-signer`.

Solidity M-01 from those reports therefore relies on an enclave guarantee that no longer exists. The path needs a dishonest RPC answer. An honest deposit tagged with another user's mint costs the attacker at least the payout, because `onFundsIn` records the amount actually paid. The payout is bounded by `lockedLiquidity` and the outflow windows.

Proof of Concept

1. A user burns RGB tokens that were created by mint M. The listener sends the burn to the enclave together with the transaction hash of the EVM deposit behind M.
2. The enclave requests that receipt from its pinned-TLS RPC. The RPC operator returns a receipt whose `FundsIn` log matches M and whose `BridgeFundsIn` record is deposit D1. The enclave puts D1 into `settlementData` and signs. The contract pays and stores the first `burnId`.

3. The same burn is sent to the enclave again. This time the RPC operator returns a receipt for M that cites another existing deposit, D2.
4. The enclave has no light client that could show the receipt is not on the chain, and the burn image does not remember burns it has already signed. It puts D2 into `settlementData` and signs again.
5. The contract derives a second `burnId`, which differs from the first because `settlementData` differs. It finds D2 in the deposit ledger with the right amount and chain, and pays a second time.

The scenario was traced in the enclave at `dd55d5f` and in the contracts at `c2a5f213`.

Recommendation

We recommend a fix that does not depend on the RPC. Helios cannot be restored on Arbitrum. Consume the source burn on-chain in `fundsOut` and `rebalanceLiquidity`; the enclave already binds `sourceBurnTxId` to the burn OpId:

```
if (consumedSourceBurnTxIds[params.sourceBurnTxId]) revert
SourceBurnAlreadyConsumed(params.sourceBurnTxId);
consumedSourceBurnTxIds[params.sourceBurnTxId] = true;
```

We also recommend marking every cited `operationId` as consumed in `RgbSettlementModule.beforeFundsOut`, so that one deposit cannot back two releases. Until this is deployed, Enclave M-06 from the previous reports should be treated as reopened, and the attested policy should state that the RPC operator is trusted for `FundsIn`.

Update

Client's response

Fixed in bridge-smart-contracts PR #172 (0ddf284) and enclave-signer PR #290 (c83b02c). The `burnId` preimage no longer includes `settlementData`. It hashes only the amount, source chain, source address and `sourceBurnTxId`, so one RGB burn has exactly one `burnId`, whichever deposits it cites. A second `fundsOut` or `rebalanceLiquidity` for the same burn reverts with `BurnIdAlreadyConsumed`. The enclave now derives each mint's deposit id from pinned config and the consignment, with no RPC read, and it requires `settlementData` to equal that set. `sourceBurnTxId` is bound to the validated burn OpId (PR #285). We did not add per-`operationId` consumption, because a burn can legitimately cite deposits that other burns also cite. The pinned EVM RPC stays trusted for `FundsIn` receipts on the mint path. This trust is documented in docs/tee-spec.md (F05-NEW-AF-07), and the attested policy pins the RPC host and CA (PinnedTlsRpc).

Oxorio's response

Verified at bridge [e49adf7](#) and enclave [f658a45](#). `_deriveBurnIdFromFields` hashes `amount`, `sourceChainId`, `sourceAddress` and `sourceBurnTxId`. `settlementData` is not in `BURN_TYPEHASH`. A second `fundsOut` or `rebalanceLiquidity` for that id reverts with `BurnIdAlreadyConsumed`. `derived_lock` builds the deposit id from the pinned bridge, chain, token and the mint `OpId`, and `validate_funds_out_settlement` requires `settlementData` to equal that set. `validate_funds_out_source_burn_tx_id` binds `sourceBurnTxId` to the validated burn `OpId`.

Fixed at: bridge [0ddf2842](#), enclave [c83b02cc](#)

M-03

A burn of units minted after 62 deposits cannot be released on this implementation, because settlement must cite every mint in that unit's history and `settlementData` is capped at 4096 bytes in `crosscheck.rs`, `bfa.rs`, `Bridge.sol`

Severity **MAJOR**Status • FIXED

Location

File	Location	Line
crosscheck.rs	function <code>validate_funds_out_settlement</code>	202-250
bfa.rs	function <code>bfa_burn_ancestry_events</code>	163-183
bfa.rs	function <code>bfa_binding</code> (mint chaining)	121-123
sign.rs	settlement check on the burn path	342
Bridge.sol	<code>MAX_SETTLEMENT_DATA_LENGTH</code>	68-79
Bridge.sol	contract <code>Bridge</code> > function <code>fundsOut</code> (length check)	783-785

Description

In the function `validate_funds_out_settlement` of `crosscheck.rs`, the enclave requires `settlementData` to name every mint in the history of the units being burned. `fundsOut` in `Bridge` accepts at most 4096 bytes, which is 62 mints.

The bridge does not stop, and this does not pick out one holder. Units from mints 1 to 62 can still be released, because a burn of mint 5 names only mints 1 to 5. Units from mint 63, and from every mint after it, cannot be released. A person who deposited at mint 10 and again at mint 70 can release the first deposit and cannot release the second. Each later holder is stuck on their own burn. The pool is not debited.

Every BFA mint (`TS_BRIDGE`) spends the mint right `OS_BRIDGE` that the previous mint created and creates a new one, so all mints of the asset form one chain. The enclave states this in `bfa_binding`:

```
// validation/bfa.rs:121-123
/// A mint spends the bridge right from the previous mint. Thus mint N contains
```

```
/// mints 1..N-1, and consensus runs `cea` on each. Each needs its own verified
/// lock.
```

An RGB consignment carries the full history of the allocations it spends, so a burn of units created by deposit N carries all N mints. Those are two different sets. The ancestry set is every earlier mint, and it is present because each mint spends the `OS_BRIDGE` right of the one before. The funding set is only the mints whose `OS_ASSET` units this burn destroys. A burn of one deposit has one funding mint, at any depth, and N ancestry mints. On the burn path, `bfa_burn_ancestry_events` requires a verified EVM receipt for every `TS_BRIDGE` in the consignment, which consensus needs. `validate_funds_out_settlement` then requires `settlementData` to equal that whole ancestry, canonically encoded and strictly ascending, with nothing missing. The contract does not require the ancestry. The enclave is what puts it into the signed list:

```
// server/bfa.rs, bfa_burn_ancestry_events
let plan = binding
    .mint_opids
    .iter()
    .map(|opid| Ok((*opid, ancestor_tx_hash(opid, &source.mint_ancestors)?)))
    .collect::<<Result<Vec<_>>>()>;
```

The contract caps the same field, and its own comment gives the capacity:

```
// Bridge.sol:74-79, fundsOut:783-785
///         bytes for `n` records, so 4096 bytes admits ~62 records – ample
uint256 public constant MAX_SETTLEMENT_DATA_LENGTH = 4096;

if (params.settlementData.length > MAX_SETTLEMENT_DATA_LENGTH) {
    revert SettlementDataTooLong(params.settlementData.length, MAX_SETTLEMENT_DATA_LENGTH);
}
```

The two limits were added for different reasons and never reconciled. The cap is the fix for Solidity R-I-24 in the previous reports, added on 10 July 2026 in [b192ec7](#) ([#122](#)) as a gas bound, when nothing required the list to cover the history, and it is in the August pin `ea5de9b4`. The enclave requirement came later, in three steps: burn ancestry locks on 31 August 2026 in [77d9182](#), verification of every mint in a chained consignment on 2 September in [947b892](#), and the full-ancestry settlement bind on 8 September in [117628a](#). The August enclave baseline `2df6d50` has no `validate_funds_out_settlement`, and the issue is not in the 12 August reports or the 23 September re-audit.

The encoding is exact: `abi.encode(bytes32[], uint256[])` is $128 + 64 * n$ bytes, so 62 records are 4096 bytes and pass `length > MAX_SETTLEMENT_DATA_LENGTH`, and 63 records

are 4160 bytes and revert. For a later mint the enclave refuses a shorter `settlementData`, and the contract reverts the full one, so no signature for that burn can execute. The check is inside `fundsOut`, and `1zFundsOutCall` reaches it through `_releaseToAdapter`, so both routes are affected. The bridge does not stop. Mints 1 to 62 stay releasable, because their ancestry is shorter. Every unit created by mint 63 or later is stuck, each holder on their own burn, and the pool is not debited. The RGB burn is mined before the enclave is asked to sign, so the units are already destroyed when the release fails. Raising the constant only moves the wall: the ancestry of mint N is N records at any cap, and a burn of a late mint cannot be split into a shorter history. The funding set can also exceed 62, but only when this one burn spends units from more than 62 different mints. That bundle can be split. Nothing warns the holder before the burn, because the ancestry length depends on the depth of the mint and not on the size of the transfer. A production bridge passes 62 deposits in normal use, and every deposit after that is in the same position. The same history length also makes each such burn slower, because the enclave fetches and checks one EVM receipt per historical mint.

Proof of Concept

1. The bridge has processed 62 RGB deposits, and each mint spent the mint right of the one before. A holder of units from mint 5 can still be released: that consignment cites mints 1 to 5, which is $128 + 64 * 5 = 448$ bytes.
2. The next depositor receives units from mint 63. The consignment for those units contains mints 1 to 63. The depositor after that, mint 64, contains mints 1 to 64. Both are past the cap.
3. The holder of mint 63 burns the units. The burn is mined on Bitcoin and the units are destroyed.
4. The listener asks the enclave to sign. The enclave verifies 63 receipts and requires a `settlementData` with 63 pairs, $128 + 64 * 63 = 4160$ bytes.
5. A `settlementData` with 62 or fewer pairs is refused by the enclave. The 63-pair one is signed, and `fundsOut` reverts with `SettlementDataTooLong(4160, 4096)`.
6. That burn cannot be released on this implementation. Mint 64 fails the same way, at 4224 bytes. Mints 1 to 62 still pass. The pool is not debited.

The steps were traced in the enclave at `dd55d5f` and in the contracts at `c2a5f213`.

Recommendation

We recommend that `validate_funds_out_settlement` stop building `expected` from every ancestry lock. Keep `bfa_burn_ancestry_events` verifying a receipt for every `TS_BRIDGE` in the consignment, so consensus still runs `cea` on each mint. Cite in `settlementData` only the locks whose mint created an `OS_ASSET` allocation that this `TS_BURN` spends. A mint that is present only because a later mint spent its `OS_BRIDGE` output is not cited. The comparison stays exact: a shorter list, a longer list, and a different deposit are still refused, so a second subset cannot produce a second `burnId`.

A burn of one deposit then cites one pair, 192 bytes, at any depth. The cited set exceeds 62 only when this one burn spends units created by more than 62 different mints. That is the holder's bundle, not the age of the bridge, and it can be split. If $128 + 64 * \text{funding_mints} > 4096$, refuse to sign before anything is broadcast:

```
if 128 + 64 * funding_mints.len() > MAX_SETTLEMENT_DATA_LENGTH {
    return Err(EnclaveError::CrossCheck(
        "burn spends units from more than 62 mints; split it before broadcast".into(),
    ));
}
```

Raising `MAX_SETTLEMENT_DATA_LENGTH` does not fix the ancestry case. The chain outgrows any fixed cap, and a burn of a late mint cannot be split into a shorter history.

Update

Client's response

Fixed in a different way than recommended. A BFA mint now consumes its mint right and passes none on (enclave `c30806e, 9b060b4`; `rgb-lib-bfa 3f571ac`). New rights come only from rights refills. The enclave refuses any mint or refill whose right a mint created (parent proof PP4), so the history of a mint no longer contains earlier mints. The `settlementData` of a burn therefore covers only the mints whose units the burn spends: one pair (192 bytes) for one deposit, at any depth. Separately, `bridge-smart-contracts` PR #172 split the cap and raised the `fundsOut` and `rebalance settlementData` limit from 4096 to 90,000 bytes (`MAX_SETTLEMENT_DATA_OUT_LENGTH`, about 1404 records). The limit applies to both `fundsOut` and `lzFundsOutCall`. Units minted under the old chained form still cite their old ancestry, and now fit up to 1404 records.

Oxorio's response

Verified at enclave [f658a45](#). A mint that passes a mint right on is refused in `psbt_validation/mint.rs`. A parent that is not a transfer is refused in `parent_proof.rs`, so a mint cannot sit in the history of a later mint. `Cargo.lock` pins `rgb-lib-bfa` at `3f571ac`. The library sources were not in the repositories we could read. `Bridge.MAX_SETTLEMENT_DATA_OUT_LENGTH` is 90,000 at [e49adf7](#), and `_validateFundsOutParams` applies it to the release `settlementData`.

Fixed at: enclave [c30806ee](#), [9b060b44](#); bridge [0ddf2842](#)

3.3 WARNING

W-01

`minAmountLD` is not taken from the burned amount in `signing.rs`

Severity

WARNING

Status

• FIXED

Location

File	Location	Line
signing.rs	<code>lz_funds_out_digest</code>	155
MultisigProxy.sol	<code>_releaseToAdapter</code> then <code>sendOut</code>	474

Description

In the function `lz_funds_out_digest` of `signing.rs`, the enclave checks that the proto `min_amount_ld` equals the calldata `minAmountLD`. It does not compare that floor with `MS_BURNED_ASSET`, `fundsOut.amount`, or the net the adapter receives. Zero is accepted.

`lzFundsOutCall` sends `delivered`, the token balance the adapter actually received, and passes `minAmountLD` through as the remote floor. The burn does not commit that floor. A listener can ask the enclave to sign a floor of zero.

The commission is not fixed at signing. The enclave signs the gross amount. `Bridge.fundsOut` asks `commissionManager.calculateFundsOutCommission` at execution time, and the manager is replaceable by governance. The contract enforces `tokenCommission + netAmount == amount` and a nonzero net. It does not bound the commission against the quote that was current when the enclave signed.

Scenario

1. A burn of `1_000e6` is signed with `minAmountLD = 0` and a deadline within `MAX_TEE_DEADLINE`. The signed type string names `amount` and `minAmountLD`. It does not name a commission or a net.
2. At signing, the rule `stablePercent = 100` (1%) quotes a net of `990e6`.
3. Before the call is mined, governance sets the same rule to `stablePercent = 9_000` (90%). `_MAX_FEE_BPS` allows it.

4. `fundsOut` debits the gross `1_000e6` from `lockedLiquidity`, takes the new commission, and releases a net of `100e6` to the adapter. `fee + net == amount`, and the net is not zero, so the contract accepts it.
5. `sendOut` forwards that `100e6` as `delivered`. The signed floor of zero accepts it. The recipient gets `100e6` instead of the `990e6` quoted at signing.

The pool still releases one amount. This is a loss to the recipient, not a second payout.

Proof of Concept

`enclave/src/networks/evm/signing.rs`. The type string names `minAmountLD` and does not name a commission:

```
const TEE_LZ_FUNDS_OUT_TYPE_HASH_STR: &str = "TeeLzFundsOut(uint256 amount,uint256 burnId,\nuint256 sourceChainId,uint256 destinationChainId,string sourceAddress,\nbytes proof,bytes settlementData,uint32 dstEid,bytes32 recipient,\nuint256 minAmountLD,bytes extraOptions,bytes32 sourceBurnTxId,uint256 nonce,\nuint256 deadline)";
```

The only check of the floor compares the `proto` field with the `calldata`. It does not compare the burned amount or the net, so zero passes:

```
let min_amount_ld: u64 = params\n    .minAmountLD\n    .try_into()\n    .map_err(|_| EnclaveError::CrossCheck("lzFundsOut minAmountLD exceeds u64\nrange".into()))?;\nif lz_release.min_amount_ld != min_amount_ld {\n    return Err(EnclaveError::CrossCheck(format!(\n        "lz_release.min_amount_ld {} != calldata minAmountLD {}",\n        lz_release.min_amount_ld, min_amount_ld\n    )));\n}
```

Previously reported. After the audits. The enclave signature of this floor did not exist on 12 August. The field was already in the August Solidity pin, and the closed `sendOut` warning looked at this call without recording a zero floor. That is a warning-level note on a function reviewed for another bug, not a missed second payout. Not in the August 12 reports and not in the 23 September re-audit. `minAmountLD` arrives with `lzFundsOutCall` on 30 June 2026 ([6597381](#), [#68](#)) and is in the August Solidity pin `ea5de9b4`. The enclave starts signing it on 18 August 2026 ([c8633d8](#), [#172](#)), after both August 12 reports. The August enclave baseline `2df6d50` has no `TeeLzFundsOut`. The closed Solidity warning "Outbound `sendOut`

is not bound on-chain to its paired `fundsOut` " bound the delivered token amount. It did not bind this floor to the burn, and it did not freeze the commission quote.

Recommendation

We recommend fixing `minAmountLD` once, when the enclave signs. The number is the net the user was quoted then: the burned amount minus that quote. Do not read `CommissionManager` again to fill it in at execution. Refuse to sign zero when that net is not zero. `sendOut` keeps the signed number as the remote minimum, so a later commission that would pay less reverts instead of delivering the smaller net. Alternatively, put a maximum commission in the signed type, so the quote at execution cannot exceed what was signed.

Update

Client's response

Partially fixed in PR #320 (commit 07f123a). On the LayerZero route, the enclave now refuses a `minAmountLD` of zero and a `minAmountLD` above the burned amount (`validation.rs`, `validate_min_amount_ld`). Unit tests cover both cases. The backend sets `minAmountLD` from the net amount quoted at signing, and `sendOut` enforces it as the remote minimum. The enclave does not compute that net from the commission quote, and the signed type does not carry a maximum commission. Commission changes are timelocked governance operations.

Oxorio's response

Verified at [f658a45](#). `validate_min_amount_ld` refuses a zero `minAmountLD` and a `minAmountLD` above `amount`.

Fixed at: [07f123a2](#)

W-02

The LayerZero native fee is not tied to one release in `gas_tx.rs`

Severity

WARNING

Status

• ACKNOWLEDGED

Location

File	Location	Line
gas_tx.rs	module note	14
gas_tx.rs	value carve-out	128

Description

In `gas_tx.rs`, native value is allowed only for the on-chain `lzFundsOutCall` selector `0x8f6b3031`, only to the pinned proxy, and only up to `GAS_TX_MAX_VALUE_WEI`. The check is stateless. The file says the LayerZero fee is not bound to its release, and that total fee spend across many transactions is not bounded here.

Each signed gas transaction can spend the cap again. This drains the native fee balance. It does not release a second token amount.

Scenario

1. The operator pins `GAS_TX_MAX_VALUE_WEI = 1_000_000`. The gas key signs an EIP-1559 transaction to the pinned proxy. Calldata starts with selector `0x8f6b3031` (`lzFundsOutCall`). Value is 999_999 wei. The enclave accepts it.
2. The same shape with value `1_000_000` is accepted. Value `1_000_001` is rejected. A nonzero value on any other selector is rejected with `value must be 0`.
3. A second transaction of `1_000_000` wei is checked the same way. The check keeps no record of the first. The native fee balance can be spent up to the ceiling again. The bridge token is not released a second time.

Proof of Concept

`enclave/src/networks/evm/gas_tx.rs`. The module comment states that spend across many transactions is not bounded and that the fee is not bound to its release:

```
//! Carve-out to `value == 0`: the payable `lzFundsOutCall` forwards the  
//! LayerZero messaging fee. Allowed only when the selector is `lzFundsOutCall`,  
//! `to` == pinned `BRIDGE_CONTRACT`, and `value` <= `GAS_TX_MAX_VALUE_WEI`.
```

```

//!
//! Not bounded here: total fee spend across many txs (validation is
//! stateless). The LayerZero fee is not bound to its release.

```

The selector and the check. A value above the ceiling is refused. This branch does not name a release:

```

const ONCHAIN_LZ_FUNDS_OUT_CALL_SELECTOR: [u8; 4] = [0x8f, 0x6b, 0x30, 0x31];

if tx.value != 0 {
    if !tx.data.starts_with(&ONCHAIN_LZ_FUNDS_OUT_CALL_SELECTOR) {
        return Err(reject(
            "gas tx: value must be 0 unless calldata is the payable lzFundsOutCall",
        ));
    }
    if tx.to != cfg.bridge_contract {
        return Err(reject(format!(
            "gas tx: non-zero value is only allowed to the pinned MultisigProxy {}, not {}",
            hex::encode(cfg.bridge_contract),
            hex::encode(tx.to)
        )));
    }
    let max = cfg.gas_tx_max_value_wei.ok_or_else(|| {
        reject(
            "gas tx: non-zero value requires a pinned ceiling (GAS_TX_MAX_VALUE_WEI unset) \
            - refusing to sign",
        )
    })?;
    if tx.value > max {
        return Err(reject(format!(
            "gas tx: value {} exceeds pinned GAS_TX_MAX_VALUE_WEI {}",
            tx.value, max
        )));
    }
}

```

This was not run against a live enclave.

Previously reported. A relaxation of an August fix, added after the audits. The August 12 enclave report closed the gas-key drain by requiring `value == 0` (#81). That rule is still in the August baseline `2df6d50`. On 18 August 2026 [c8633d8](#) (#172) and [c4691ba](#) (#174) add one exception: a payable `lzFundsOutCall` may carry value up to `GAS_TX_MAX_VALUE_WEI`,

and that value is not tied to one release. This warning is that exception. It is not in the August reports or the September re-audit.

Recommendation

We recommend binding the signed value to the fee of the release being signed, and add a per-window ceiling on native spend by the gas key.

Update

Client's response

Acknowledged. The native-value carve-out in `gas_tx.rs` is unchanged. Value is allowed only on the `IzFundsOutCall` selector, only to the pinned `MultisigProxy`, and only up to the attested `GAS_TX_MAX_VALUE_WEI` per transaction. The limit is documented in the module header and in `docs/tee-spec.md`. The risk is limited to the native balance of the gas key, and it cannot release bridge tokens.

Oxorio's response

Verified at [f658a45](#). The module header still states that spend across many transactions is not bounded and that the LayerZero fee is not bound to its release. The carve-out is unchanged: value is allowed only for `IzFundsOutCall`, only to the pinned proxy, and only up to `GAS_TX_MAX_VALUE_WEI`. The residual is accepted. It drains the gas key's native balance and does not release bridge tokens.

W-03

`extraOptions` are signed without being parsed in `signing.rs`

Severity

WARNING

Status

• FIXED

Location

File	Location	Line
signing.rs	<code>keccak256(extraOptions)</code>	183
MultisigProxy.sol	<code>sendOut</code> forwards <code>extraOptions</code>	479
UtxoLZAdapter.sol	<code>sendOut</code> , local checkout <code>befd0e6</code> , not the bridge pin	243

Description

In the function `lz_funds_out_digest` of `signing.rs`, `extraOptions` are hashed and not decoded. `burnId` does not contain them. `lzFundsOutCall` forwards the bytes to `sendOut`.

The adapter checkout `befd0e6` is outside the bridge pin. In that file `composeMsg` is empty and `extraOptions` are passed into `OFT.send`, which combines them into the LayerZero options. Compose theft is closed on that checkout. A type-3 executor native drop (worker 1, option type 2, a uint128 amount, a bytes32 receiver) is still signed, and it names an address the burn did not. The native comes from the caller's `msg.value`.

Scenario

1. The listener sets `extraOptions` to a type-3 executor native drop. The bytes are `uint16(3)`, then `uint8(1)`, then `uint16(optionLength + 1)`, then `uint8(2)`, then a `uint128` amount and a `bytes32` receiver. The receiver is an address the burn did not name.
2. One drop carries amount 1 ether to address a. Another carries amount 2 ether to address b. `keccak256` of the two blobs differs.
3. The enclave puts `keccak256(extraOptions)` into the signature and does not decode the bytes. `burnId` is hashed from a type string that has no `extraOptions`, so it does not change.
4. `lzFundsOutCall` forwards the bytes to `sendOut`. On the adapter checkout `befd0e6`, `composeMsg` is empty and the options reach `OFT.send`. The caller of `lzFundsOutCall` pays the native fee, including the drop. The bridged token still goes to the burn recipient.

Proof of Concept

`enclave/src/networks/evm/signing.rs`. `extraOptions` enter the signature only as a hash. `lz_funds_out_digest` has no decoder for option type 2:

```
buf.extend_from_slice(&params.minAmountLD.to_be_bytes::<HASH_LEN>());  
buf.extend_from_slice(&Keccak256::digest(&params.extraOptions));  
buf.extend_from_slice(&params.sourceBurnTxId.0);
```

Previously reported. After the audits. The enclave hashing of these bytes did not exist on 12 August. The bytes were already a parameter of `sendOut`, and the closed warning on that function bound the delivered amount without requiring the options to be decoded. Not as this finding. `sendOut` already took `extraOptions` in the August adapter pin. The enclave starts hashing those bytes, without parsing them, on 18 August 2026 ([c8633d8](#), [#172](#)). The August enclave baseline cannot sign this path. The closed `sendOut` warning bound the delivered amount inside one typed call. It did not require the enclave to decode the options.

Recommendation

We recommend refusing executor options other than a gas limit for `lzReceive`. Keep `composeMsg` empty. Do not sign a native drop.

Update

Client's response

Fixed in PR #320 (commit 07f123a). The enclave now decodes `lzFundsOut` `extraOptions` before it signs (`lz_options.rs`, `validate_extra_options`). It accepts only empty options, or one type-3 executor `lzReceive` gas option with no value and gas capped at 1,000,000. It refuses a native drop, compose, `lzReceive` with value, ordered execution, a non-executor worker and multiple options. Each case has a unit test. The adapter's `sendOut` keeps `composeMsg` empty.

Oxorio's response

Verified at [f658a45](#). `validate_extra_options` accepts empty options, or one type-3 executor `lzReceive` option whose body is 16 bytes of gas and whose gas is at most 1,000,000. A native drop, a compose option, an `lzReceive` value, a second option and a non-executor worker are refused. `validate_destination` calls this check before signing.

Fixed at: [07f123a2](#)

W-04

A signet burn can be signed for a production EVM domain in `spv_crosscheck.rs` and `validation.rs`

Severity

WARNING

Status

• FIXED

Location

File	Location	Line
spv_crosscheck.rs	<code>assert_chain_net</code>	269-279
validation.rs	<code>validate_rgb_source_identity</code>	260-277
policy.rs	<code>SecurityPolicy::attested</code>	235-259

Description

In the function `assert_chain_net` of `spv_crosscheck.rs`, the consignment prefix must match the Bitcoin network this image is running: `mainnet`, `signet`, `testnet3` or `regtest`. `validate_rgb_source_identity` in `validation.rs` only requires `sourceChainId == 96` and an empty `sourceAddress`. The two checks do not meet. `96` is the RGB route id on every Bitcoin network. `validate_destination` then checks the pinned EVM chain and proxy.

Nothing requires the Bitcoin network to be the one that belongs to those EVM pins. An image can be built with `BITCOIN_NETWORK=signet` and with the production proxy, bridge and token. A signet burn matches the signet header chain. The calldata still says source chain `96`. The signature is for the production EIP-712 domain.

`RGB_ASSET_ID` only checks that the consignment is the asset configured in this image. It does not check that the asset's Bitcoin network is the production one.

The attested policy commits `chain_id`, the proxy, the bridge, the token and the asset id. It does not commit `BITCOIN_NETWORK`. The image comment in `spv/types.rs` says the Dockerfile sets that variable, so it is part of PCR0: a signet image is a different measurement from a production image. An outside user cannot change it on a running production image. A check that reads only `user_data` still would not see signet. The release happens only if the federation puts that image's key in the production signer set.

Scenario

1. The image is built with `BITCOIN_NETWORK=signet`, a signet `RGB_ASSET_ID`, and the production `EVM_CHAIN_ID`, proxy, bridge and token.
2. The attestation names the production proxy and token. It does not name signet.
3. A user burns on signet. `assert_chain_net` accepts the consignment because the local header chain is signet. SPV checks that signet chain.
4. `sourceChainId == 96` passes. Destination validation accepts the production proxy.
5. The enclave signs an EIP-712 digest for the production domain.
6. If this key sits in the production signer set, `fundsOut` releases production tokens for a signet burn.

Proof of Concept

The image is built with `BITCOIN_NETWORK=signet` and with the production EVM addresses. A consignment whose prefix is `sb` passes the network check, because the image's network is signet too. The same request with `sourceChainId = 96` and an empty `sourceAddress` passes the source check. The digest is built for the production proxy. Neither function reads the other side. The attestation names `chain_id` and the proxy and does not name signet.

`enclave/src/networks/rgb/spv_crosscheck.rs`. The consignment is compared with this image's network. The EVM pins are not here:

```
pub fn assert_chain_net(consignment_chain_net: &str, enclave_network: Network) -> Result<()>
{
    let chain_net = expected_chain_net(enclave_network);
    let expected = chain_net.prefix();
    if consignment_chain_net != expected {
        return Err(EnclaveError::Spv(format!(
            "consignment chain_net {consignment_chain_net:?} does not match \
            enclave network {enclave_network:?} (expected {expected:?})"
        )));
    }
    Ok(())
}
```

`enclave/src/networks/evm/validation.rs`. The EVM side only knows route `96`. It does not read the Bitcoin network:

```
pub const RGB_SOURCE_CHAIN_ID: u64 = 96;
```

```
if source.source_chain_id != U256::from(RGB_SOURCE_CHAIN_ID) {
    return Err(EnclaveError::CrossCheck(format!(
```

```

        "calldata sourceChainId {} != RGB network id {RGB_SOURCE_CHAIN_ID}: the source chain
        \
        selects the verifier, settlement module and commission rate - refusing to sign an \
        RGB release under a foreign source chain",
        source.source_chain_id
    ));
}

```

enclave/src/policy.rs. `user_data` commits the EVM pins. There is no `BITCOIN_NETWORK` field:

```

Self::Production(p) => AttestedPolicy::Production {
    allow_vanilla_psbt: p.allow_vanilla_psbt,
    signer_role: p.signer_role,
    attestation: p.attestation,
    evm_source: p.evm_source,
    btc_source: p.btc_source,
    chain_id: p.chain_id,
    bridge_contract: p.bridge_contract,
    rgb_asset_id: p.rgb_asset_id.clone(),
    funds_in_contract: p.funds_in_contract,
    token_contract: p.token_contract,
}

```

enclave/src/networks/rgb/spv/types.rs. The Dockerfile sets the network, so it is in PCR0, and it is not in that list:

```

/// Bitcoin network for header validation. It comes from the `BITCOIN_NETWORK`
/// env var. The Dockerfile sets that var, so PCR0 includes it.

```

Previously reported. A close wording is in the enclave report of 12 August, INFO, ACKNOWLEDGED: "Cross-chain metadata is request/calldata-supplied rather than derived from enclave-validated facts". The recommendation already asked to check `validated.chain_net` against the Bitcoin network of the route, and not to equate it with `sourceChainId`. The client deferred that to the canonical bridge intent: chain id and the proxy are pinned, and `sourceChainId` was called informational. The case of a signet image with production EVM pins, and the network absent from `user_data`, is not in that report. The re-audit of 23 September does not have it.

Recommendation

We recommend committing the Bitcoin network in the attested policy next to the EVM pins, and refuse to sign when that network is not the one configured for those pins. A signet image must not carry the production proxy.

Update

Client's response

Fixed in PR #323 (commit 618d793). The attested policy now commits `bitcoin_network` next to `chain_id`, so a verifier that reads `user_data` sees the Bitcoin network of the image. A production policy refuses to boot when `EVM_CHAIN_ID` is a mainnet chain (Ethereum 1 or Arbitrum One 42161) and `BITCOIN_NETWORK` is not mainnet. `attest-verify` checks `--expect-bitcoin-network`, which defaults to `bitcoin`, and the parent launch check compares the attested network with the image environment. A Bitcoin network that the attested encoding cannot represent is refused at boot.

Oxorio's response

Verified at [f658a45](#). `SecurityPolicy` commits `bitcoin_network`. `check_build_invariants` refuses to boot when `chain_id` is 1 or 42161 and `bitcoin_network` is not mainnet. Those are the production chain ids in the check.

Fixed at: [618d793d](#)

W-05 The signature stays valid after the bridge implementation changes in `policy.rs` and `MultisigProxy.sol`

Severity **WARNING**

Status

- ACKNOWLEDGED

Location

File	Location	Line
policy.rs	<code>check_build_invariants</code>	340-354
bfa.rs	<code>bfa_binding_for</code>	148-151
signing.rs	EIP-712 domain	8
MultisigProxy.sol	<code>_buildDomainSeparator</code>	1504
MultisigProxy.sol	<code>upgradeToAndCall</code>	1466

Description

In the function `check_build_invariants` of `policy.rs`, the enclave requires a non-zero chain id, proxy, asset, `FundsIn` contract and token. `bfa_binding_for` checks that the BFA names the pinned `FundsIn` contract. Neither reads the bridge implementation or its runtime code hash.

The EIP-712 domain binds `verifyingContract` to `MultisigProxy`. That address does not change when the federation upgrades the bridge. `UpgradeBridgeImplementation` replaces the logic behind `BridgeProxy` and leaves the proxy address in place. A signature made for the old logic still verifies.

`fundsOutCall` then calls the current implementation with the signed parameters. An implementation installed by that upgrade can ignore the amount and send a larger balance. Checking `extcodehash` of the proxy would not see the change: the proxy bytecode stays the same, and the new logic lives at `implementation()`.

Sending more than the signed amount takes a bridge upgrade. The federation already has that upgrade, behind the timelock. This is a missing bind on a path the federation controls.

Scenario

1. The enclave signs a release of 1 000 against the current implementation. The domain is `MultisigProxy`.
2. The federation executes `UpgradeBridgeImplementation`. The proxy address is unchanged. The new implementation's `fundsOut` does not limit itself to the amount argument.
3. The old signature still matches the domain.
4. `fundsOutCall` calls the new implementation. It can move more than 1 000. The enclave has no implementation hash to refuse.

Proof of Concept

The enclave builds the domain from the name `MultisigProxy`, version `1`, the chain id and the proxy address in the request. The contract builds the same domain from `address(this)`, which is `MultisigProxy` itself. The Bridge implementation hash is not in the domain. `UpgradeBridgeImplementation` calls `upgradeToAndCall` on `BridgeProxy` and does not change the `MultisigProxy` address. A signature for 1 000 made before the upgrade still matches that domain afterwards, and the new implementation can send more.

`enclave/src/networks/evm/signing.rs`. The domain carries the proxy address, not the code hash behind it:

```
const DOMAIN_TYPE_HASH_STR: &str =
    "EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)";

Ok(Eip712Domain {
    name: "MultisigProxy".to_string(),
    version: "1".to_string(),
    chain_id,
    verifying_contract,
})
```

`ethereum/src/MultisigProxy.sol` at `c2a5f213`. `verifyingContract` is the proxy itself:

```
function _buildDomainSeparator() private view returns (bytes32) {
    return keccak256(
        abi.encode(_DOMAIN_TYPEHASH, keccak256("MultisigProxy"), keccak256("1"),
        block.chainid, address(this))
    );
}
```

The same file. The upgrade replaces the logic of `BridgeProxy` and does not change `address(this)`:

```

} else if (opType == OperationType.UpgradeBridgeImplementation) {
    (address bridgeProxy, address newImplementation, bytes memory initializationData) =
        abi.decode(opData, (address, address, bytes));
    _requireValidBridgeUpgrade(bridgeProxy, newImplementation, initializationData);
    IBridgeProxy(bridgeProxy).upgradeToAndCall(newImplementation, initializationData);
}

```

Previously reported. The bind "an enclave signature survives an implementation change because the domain is MultisigProxy" is not in the reports. The re-audit of 23 September has a neighbouring INFO, ACKNOWLEDGED: "A 32-byte answer to `implementation()` is treated as a proxy". Its recommendation is to accept an upgrade only when `extcodehash` of the candidate is an approved hash. The client kept the UUID check and explicitly accepted the residual risk: a code-hash list was not added. `UpgradeBridgeImplementation` is not in the August pin `ea5de9b4`.

Recommendation

We recommend that, before `fundsOut`, `MultisigProxy` require the signed implementation address and `extcodehash` to match `BridgeProxy.implementation()` and the code at that address. The enclave should sign only hashes listed in the attested policy. After an upgrade, releases wait until that policy and its attestation name the new hash.

Update

Client's response

Acknowledged. The EIP-712 domain binds the `MultisigProxy` address, and the typed release messages do not include the Bridge implementation or its code hash. The window is bounded. An implementation upgrade is a timelocked federation operation, and each enclave release signature expires within `MAX_TEE_DEADLINE` (one day). So only signatures issued in the last day before an upgrade executes can reach the new logic, and only through code that the federation approved.

Oxorio's response

Verified at [f658a45](#). The EIP-712 domain is still the name `MultisigProxy`, version `1`, the chain id and the proxy address. The typed release does not include the Bridge implementation or its code hash. The one-day deadline is now enforced by the enclave as well as the contract, so the window is the signatures of the last day before a timelocked upgrade. The residual is accepted.

3.4 INFO

I-01

The enclave does not apply the one-day deadline cap of **MultisigProxy** in **validation.rs**

Severity

INFO

Status

• FIXED

Location

File	Location	Line
validation.rs	deadline check	410-412
MultisigProxy.sol	MAX_TEE_DEADLINE	113

Description

In the function `validate_destination` of `validation.rs`, the enclave rejects a deadline only when it is not strictly after the enclave clock, and it has no upper bound. **MultisigProxy** rejects a deadline more than one day ahead, so the two sides apply different rules to the same field:

```
// validation.rs:410-412
if destination.deadline <= now {
    return Err(EnclaveError::CrossCheck("request deadline expired".into()));
}
```

```
// MultisigProxy.sol:113
uint256 public constant MAX_TEE_DEADLINE = 1 days;
```

The enclave check dates from the first enclave merge on 27 March 2026, [e476e78](#) (#10). `MAX_TEE_DEADLINE` was added to the contract on 18 June 2026 in [6a2f78b](#) (#41), and the enclave was not updated to match.

A deadline more than a day ahead passes the enclave and fails on chain. The contract is the stricter side, so the signature cannot execute and nothing is lost; the effect is a wasted signature and a less clear error for the backend.

Proof of Concept

1. The enclave clock is `T`, and the request deadline is `T + 2 days`.
2. `deadline <= now` is false, so the enclave signs.
3. `fundsOutCall` reverts, because the deadline is past `MAX_TEE_DEADLINE`.

The steps were traced in the enclave at `dd55d5f` and in the contracts at `c2a5f213`.

Recommendation

We recommend refusing `deadline > now + MAX_TEE_DEADLINE` in `validate_destination`, with the constant defined once and documented as equal to the contract value:

```
const MAX_TEE_DEADLINE_SECS: u64 = 86_400;
if destination.deadline > now + MAX_TEE_DEADLINE_SECS {
    return Err(EnclaveError::CrossCheck("request deadline exceeds MAX_TEE_DEADLINE".into()));
}
```

The enclave then refuses exactly the requests the contract would reject.

Update

Client's response

Fixed in PR #320 (commit 07f123a). `validate_destination` now refuses an expired deadline and a deadline more than `MAX_TEE_DEADLINE_SECS` (86,400 seconds) after the enclave clock. The constant is defined once in `validation.rs` and is documented as equal to `MultisigProxy.MAX_TEE_DEADLINE` (1 days). Unit tests cover a deadline just below the cap, a deadline just beyond it, and a deadline far in the future.

Oxorio's response

Verified at [f658a45](#). `MAX_TEE_DEADLINE_SECS` is 86,400. `validate_destination` refuses a deadline that has passed and a deadline more than that many seconds after the enclave clock.

Fixed at: [07f123a2](#)

I-02

The signed nonce is taken from the request rather than from `teeNonce` in `signers.rs`

Severity

INFO

Status

• ACKNOWLEDGED

Location

File	Location	Line
signers.rs	<code>req.nonce</code> into the LayerZero digest	67
signers.rs	<code>req.nonce</code> into the direct digest	80
MultisigProxy.sol	contract <code>MultisigProxy</code> > function <code>fundsOutCall</code> (<code>nonce != teeNonce</code>)	323
MultisigProxy.sol	contract <code>MultisigProxy</code> > function <code>lzFundsOutCall</code> (<code>nonce != teeNonce</code>)	459

Description

In the function `handle_sign_evm` of `signers.rs`, `req.nonce` is put into the digest as supplied by the request, and the burn image does not read the current `teeNonce` for the source chain. The contract accepts only the current counter, in `fundsOutCall` at line 323 and in `lzFundsOutCall` at line 459:

```
// signers.rs:67, LayerZero digest
req.nonce,

// signers.rs:80, direct fundsOut digest
funds_out_digest(&domain, params, req.nonce, req.deadline)?
```

```
// MultisigProxy.sol:323 and :459
if (nonce != teeNonce[params.sourceChainId]) revert InvalidNonce();
```

The current signing path with `req.nonce` dates from 18 August 2026, [fde5ce7](#) (#191). The contract's equality check has existed since 30 June 2026, [6597381](#) (#68).

A signature for a future nonce cannot execute until the counter reaches that value, by which time the one-day deadline may have expired, and a signature for a past nonce never executes. The contract enforces the counter, so this is not a second payout, but the backend can obtain signatures that are useless or that execute later than expected.

Proof of Concept

1. `teeNonce` for the source chain is 4, and the listener asks for nonce 5.
2. The enclave signs nonce 5 without reading the counter.
3. The contract reverts with `InvalidNonce` until the counter becomes 5, and by then the deadline may have passed.

The steps were traced in the enclave at `dd55d5f` and in the contracts at `c2a5f213`.

Recommendation

We recommend reading `teeNonce[sourceChainId]` from the pinned proxy over the RPC the enclave already uses, and refusing a request whose nonce is not that value:

```
let current = evm.tee_nonce(cfg.bridge_contract, params.sourceChainId?);
if req.nonce != current {
    return Err(EnclaveError::CrossCheck(format!("nonce {} != teeNonce {current}",
req.nonce)));
}
```

Every signature the enclave produces is then executable at the current counter.

Update

Client's response

The client set this finding to `ACKNOWLEDGED` and left the response empty.

Oxorio's response

At [f658a45](#), `handle_sign_evm` still puts `req.nonce` into both digests and does not read `teeNonce`. The contract still reverts with `InvalidNonce` when the nonce is not the counter. The residual stands as described.

I-03

The header-chain lock is not held across the signature in `signers.rs`

Severity **INFO**

Status • FIXED

Location

File	Location	Line
signers.rs	pin check, then <code>sign_evm</code>	99-101

Description

In the function `assert_chain_pins_unchanged` of `signers.rs`, the header chain is locked, the Bitcoin blocks used during validation are checked, and the lock is released when the function returns. `sign_evm` runs after that, so the check and the use of the key are not atomic:

```
// signers.rs:99-101
assert_chain_pins_unchanged(ctx, pins)?;
let signature = ctx.state.sign_evm(&digest)?;
```

The re-check was added on 22 September 2026 in [f274bae](#) ([#230](#)) as the fix for F05-NEW-AF-08 in the previous reports. It closed most of the window, but the lock is released before the signature.

A header update that reorganises the pinned blocks and lands between the check and the signature is not prevented by the lock. The window is one signature call, which was not timed, and a reorg deeper than the six required confirmations is also needed, so the practical risk is low.

Proof of Concept

1. The pin check locks the header chain and confirms the pins.
2. The function returns, and the lock is dropped.
3. A header update is applied, and `sign_evm` then signs against blocks that are no longer pinned.

The steps were traced in the enclave at `dd55d5f`. The window was not timed.

Recommendation

We recommend taking the header-chain lock in the caller and holding it until `sign_evm` returns:

```
let chain = ctx.header_chain.lock()?;
pins.assert_unchanged(&chain)?;
let signature = ctx.state.sign_evm(&digest)?;
drop(chain);
```

The check and the signature are then atomic with respect to header updates. Alternatively, document the window as the accepted remainder of F05-NEW-AF-08 in the previous reports.

Update

Client's response

Fixed in PR #332 (commit a2e02f4). `sign_with_chain_pins_unchanged` now takes the header-chain lock. It checks the pinned blocks again under that lock and holds the lock until the key has signed. This applies to both EVM release digests (direct and LayerZero) and to the colored PSBT signing path. Unit tests check that the header chain stays locked while the signer runs, and that a replaced pin causes a refusal before the key is used.

Oxorio's response

Verified at [f658a45](#). `sign_with_chain_pins_unchanged` takes the header-chain lock, rechecks the pins, and drops the lock only after `sign` returns. `handle_sign_evm` and the colored PSBT path both use it.

Fixed at: [a2e02f44](#)

I-04	EVM confirmation depth is measured against a head from the same provider that returned the receipt in <code>events.rs</code>	
Severity	INFO	
Status	<ul style="list-style-type: none"> ACKNOWLEDGED 	

Location

File	Location	Line
events.rs	function <code>check_confirmation_depth</code>	423-428

Description

In the function `check_confirmation_depth` of `events.rs`, the depth of a deposit receipt is its block number subtracted from `provider.get_block_number()`, called on the same provider that returned the receipt:

```
// events.rs:423-428
fn check_confirmation_depth(
    provider: &dyn EvmReceiptProvider,
    receipt_block: u64,
    min_confirmations: u64,
) -> Result<u64> {
    let head = provider.get_block_number()?;
```

A provider that can invent the receipt can also invent the head, so the depth requirement adds no independent assurance of finality. The function was added on 26 August 2026 in [6842b34](#), and it became the only depth source when Helios was removed from production on 28 August in [35efb9f0](#) (#199).

This is the depth half of the Helios MAJOR, not a separate way to take funds: an honest provider reports both values correctly, and a dishonest one can already forge the receipt. It does mean that `EVM_MIN_CONFIRMATIONS` should not be read as protection against the RPC operator.

Proof of Concept

1. The RPC returns a receipt at block 100 and a head of 106.
2. The function computes a depth of 6 and accepts it against a minimum of 6.

3. Both numbers came from the same provider, and no second source is consulted.

The steps were traced in the enclave at `dd55d5f`.

Recommendation

We recommend taking the head from a source independent of the receipt provider, for example a second pinned RPC operator, and requiring both to agree within a small margin:

```
let head = min(primary.get_block_number()?, secondary.get_block_number()?);
```

If that is not possible, the attested policy should state that the RPC operator is trusted for both receipts and depth.

Update

Client's response

In commit 173155e, the RPC's safe or finalized block tag replaced the confirmation-count check. The enclave also checks that the receipt block is canonical and that the tagged head does not change during verification. Production refuses the latest tag (91a48cc). All of these values still come from the one pinned TLS RPC, so the RPC operator is trusted for both receipts and finality. docs/tee-spec.md (F05-NEW-AF-07) documents this as an explicit design assumption, and the attested policy pins the RPC endpoint and the finality tag.

Oxorio's response

Verified at [f658a45](#). The receipt, the canonical block and the finality head are read from the one pinned TLS RPC. Production policy refuses `EVM_FINALITY_TAG=latest`. The tag change does not add a second source. The residual is accepted.

I-05

Enclave finality thresholds are not aligned with the **RGBVerifier** thresholds in **crosscheck.rs** and **RGBVerifier.sol**

Severity **INFO**

Status • ACKNOWLEDGED

Location

File	Location	Line
spv_crosscheck.rs	SPV_MIN_CONFIRMATIONS = 6	43
crosscheck.rs	relay lag equals MAX_REORG_DEPTH	595
RGBVerifier.sol	contract RGBVerifier > immutable thresholds	70-79

Description

In **spv_crosscheck.rs**, the enclave requires the burn witness to be 6 blocks deep in its own header chain and accepts a relay tip up to 100 blocks behind. **RGBVerifier** checks the same BtcRelay proof against three immutable constructor values that the enclave never reads. The two sides decide independently how deep a Bitcoin burn must be:

```
// spv_crosscheck.rs:43, crosscheck.rs:595
pub const SPV_MIN_CONFIRMATIONS: u32 = 6;
const MAX_RELAY_TIP_LAG_BLOCKS: u32 = crate::networks::rgb::spv::chain::MAX_REORG_DEPTH;
```

```
// RGBVerifier.sol:70-79
uint256 public immutable minSourceConfirmations;
uint256 public immutable maxLatestConfirmations;
uint256 public immutable minConfirmationGap;
```

SPV_MIN_CONFIRMATIONS = 6 dates from 14 May 2026, [c909227](#) (#31), and the relay-lag constant from 28 August 2026, [35efb9f0](#) (#199). The **RGBVerifier** thresholds were introduced in the audit-fix stack on 14 July 2026, [3a7eb76](#) (#127).

If the deployed contract values are stricter, the enclave signs proofs the contract rejects; if they are looser, the enclave is the stricter side. In neither case can a shallower burn be paid,

and a rejected signature does not consume `burnId`, so the effect is wasted signatures and delayed releases.

Proof of Concept

1. `RGBVerifier` is deployed with `minSourceConfirmations = 12`.
2. A burn is 8 blocks deep. The enclave sees a depth of 8, which is above 6, and signs.
3. `verify` reverts with `InsufficientSourceConfirmations(8, 12)`, so `fundsOut` reverts and `burnId` is not consumed.

The steps were traced in the enclave at `dd55d5f` and in the contracts at `c2a5f213`.

Recommendation

We recommend pinning the deployed `RGBVerifier` thresholds in the attested enclave policy and applying them before signing:

```
if depth < max(SPV_MIN_CONFIRMATIONS, cfg.rgb_verifier_min_source_confirmations) {  
    return Err(EnclaveError::Spv("burn witness below the RGBVerifier threshold".into()));  
}
```

The enclave then refuses every proof the contract would reject. If the thresholds are meant to differ, document that the contract values must never be stricter than the enclave values.

Update

Client's response

Partially addressed. The Bitcoin witness depth of the enclave is no longer a hardcoded constant. `SPV_MIN_CONFIRMATIONS` sets it, the attested policy commits it as `spv_min_confirmations`, and in production it must be at least 1. The burn image sets 6, which is the `RGBVerifier` deployment default for `minSourceConfirmations`. The enclave does not mirror the relay freshness and gap rules of `RGBVerifier` (`maxLatestConfirmations`, `minConfirmationGap`). If the contract rejects a proof, `fundsOut` reverts without consuming `burnId`, so the only effect is a wasted signature.

Oxorio's response

Verified at [f658a45](#). `spv_min_confirmations` is part of the attested policy, and a production policy with zero confirmations does not boot. The client states that the burn image sets 6. The enclave still does not read `maxLatestConfirmations` or `minConfirmationGap`. A proof the contract rejects does not consume `burnId`. That residual is acknowledged.

4. APPENDIX

4.1 SECURITY ASSESSMENT METHODOLOGY

This engagement used manual source-code review and code-level tracing of attack scenarios at enclave `dd55d5f`, bridge `c2a5f213` and adapter `be01e61`. Remediations were reviewed at enclave `f658a45` and bridge `e49adf7`. The scenarios and remediations were not executed against a running enclave, and no large-mint-history consignment was constructed. Remediation verification in this report is based on source-code review.

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 smart contract'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 <code>call</code> or <code>delegatecall</code> , 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

OXERIO