CERTIK

Autheo - Chain Audit

Security Assessment

CertiK Assessed on May 11th, 2026

CertiK Assessed on May 11th, 2026

Autheo - Chain Audit

The security assessment was prepared by CertiK.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Layer 1	Cosmos (ATOM)	Manual Review

LANGUAGE	TIMELINE
Go	Preliminary comments published on 04/29/2026 Final report published on 05/11/2026

Vulnerability Summary

51	44	0	7	0
Total Findings	Resolved	Partially Resolved	Acknowledged	Declined

0	Centralization		Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
5	Major	4 Resolved, 1 Acknowledged	Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
14	Medium	14 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
20	Minor	17 Resolved, 3 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
12	Informational	9 Resolved, 3 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | AUTHEO - CHAIN AUDIT

CODEBASE | AUTHEO - CHAIN AUDIT

Repository

https://github.com/autheo-blockchain/autheo-chain-core/tree/823e560c3d0dd13ec76cfcd1e42dd7f94a53baa7

Commit

823e560c3d0dd13ec76cfcd1e42dd7f94a53baa7

Audit Scope

The file in scope is listed in the appendix.

APPROACH & METHODS | AUTHEO - CHAIN AUDIT

This report, prepared for Autheo, identifies issues and vulnerabilities within the Autheo - Chain Audit project's source code. The assessment included a comprehensive review of the in-scope chain codes. A thorough examination was conducted using a combination of Manual Review.

The auditing process pays special attention to the following considerations:

- Testing compiled code against both common and uncommon attack vectors.
- Evaluating the codebase to ensure adherence to current best practices and industry standards.
- Verifying implementation logic to align with client specifications and intentions.
- Comparing code structure and implementation to similar projects from industry leaders, including mainstream blockchain protocols such as Ethereum, Cosmos, and Polkadot.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment identified findings across a spectrum, from informational to critical. We advise addressing these to uphold high security standards and industry practices. Our recommendations aim to enhance the project's security posture:

- Enhance general coding practices for better structures of source codes.
- Incorporate a sufficient number of unit tests to encompass all potential use cases.
- Ensure every function is well commented, especially for core logic to enhance readability.
- Provide more transparency on privileged activities once the protocol is live.
- Continuously monitor the chain project's codebase throughout its lifecycle, specifically after launch and with each merged commit.

SYSTEM OVERVIEW | AUTHEO - CHAIN AUDIT

Autheo is a Cosmos SDK + Ethermint Layer 1 blockchain forked from Cronos whose native token (`utheo`) powers a permissioned, NFT-gated validator economy. The right to operate or be delegated to a validator is intended to be controlled by ERC-721 licenses issued on the EVM side and mirrored on the Cosmos side, while a custom emissions engine handles staking inflation, community-pool funding, and per-license rewards subject to configurable caps.

On top of the Cronos baseline, the fork introduces an `app/` wiring layer and a set of new Cosmos modules — broadly grouped into a licensing module (data model and lifecycle of the on-chain license records), a licensed-staking wrapper (which gates the standard staking flows and keeps validator jail/unjail transitions in sync with license state), an emissions module (which manages the reward lanes and their caps), and a license-distribution module (which mediates the claim path for license-accrued rewards). The `app/` layer also introduces a refreshed blocklist pipeline, EVM hooks and precompiles that bridge selected EVM activity into the Cosmos-side modules, and a number of platform-level upgrades (notably a newer Cosmos SDK and IBC-go line, alongside changes to mempool, execution, and storage).

This audit was conducted as a diff review of the `autheo-chain` repository at commit `823e56` against the fork commit of the `cronos` repository at `e1d819c`. For a detailed view of the files included, please review the `Scope` section of this report.

REVIEW NOTES | AUTHEO - CHAIN AUDIT

Out-of-Scope Dependencies

This engagement was conducted as a diff audit of `autheo-chain-core` against its Cronos fork base. Any file, module, or dependency not explicitly listed in the Scope section was assumed to be functionally equivalent to the corresponding Cronos baseline and was treated as a black box, with its functional correctness assumed. This includes the upstream Cosmos SDK and Ethermint stack, the IBC-go modules, the storage and execution layers, the modules kept unchanged from the fork, the auxiliary client / RPC / simulation / migration code, and all third-party libraries on which the chain depends — none of which were re-audited as part of this engagement.

Testing and Documentation

The repository ships with a substantial Go unit-test suite, organized in a testify-suite style, covering the keepers and message servers of the new modules as well as the bulk of the rebranded `x/autheo` surface. Alongside this, a Python `integration_tests/` pytest suite exercises the inherited platform flows (IBC, ICA, EVM, governance, mempool, upgrade, etc.). Coverage is broadly scenario-based and tends to follow the happy path; edge-case interactions across modules, partial-failure paths, and genesis import/export round-trips of the new cap- and cycle-accounting state are only lightly exercised, and the integration suite does not yet include scenarios that target the new licensing, emissions, or jail-interceptor flows end-to-end. Inline code comments document most of the non-obvious control-flow choices, but a higher-level specification of the licensing lifecycle, the cap-accounting state machine, and the trust model of the encrypted blocklist is missing or incomplete; producing such documentation is recommended to improve maintainability and ease integration.

Code Clarity and Completeness

The in-scope code contains a number of `TODO` annotations as well as unused or unreachable artifacts (dead code, unread parameters, and fields whose names do not reflect their actual semantics). It is recommended to either complete or remove these to improve maintainability and reduce the risk of dangling paths being re-enabled without proper review.

FINDINGS | AUTHEO - CHAIN AUDIT

51	0	0	5	14	20	12
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared for Autheo to identify potential vulnerabilities and security issues within the reviewed codebase. During the course of the audit, a total of 51 issues were identified. Leveraging a combination of Manual Review the following findings were uncovered:

ID	Title	Category	Severity	Status
ACA-01	Emissions Module BeginBlocker Panics On Any Error, Halting The Chain	Volatile Code	Major	Resolved
ACA-02	Emissions Cap Drain Via Single Governance Proposal	Volatile Code, Coding Issue	Major	Resolved
ACA-03	Dual Transfer Authorities (`MsgTransferLicense` And ERC-721 `transferFrom`) Cause Bidirectional Ownership Drift Between X/License And NFT State	Coding Issue	Major	Acknowledged
ACA-04	Permissionless IBC Relay Allows Untrusted Relayers To DoS All Inbound ICS-20 Transfers Via Trusted-Relayer Check	Access Control	Major	Resolved
ACA-05	Blocklist Refresh Fails Open On Broken Validator E2ee Key	Coding Issue	Major	Resolved
ACA-06	Relayer Precompile RequiredGas Uses Panics On Malformed Or Unknown Calldata	Coding Issue	Medium	Resolved
ACA-07	Proposal-Level EIP-7702 Blocklist Check Uses A Different Address Encoder	Coding Issue, Volatile Code	Medium	Resolved
ACA-08	Eth_createAccessList RPC Method Is Not Implemented	Coding Issue	Medium	Resolved
ACA-09	No Support For Historical Block Hash Retrieval Beyond Native BLOCKHASH (EIP-2935)	Coding Issue	Medium	Resolved
ACA-10	BeginBlocker Iterates All Historical License IDs On Every Block	Logical Issue	Medium	Resolved

ID	Title	Category	Severity	Status
ACA-11	StakingMintedTotal Counter Missing From Genesis Import/Export	Coding Issue	Medium	Resolved
ACA-12	Per-Owner Sovereign-License Unenforced	Volatile Code, Coding Issue	Medium	Resolved
ACA-13	ExportGenesis Uses Map Iteration — Produces Non-Deterministic Output Across Validators	Coding Issue, Volatile Code	Medium	Resolved
ACA-14	Staking Reward Inflation: Integer Division BlocksPerYear / 12 Can Create Extra Monthly Cycles	Logical Issue	Medium	Resolved
ACA-15	Relayer Precompile Run Returns Raw Protobuf Bytes Where The ABI Declares Encoded Bytes	Coding Issue	Medium	Resolved
ACA-16	Blocklist Enforcement Does Not Unwrap Authz MsgExec, Allowing Blocklisted Granters To Act Through Grantees	Coding Issue	Medium	Resolved
ACA-17	OnPacketResult Silently Treats Failed EVM IBC Callbacks As Success	Coding Issue	Medium	Resolved
ACA-18	Token Mappings Do Not Verify Contract Code Exists	Coding Issue	Medium	Resolved
ACA-19	Critical Forked Dependencies Lack Concrete Version Tracking	Coding Issue	Medium	Resolved
ACA-20	NFT Reward Dust Consistently Assigned To Highest Active License ID	Coding Issue	Minor	Resolved
ACA-21	Community Remainder Is Not Recomputed When Block Rewards Reaches Maximum Emission	Coding Issue	Minor	Resolved
ACA-22	MsgStoreBlockList Blob Has No Size Limit	Denial of Service, Access Control	Minor	Resolved
ACA-23	Placeholder Upgrade Handler `v1.6` Pre-Registered On A Pre-Launch Chain — Last-Writer-Wins Silent-Migration Risk	Coding Issue	Minor	Resolved

ID	Title	Category	Severity	Status
ACA-24	Address Conversion Functions In Event Decoders Silently Produce Invalid Addresses On Parse Failure	Coding Issue	Minor	Resolved
ACA-25	Relayer Precompile ABI Not Updated For Ibc-Go V10, Breaking EVM Event Emission For Token Packets And Timeout Refunds	Coding Issue	Minor	Resolved
ACA-26	ICA Precompile SubmitMsgs Silently Truncates Uint256 Timeout To Uint64, Causing Unintended Packet Expiry	Logical Issue	Minor	Resolved
ACA-27	ERC-721 Transfer Handler Reverts Entire EVM Transaction On Uint256 Token IDs, Breaking Standards-Compliant License Contracts	Logical Issue	Minor	Acknowledged
ACA-28	Token Mapping Keeper Missing Invariant Guards In SetExternalContractForDenom, SetAutoContractForDenom And GetDenomByContract	Coding Issue	Minor	Resolved
ACA-29	ICA And Relayer Precompiles Panic On Short Or Malformed Calldata	Coding Issue	Minor	Resolved
ACA-30	Relayer Precompile Panics On String-Argument Selectors Due To Unconditional Args[0].([]Byte) Assertion	Coding Issue	Minor	Resolved
ACA-31	TransferTokens Emits Requested EvmDenom Instead Of Rounded Burned/Bridged Amounts	Coding Issue	Minor	Resolved
ACA-32	Emissions GetParams Returns Unvalidated Params And Defaults When ParamsKey Is Absent	Coding Issue	Minor	Resolved
ACA-33	TrustedRelayer Check Conflates Authorized Signer With Voucher Beneficiary, Breaking MsgConvertVouchers	Coding Issue	Minor	Resolved
ACA-34	X/License DestEvmTokenMap Index Not Serialized In Genesis Import/Export Enabling Duplicate License Mints On Genesis Import	Coding Issue	Minor	Resolved

ID	Title	Category	Severity	Status
ACA-35	Sovereign License Loops Use First-Match-Wins Semantics In Both Unjail And EditValidator Checks	Coding Issue	Minor	Resolved
ACA-36	EditValidator O(N) License Scan Spammable Via Unlimited Permissionless Prime/Core Bindings	Coding Issue	Minor	Acknowledged
ACA-37	Token Mapping Governance Proposals Lack Input Validation And Contract-Code Existence Check	Coding Issue	Minor	Resolved
ACA-38	Sovereign License Lifecycle Transitions Skip Validator Status Checks (Bind + Reinstate)	Coding Issue	Minor	Acknowledged
ACA-39	Blocklist Validation Skips Cosmos Bank, IBC, And Authz Recipients	Coding Issue	Minor	Resolved
ACA-40	Emission Param `Enabled` Is Not Consistently Applied To Token Mint Pathways	Volatile Code	Informational	Resolved
ACA-41	OnRecvVouchers Conversion Failures Are Silently Swallowed Without Event Emission	Coding Issue	Informational	Resolved
ACA-42	Payable Relayer Precompile Can Permanently Trap EVM-Native Token	Coding Issue	Informational	Resolved
ACA-43	Emissions GetEmittedTotal Treats Unparsable EmittedTotal Bytes As Zero	Coding Issue	Informational	Resolved
ACA-44	ERC-721 Transfer On License Contracts Reverts Entire Tx When Token ID Exceeds Uint64	Coding Issue	Informational	Acknowledged
ACA-45	Revoked Prime/Core Delegators Keep Receiving Standard Delegator Rewards When Unbonding Queue Is Full	Coding Issue	Informational	Acknowledged
ACA-46	Dormant BeginRedelegate License-Rebind Path Has Latent Defects If Re-Enabled	Coding Issue	Informational	Acknowledged
ACA-47	Licensedstaking ValidateGenesis Drops Upstream SDK Validator Invariants	Coding Issue	Informational	Resolved

ID	Title	Category	Severity	Status
ACA-48	Keeper Token Mapping Trusts AutheoAdmin Without On-Chain Contract-Semantics Validation	Coding Issue	Informational	Resolved
ACA-49	E2ee InitGenesis Skips GenesisState.Validate; Malformed X25519 Keys Stored Verbatim	Coding Issue	Informational	Resolved
ACA-50	Mempool TxReplacement Int64 Overflow Allows Fee-Bump Bypass	Coding Issue	Informational	Resolved
ACA-51	E2ee Keygen Silently Overwrites Existing Identity	Coding Issue	Informational	Resolved

ACA-01 Emissions Module BeginBlocker Panics On Any Error, Halting The Chain

Category	Severity	Location	Status
Volatile Code	Major	x/emissions/abci.go: 11; x/emissions/keeper/block_rewards.go: 62; x/emissions/keeper/nft_rewards.go: 92	Resolved

Description

The emissions `BeginBlocker` converts any error from `MintBlockRewards` or `MintNFTBlockRewards` into `panic(err)`. Unlike transaction-scoped panics (which `baseapp.runTx` recovers and converts into failed transactions), BeginBlocker executes outside `runTx`. A panic there aborts `FinalizeBlock` on every honest validator simultaneously, halting the chain.

Reachable error sources include `stakingKeeper.BondDenom`, `bankKeeper.MintCoins`, `distrKeeper.FundCommunityPool`, and `bankKeeper.BurnCoins` in the NFT rollback path. Today these are unlikely to fail in production, but any future change that adds a keeper call, a precondition check, or an invariant assertion to the emission paths turns a per-block error into an unrecoverable halt. The `panic(err)` posture is appropriate for true invariant violations but not for ordinary operational errors.

Proof of Concept

Vulnerable panic surface (`x/emissions/abci.go:11-23`):

```
func BeginBlocker(ctx sdk.Context, k keeper.Keeper) {
    if err := k.MintBlockRewards(ctx); err != nil {
        panic(err)   // ← BeginBlocker panic = chain halt (no runTx recover)
    }
    if err := k.MintNFTBlockRewards(ctx); err != nil {
        panic(err)
    }
}
```

`baseapp.go` in the replaced SDK fork wraps only `runTx` with a deferred recover — BeginBlocker and EndBlocker propagate panics upward into `FinalizeBlock`, which crashes the node.

Recommendation

Distinguish invariant violations from operational errors. For operational failures (bank mint, community-pool fund, bond denom lookup), log the error, emit an event, and skip that block's emissions rather than halting. Keep `panic` only for state that cannot be safely continued (e.g. corrupted store reads). Add a metrics counter for skipped-emission blocks so operators are alerted.

Alleviation

[Autheo, 05/03/2026]: Hi, this issue has been acknowledged and fixed in commit

c63a090f3cf28578032b29ac07c48def5269dcc5

[CertiK, 05/05/2026]: The team heeded the advice and resolved the issue by replacing the unconditional `panic(err)` in `MintBlockRewards` and `MintNFTBlockRewards` (`x/emissions/abci.go`) with a `Logger().Error(...)` call and an emitted `emission_failure` event, so an inflation error degrades that block's emission instead of halting consensus while staying observable on-chain, in commit c63a090.

ACA-02 | Emissions Cap Drain Via Single Governance Proposal

Category	Severity	Location	Status
Volatile Code, Coding Issue	Major	app/staking_mint_inflation.go: 46; x/emissions/abci.go: 11; x/emissions/keeper/block_rewards.go: 44; x/emissions/keeper/msg_server.go: 25; x/emissions/types/nft_emissions_params.go: 90; x/emissions/types/params.go: 21	Resolved

Description

A single governance proposal can mint the entire remaining emissions cap in one block. `MsgUpdateParams` accepts arbitrarily large per-block / per-tier emission values because `Params.Validate()` only rejects nil/negative amounts — no upper bound exists.

When the resulting per-block mint exceeds the remaining cap, the remainder minting logic at `block_rewards.go:45-58` substitutes `remaining_cap` for the requested amount and mints it in a single block. The subsequent cap-reached short-circuit then permanently disables all future community emissions on that path.

The attack surface is broader than the community-pool path alone. `CommunityPerBlockUtheo`, the NFT tier params (`SovereignAnnualUtheo`, `PrimeAnnualUtheo`, `CoreAnnualUtheo`), and the staking caps (`StakingMonthlyCapUtheo`, `StakingHardCapUtheo`) all share the same unbounded-validation pattern. Because the community and NFT paths share `emittedTotal`, a single proposal on any of these parameters kills both schedules at once.

Impact: a passing governance proposal irreversibly destroys the affected emissions schedule, concentrates the reward budget at a single block height, and produces a supply shock that no further on-chain action can reverse.

Proof of Concept

Vulnerable code (`x/emissions/keeper/block_rewards.go:44-58`):

```
if emittedTotal.Add(totalToMint).GT(params.EmissionsCap) {
    remaining := params.EmissionsCap.Sub(emittedTotal)
    if remaining.IsPositive() {
        communityAmount = remaining  // mints entire remaining cap in ONE block
        totalToMint = remaining
    } else {
        return nil
    }
}
```

Steps to reproduce:

1. Boot a 2-node testnet.

2. Submit a `MsgUpdateParams` governance proposal carrying: `community_per_block_utheo = 10^76` (other params unchanged).
3. Vote yes from the validator (>50% voting power).
4. Wait for the 30-second voting period to elapse.

Observed result on the testnet:

```
Baseline:
  emissions_cap   : 2 975 000 000 000 000 000 000 000 000  utheo
  community_pool  :         6 623 926 945 849 695 003 419  (0.00022% of cap)
Post-proposal (38 blocks later):
  community_pool  : 2 975 000 026 867 160 848 733 739 935  (100.00% of cap)
  Δ pool          : 2 974 993 402 940 214 999 038 736 516
```

Node log from the attack block confirms the clamp branch fired:

```
INF minting would exceed cap, adjusting module=x/emissions
    remaining=297496098902841196520000000

requested=10000000000000000000000000000000000000000000000000000000000000000000000000000000
000
```

After this block, every subsequent block logs `emissions cap reached` and no further community rewards are minted.

Recommendation

Add upper bounds to `Params.Validate()` and `NftEmissionsParams.Validate()` for every mintable per-block or per-tier value, derived from a sane fraction of the emissions cap (for example, `EmissionsCap / DefaultBlocksPerYear`). Add a joint cross-path invariant so the sum of community, NFT, and staking annualised flows cannot exceed the cap within a plausible schedule horizon. Independently cap the amount that `MintBlockRewards` and `MintNFTBlockRewards` can mint in a single block at the keeper level, so a future bug in parameter validation cannot drain the schedule. Consider rate-limiting governance changes to emission parameters with a per-proposal delta bound.

Alleviation

[Autheo, 05/03/2026]: Hi, this issue has been acknowledged and fixed in commit

c63a090f3cf28578032b29ac07c48def5269dcc5

[CertiK, 05/05/2026]: The team heeded the advice and resolved the issue by adding upper-bound checks to `Params.Validate()` in `x/emissions/types/params.go` (`CommunityPerBlockUtheo ≤ EmissionsCap / BlocksPerYear`, every NFT-tier annual ≤ `EmissionsCap`, `StakingHardCapUtheo ≤ EmissionsCap`) and a runtime safety clamp in `block_rewards.go` and `nft_rewards.go` that caps per-block mints against the running emitted total, in commit c63a090.

ACA-03 Dual Transfer Authorities (`MsgTransferLicense` And ERC-721 `transferFrom`) Cause Bidirectional Ownership Drift Between X/License And NFT State

Category	Severity	Location	Status
Coding Issue	Major	app/app.go: 571; integration_tests/contracts/contracts/AutheoLicense.sol: 164; x/license/keeper/evm_hooks.go: 101; x/license/keeper/keeper.go: 396; x/license/keeper/msg_server.go: 178	Acknowledged

Description

The system currently has two independent ownership transfer planes for the same economic object (a license): (1) Cosmos plane: `MsgTransferLicense` updates `license.Owner` and owner index in x/license. (2) EVM plane: ERC-721 `transferFrom` / `safeTransferFrom` updates NFT ownership and emits `Transfer` .

There is no single source of truth and no strict synchronization invariant between these planes. `MsgTransferLicense` has no guard that the license is non-EVM-backed and no call-out to enforce a matching ERC-721 transfer. In the opposite direction, the EVM hook path for existing `(contract, tokenId)` mappings returns early without applying owner updates. As a result, ownership can drift in both directions, breaking authority assumptions for bind/unbind/revoke/reward rights because x/license uses `license.Owner` for authorization while wallets and OTC markets may treat ERC-721 `ownerOf` as canonical ownership.

Proof of Concept

Handler ignores `from` (`x/license/keeper/evm_hooks.go:82-84`):

```
// Extract to address (from address is in topics[1] but we don't need it for
minting)
toBytes := topics[2].Bytes()
to := common.BytesToAddress(toBytes[12:])
```

Keeper returns early on existing mapping (`x/license/keeper/keeper.go:370-387`):

```
if licenseId, found := k.GetLicenseIdByEvmToken(ctx, evmContract, evmTokenId); found
{
    k.Logger(ctx).Warn("License already exists for EVM token, skipping mint", ...)
    return licenseId, nil   // drops ownership change
}
```

Live reproduction :

```
mint tokenId=1 → alice; license id=1, Owner=autheo1...pwumgwek
transfer 1 alice→bob (ERC-721)
ERC-721 ownerOf(1) = 0x...b0b0              (bob)
x/license Owner    = autheo1...pwumgwek     (still alice)
→ DIVERGENCE CONFIRMED
```

Minimal divergence sequence:

1. Mint ERC-721 token to A (hook mints license with owner A, status ISSUED).
2. A submits MsgTransferLicense to transfer license to B (allowed for ISSUED licenses).
3. x/license owner is now B; ERC-721 ownerOf(tokenId) is still A unless a separate EVM transfer is executed.
4. Rights keyed off x/license ownership are now held by B while market-visible NFT ownership remains A.

Recommendation

Establish a single canonical ownership authority and enforce it at module boundaries. Option A (recommended for tradable ERC-721): make ERC-721 owner canonical for EVM-origin licenses; block MsgTransferLicense for EVM-backed licenses (or require proof of matching ERC-721 transfer), and extend EVM hooks to process transfer/burn explicitly. Option B (recommended for Cosmos-canonical rights): make license NFTs non-transferable (soulbound) and only allow ownership changes via x/license messages. In both options, add an invariant test: for every mapped `(contract, tokenId)`, x/license owner must equal the canonical owner source. Alternatively we invite the dev team to clarify the intended design.

Alleviation

[Autheo, 05/10/2026]: Thank you for the clarification. Yes, the actual intended design is the first case you described.

The EVM-side NFT on Autheo is used only as a bridge-ingress representation for licenses arriving through Hyperlane and as a trigger for creating the canonical Cosmos-side license record in x/license. After the license is created in x/license, license ownership and all license-related permissions are canonical in the Cosmos module only.

The EVM-side NFT is not intended to be an independently tradable ownership object on Autheo. User-initiated EVM-side transfers are locked at the Solidity layer, including:

- transferFrom
- safeTransferFrom(address,address,uint256)
- safeTransferFrom(address,address,uint256,bytes)

Users cannot transfer these license NFTs through MetaMask, marketplaces, approvals, or operator-mediated ERC-721 transfers.

The Hyperlane relayer is only authorized to perform the bridge-ingress mint flow. The relayer-triggered mint creates the EVM-side representation and then the corresponding x/license record. After this point, all ownership changes happen only through x/license.

Regarding bridge-back/burn: the EVM-side burn / bridge-back path is not exposed as an independent user-controlled transfer authority. Users cannot use the EVM NFT owner state to burn or bridge back a license independently of x/license.

Any bridge-back or withdrawal flow, if enabled, is intended to be authorized from the canonical x/license owner state, not from ERC-721 ownerOf.

Therefore, the apparent mismatch after a Cosmos-side MsgTransferLicense is acceptable under the intended design because the EVM-side NFT is non-transferable and non-authoritative after ingress. The canonical owner for bind, unbind, revoke, reward, transfer, and bridge-related authorization is always x/license.Owner.

[Autheo, 05/11/2026]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement, because the mechanism is the intended behavior.

ACA-04 Permissionless IBC Relay Allows Untrusted Relayers To DoS All Inbound ICS-20 Transfers Via Trusted-Relayer Check

Category	Severity	Location	Status
Access Control	Major	x/autheo/middleware/conversion_middleware.go: 97~135	Resolved

Description

`IBCConversionModule.OnRecvPacket` uses the packet-delivering relayer identity as an authorization primitive via `assertTrustedRelayer`. IBC packet relay is permissionless by design. Any party that observes a valid packet commit proof can submit `MsgRecvPacket` so the `relayer sdk.AccAddress` argument is not a reliable authorization signal.

When `TrustedRelayer` is configured to a non-empty value, any untrusted relayer that front-runs the trusted relayer forces the middleware to convert a successful downstream `OnRecvPacket` result into `channeltypes.NewErrorAcknowledgement(err)`. Because the IBC receive handler commits the packet receipt (sequence) regardless of ack outcome, the trusted relayer cannot resubmit the same packet afterwards. The transfer is permanently pushed onto the refund path.

The ordering problem is amplified because the trusted-relayer check runs before `transferTypes.UnmarshalPacketData(...)` and before `canBeConverted(ctx, denom)`. As a result, this DoS affects not only conversion-eligible packets but every inbound ICS-20 transfer delivered through this middleware, including ordinary token transfers unrelated to the Autheo voucher-conversion flow.

Scenario

- `TrustedRelayer` is set to a non-empty address in `im.autheokeeper.GetParams(ctx).TrustedRelayer`.
- A valid inbound ICS-20 packet exists for this chain.
- Before the intended relayer submits it, another party observes the proof and submits `MsgRecvPacket`.
- In `IBCConversionModule.OnRecvPacket(...)`, `im.app.OnRecvPacket(...)` succeeds, but `im.assertTrustedRelayer(ctx, relayer)` fails because `relayer.String()` does not match `TrustedRelayer`.
- The middleware returns `channeltypes.NewErrorAcknowledgement(err)` using `types.ErrUnauthorizedRelayer`.
- Because the packet is now processed, the intended relayer cannot later deliver the same packet successfully.
- The same behavior occurs even if the packet would never be convertible.
- This is because the relayer check happens before `transferTypes.UnmarshalPacketData(packet.GetData(), channelVersion, "")`, before `im.getIbcDenomFromPacketAndData(...)`, and before `im.canBeConverted(ctx, denom)`.
- As a result, ordinary inbound transfers handled by this middleware can also be blocked.

Recommendation

Do not use the packet-delivering relayer as an authorization control for inbound IBC packet handling. Since `MsgRecvPacket` is permissionless, `relayer` cannot be treated as a trusted identity, and any policy based on it is bypassable and can be abused to deny service.

Remove the trusted-relayer check from `OnRecvPacket`, or redesign the mechanism so it relies only on verifiable packet and channel properties rather than the submitting relayer. If special handling is required for voucher conversion, scope that logic strictly to the conversion path and perform it only after the middleware has confirmed that the packet is actually eligible for conversion. The default ICS-20 receive flow should remain independent of who submitted the packet, so unrelated inbound transfers cannot be rejected or refunded due to relayer identity.

Alleviation

[Autheo , 05/03/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/c63a090f3cf28578032b29ac07c48def5269dcc5

[CertiK , 05/03/2026]: The team heeded the advice and resolved the issue by removing `assertTrustedRelayer` from `IBCConversionModule.OnRecvPacket` (`x/autheo/middleware/conversion_middleware.go`), so inbound ICS-20 receive is no longer gated on the permissionless `MsgRecvPacket` submitter; the middleware still runs voucher conversion only after `im.app.OnRecvPacket` succeeds and only when `canBeConverted` is true. `TrustedRelayer` remains enforced on `MsgConvertVouchers` (`x/autheo/keeper/msg_server.go`), where `msg.Address` is the actual tx signer, in commit c63a090f3cf28578032b29ac07c48def5269dcc5

ACA-05 | Blocklist Refresh Fails Open On Broken Validator E2ee Key

Category	Severity	Location	Status
Coding Issue	Major	app/app.go: 386~403, 1253~1258; app/proposal.go: 95~143, 217~232	Resolved

Description

Four bugs compose into a persistent fail-open of the on-chain blocklist on any validator that cannot decrypt the latest blob. The blocklist becomes non-binding: affected validators silently ACCEPT proposals containing blocklisted txs while healthy validators REJECT — yielding either liveness halt (mixed key health, 1/3 split) or silent network-wide bypass (homogeneous breakage), with no on-chain alarm.

1. **Pre-decrypt cache poison** (`proposal.go:104-105`): `h.lastBlockList` is committed *before* `age.Decrypt` runs at line 112. A failed decrypt poisons the cache; the next call hits `bytes.Equal` and short-circuits without retry. The failure is permanently memoized until governance publishes a *different* blob.
2. **noneIdentity defeats nil guard** (`app.go:386-403`): on key-load failure the code substitutes `noneIdentity{}` (non-nil sentinel that always returns `ErrIncorrectIdentity`). The `h.Identity == nil` guard at `proposal.go:97` does not match, so decryption is attempted and always fails. Combined with (1), `h.blocklist` stays empty for the process lifetime.
3. **EndBlocker shadows the error** (`app.go:1255-1257`): `if err := app.RefreshBlockList(ctx); err != nil` introduces an inner `err` that shadows the outer `EndBlock` return; the refresh error is logged and dropped. No metric, no chain signal. Boot has a sibling `panic` gate (`app.go:1141-1148`); runtime has no equivalent.
4. **Empty-map ACCEPT fast path** (`proposal.go:219-222`): `ProcessProposal` returns ACCEPT when `len(h.blocklist) == 0`. Intended for non-validator nodes; combined with (1)+(2)+(3), it silently fails open on validators.

Realistic triggers

The blob is age-encrypted to each bonded validator's registered pubkey. Decryption can fail on a subset whenever: a validator rotates their identity without re-registering on-chain; a new validator joins after the blob was published; the admin's `encrypt-to-validators` run silently skipped a validator with no key; or an operator restarts with a missing keyring (`noneIdentity` substitution).

Proof of Concept

Bug 1 — `proposal.go:101-115`:

```
if bytes.Equal(h.lastBlockList, blob) { return nil }
h.lastBlockList = make([]byte, len(blob)); copy(h.lastBlockList, blob)  // committed
before validation
reader, err := age.Decrypt(bytes.NewBuffer(blob), h.Identity)
if err != nil { return err }                                            // cache
already poisoned
```

Bug 2 — `app.go:394-401` : `identity = noneIdentity{}` on failure; `noneIdentity.Unwrap` always returns `ErrIncorrectIdentity` .

Bug 3 — `app.go:1253-1258` :

```
rsp, err := app.ModuleManager.EndBlock(ctx)
if err := app.RefreshBlockList(ctx); err != nil {  // shadows outer err
    app.Logger().Error("failed to update blocklist", "error", err)
}
return rsp, err
```

Bug 4 — `proposal.go:219-222` : empty-map → ACCEPT.

Reproduction (mixed key health)

1. 3-validator devnet; V3 rotates age identity but skips `register-encryption-key` .
2. Admin publishes a new `MsgStoreBlockList` blob encrypted only to V1+V2.
3. V3's `SetBlockList` writes `h.lastBlockList = blob` (Bug 1) → `age.Decrypt` fails → returns error → swallowed at EndBlocker (Bug 3). V3's `h.blocklist` keeps the prior list; subsequent calls cache-hit and never retry.
4. Admin updates the blocklist to add address X. V1/V2 see X; V3 doesn't.
5. When V1/V2 propose a block including X's tx, V3 ACCEPTs (stale list) while V1/V2 REJECT (peers' blob also stale on V3) — round fails on 1/3 split. Liveness degrades until governance publishes a *different* blob to break Bug 1's memoization or V3 is fixed.

Recommendation

Apply all three:

1. **Cache-after-success** (`proposal.go`): commit `h.lastBlockList` only after `h.blocklist = m` succeeds.

```
h.blocklist = m
h.lastBlockList = append(h.lastBlockList[:0], blob...)
return nil
```

2. **Fail closed on validators** (`proposal.go` + `app.go`): add a `refreshErrored` sentinel set on refresh failure; ProcessProposal REJECTs when set. Refuse the ACCEPT fast path until at least one successful refresh has occurred

since boot — match the existing boot-time panic gate.

3. **Stop shadowing** (`app.go:1255-1257`):

```
if refreshErr := app.RefreshBlockList(ctx); refreshErr != nil {
    app.Logger().Error("failed to update blocklist", "error", refreshErr)
    app.blockProposalHandler.SetRefreshErrored(true)
} else {
    app.blockProposalHandler.SetRefreshErrored(false)
}
```

Also add a Prometheus counter `autheo_blocklist_refresh_failures_total` and have `encrypt-to-validators` warn when a bonded validator has no registered key.

Alleviation

[Autheo, 05/03/2026]: Hi, this issue has been acknowledged and fixed in commit

c63a090f3cf28578032b29ac07c48def5269dcc5

[CertiK, 05/05/2026]: The team heeded the advice and resolved the issue by addressing all four sub-bugs together: cache-after-success in `proposal.go` (only update `lastBlockList` after `age.Decrypt` succeeds), a boot-path panic gate that fails closed if `noneIdentity{}` would be used, replacing the inner `err :=` shadowing at `app.go:1255` with a distinct `refreshErr` plus a `refreshErrored` sentinel set in the `EndBlock` refresh path, and gating `ProcessProposal` to REJECT when that sentinel is set so the empty-map ACCEPT fast-path can no longer mask refresh failure, in commit c63a090.

ACA-06 Relayer Precompile RequiredGas Uses Panics On Malformed Or Unknown Calldata

Category	Severity	Location	Status
Coding Issue	Medium	x/autheo/keeper/precompiles/relayer.go: 132~177	Resolved

Description

`RelayerContract.RequiredGas` runs in the EVM gas path for this precompile. It reads `input[:4]` without ensuring four bytes exist, which can panic on short calldata. Unknown selectors call `panic(err)` after `MethodById`, while `Run` returns an error instead. The `recvPacket` branch also panics on ABI unpack errors, bad protobuf, or packet data that is not valid ICS-20 JSON. Those inputs become runtime panics during gas estimation. Ethermint may recover them as a failed transaction or not; if not recovered, validators risk denial of service, otherwise treat as hardening.

Recommendation

Mirror `Run` safety. if `len(input) < 4`, return a conservative fixed gas without slicing. For unknown selectors, return a default gas instead of `panic`. In the `recvPacket` branch, avoid `panic` on decode errors. Use a high bounded default gas or detect ICS-20 vs non-transfer packets without panicking. Align with upstream precompile patterns so invalid input fails the tx, not the process.

Alleviation

[Autheo , 05/03/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/68c3c593b9911006e1bb0aeb87b16d5517de5e4e

[CertiK , 05/03/2026]: The team heeded the advice and resolved the issue by hardening `RelayerContract.RequiredGas` in `x/autheo/keeper/precompiles/relayer.go`: return `baseCost` when `len(input) < 4`, return `baseCost` instead of panicking on unknown selectors, and make the `recvPacket` branch decode defensively (fall back to the table's conservative gas on any unpack/protobuf/JSON failure), in commit 68c3c593b9911006e1bb0aeb87b16d5517de5e4e.

ACA-07 Proposal-Level EIP-7702 Blocklist Check Uses A Different Address Encoder

Category	Severity	Location	Status
Coding Issue, Volatile Code	Medium	app/proposal.go: 130, 195	Resolved

Description

`ProposalHandler` populates its blocklist map using the configured `addressCodec.BytesToString()` and uses that same encoder for the signer and target-address checks. The single exception is the EIP-7702 authority lookup at `app/proposal.go:197`, which uses `sdk.AccAddress(addr.Bytes()).String()` instead.

Today the two encoders resolve to the same bech32 string under the default Cosmos SDK configuration, so the blocklist check still succeeds. However, the inconsistency is a latent security boundary: any future change that causes the two encoders to diverge — a bech32 prefix migration, an address-codec swap, or multi-prefix support — would silently allow blocked EIP-7702 authorities to slip through proposal validation while the same blocked address remains blocked everywhere else in the handler.

Proof of Concept

Encoder mismatch in the same handler (`app/proposal.go`):

```
// ~line 134 — blocklist map populated using addressCodec
encoded, err := h.addressCodec.BytesToString(addr)
m[encoded] = struct{}{}

// ~line 197 — EIP-7702 authority lookup uses sdk.AccAddress
addr, err := auth.Authority()
if err == nil {
    if _, ok := h.blocklist[sdk.AccAddress(addr.Bytes()).String()]; ok {
        //                  ↑ different encoder than the map keys
        return fmt.Errorf("signer is blocked: %s", addr.String())
    }
}
```

Locating the inconsistency by line number:

```
addressCodec.BytesToString : lines 134, 169, 184, 202
sdk.AccAddress().String()  : line  197
```

Recommendation

Use `h.addressCodec.BytesToString()` for the EIP-7702 authority blocklist lookup so all blocklist comparisons in the proposal handler share a single encoder.

Alleviation

[Autheo , 05/03/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/99c5d578e6c6a9c1c2986ad8287dcadfeecdb5eb

---

[CertiK , 05/03/2026]: The team heeded the advice and resolved the issue by changing the EIP-7702 set-code authority blocklist lookup in `ProposalHandler.ValidateTransaction` (`app/proposal.go`) to use `h.addressCodec.BytesToString(addr.Bytes())` (same encoder as the rest of the handler) instead of `sdk.AccAddress(addr.Bytes()).String()` , in commit 99c5d578e6c6a9c1c2986ad8287dcadfeecdb5eb

ACA-08 | Eth_createAccessList RPC Method Is Not Implemented

Category	Severity	Location	Status
Coding Issue	Medium	app/app.go: 54; x/autheo/rpc/api.go: 38~42, 45~55, 58~87	Resolved

Description

The JSON-RPC server does not expose `eth_createAccessList`, the standard endpoint for pre-computing the addresses and storage slots an EIP-2930 transaction will access. Standard Ethereum tooling (Hardhat, Foundry, ethers.js, viem) expects this endpoint to be present and uses it to construct gas-optimised access-list transactions.

Without it, wallets and dApps cannot apply the gas savings that EIP-2930 access lists are designed to provide, and tools that auto-detect access-list support will either error or fall back to less efficient flows.

Proof of Concept

Direct RPC probe:

```
$ curl -s -X POST http://localhost:8545 \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","method":"eth_createAccessList",
         "params":[{"to":"0x0","data":"0x"},"latest"],"id":1}'
{"jsonrpc":"2.0","id":1,
 "error":{"code":-32601,
          "message":"the method eth_createAccessList does not exist/is not
available"}}
```

Source search:

```
$ grep -r 'createAccessList\|CreateAccessList' \
    autheo-chain-core/autheo-chain-core --include='*.go'
(no results)
```

Recommendation

Implement `eth_createAccessList` in the JSON-RPC server. The standard implementation runs the call inside an access-list tracer and returns both the resulting access list and the adjusted gas estimate.

Alleviation

[Autheo, 05/07/2026]: https://github.com/autheo-blockchain/autheo-chain-core/commit/e03a8eb55b4107d306eeb0418e5a6a32c5e19bf5

[CertiK, 05/07/2026]: The team heeded the advice and resolved the issue by upgrading the Dockerfile build-stage base image from `golang:1.23.12-alpine` to `golang:1.25-alpine` and refreshing `go.mod` / `go.sum` through the toolchain bump, picking up the corresponding fixes in the standard-library extension modules and indirect dependencies that were pinned to versions compatible with the older Go release, in commit e03a8eb.

ACA-09 No Support For Historical Block Hash Retrieval Beyond Native BLOCKHASH (EIP-2935)

Category	Severity	Location	Status
Coding Issue	Medium	app/app.go: 688~695; x/autheo/keeper/precompiles/interface.go: 1~20	Resolved

Description

The chain does not implement EIP-2935 (historical block-hash storage via a system contract) or any equivalent mechanism. Smart contracts are therefore limited to the native `BLOCKHASH` opcode behavior, which only exposes the most recent 256 block hashes.

Contracts that expect to read older block hashes (for cross-chain verification, randomness derivations, or any state-anchoring pattern) will receive zero values rather than the expected hashes and may behave unsafely. As EIP-2935 becomes increasingly standard on Ethereum and EVM-compatible chains, its absence creates a compatibility gap for tooling and contract code that assumes it is present.

Recommendation

Either implement EIP-2935 (deploy the history-storage system contract and override `BLOCKHASH` to fall back to it for blocks older than 256) or document that the chain does not support historical block-hash retrieval beyond the native window.

Alleviation

[Autheo, 05/07/2026]: https://github.com/autheo-blockchain/autheo-chain-core/commit/9030890a474b2ff2f148f1180491566bcc3543da

---

[CertiK, 05/07/2026]: The team heeded the advice and resolved the issue by adding a `common.IsHexAddress(msg.Contract)` check to `RegisterOrUpdateTokenMapping` in `x/autheo/keeper/keeper.go` before the `common.HexToAddress` call, so a malformed hex address now fails with `ErrInvalidAddress` instead of being silently zero-padded and accepted as a degenerate `0x0…` mapping, in commit 9030890.

ACA-10 | BeginBlocker Iterates All Historical License IDs On Every Block

Category	Severity	Location	Status
Logical Issue	Medium	x/emissions/abci.go: 21~23; x/emissions/keeper/nft_rewards.go: 130~133	Resolved

Description

`getActiveLicenseRewards` runs every block and loops `id` from 1 to `NextLicenseId-1`, with multiple KV reads per ID (license, binding, validator, delegation, dust check). Licenses are not removed from store on revoke; `NextLicenseId` only increases. Work grows with **all IDs ever minted**, not only active licenses. No secondary index of active IDs and no license ca BeginBlocker cost is unbounded; at scale this can stretch block time and hurt liveness.

Recommendation

Prefer iterating only ACTIVE license IDs (maintain a KV index updated on status changes), or lazy accrual at claim time.

If you add a true delete/reap path remove the license record and its index keys from the KV store so IDs no longer cost a read every block. Currently revoke keeps the row, so scan cost still tracks all minted IDs. Compacting `NextLicenseId` is hard without ID reuse—usually you still need an active-ID set or lazy accounting.

Alleviation

[Autheo , 05/03/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/99c5d578e6c6a9c1c2986ad8287dcadfeecdb5eb

[CertiK , 05/03/2026]: The team heeded the advice and resolved the issue by adding an `ActiveLicenseIndexPrefix` secondary index maintained by `SetLicenseStatus` (and re-populated by `InitGenesis`), exposing an `IterateActiveLicenseIDs` helper, and switching `getActiveLicenseRewards` to use it so `BeginBlocker` cost becomes O(active) instead of O(`NextLicenseId`), in commit 99c5d578e6c6a9c1c2986ad8287dcadfeecdb5eb.

ACA-11 | StakingMintedTotal Counter Missing From Genesis Import/Export

Category	Severity	Location	Status
Coding Issue	Medium	app/staking_mint_inflation.go: 67; x/emissions/keeper/state.go: 248; x/emissions/types/genesis.go: 1; x/emissions/types/keys.go: 41	Resolved

Description

StakingMintedTotal is not included in Genesis Import/Export. Any chain upgrade / recovery through genesis-based forks will cause it to reset, making the chain able to mint above the target maximum minted token allocations.

Recommendation

Include StakingMintedTotal to genesis import/export

Alleviation

[Autheo , 05/03/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/d2d743b347277eaa84770da13a60b714cf787e72

[CertiK , 05/03/2026]: The team heeded the advice and resolved the issue by adding `staking_minted_total`, `staking_cycle_minted`, and `staking_last_cycle_index` to `GenesisState` and wiring them through `InitGenesis`, `ExportGenesis`, `DefaultGenesis`, and `GenesisState.Validate`, so the staking hard cap, monthly cap, and cycle boundary are all preserved across restarts and upgrades, in commit d2d743b347277eaa84770da13a60b714cf787e72.

ACA-12 | Per-Owner Sovereign-License Unenforced

Category	Severity	Location	Status
Volatile Code, Coding Issue	Medium	x/license/keeper/keeper.go: 129; x/license/keeper/msg_server.go: 276; x/license/types/errors.go: 58	Resolved

Description

The invariant *"an owner may hold at most one sovereign license (1:1 mapping with validator)"* is declared in two places:

- Proto doc: `proto/license/v1/types.proto:14-15`.
- Error: `ErrSovereignLimitExceeded` at `x/license/types/errors.go:58`.

Neither `MintLicense` nor `BindLicense` enforces it, and `ErrSovereignLimitExceeded` has zero call sites. An owner who receives 2+ sovereign NFTs (via permissive mint contract, OTC accumulation, or the MEDIUM-31 owner-desync) holds 2+ sovereigns in x/license and can bind each to a different validator — operating two validators from one wallet and collecting 2× sovereign NFT rewards (default ~188k THEO/year each).

Proof of Concept

Invariant declared in proto (`proto/license/v1/types.proto:14-15`):

```
// An owner may hold at most one sovereign license (1:1 mapping with validator).
LICENSE_CATEGORY_SOVEREIGN = 1;
```

Error declared, never raised (`x/license/types/errors.go:58`):

```
x/license/types/errors.go:58:  ErrSovereignLimitExceeded = ...   # declaration only
```

`MintLicense` (`x/license/keeper/keeper.go:129`) and `BindLicense` (`x/license/keeper/msg_server.go:276`) do not enumerate existing sovereigns for the owner.

Live reproduction (`test_medium30_sovereign_limit_unenforced_live`):

```
mint tokenId=100 → alice, gas=150000
mint tokenId=101 → alice, gas=150000
alice's SOVEREIGN licenses from this contract: 2
  license id=1 tokenId=100 status=LICENSE_STATUS_ISSUED
  license id=2 tokenId=101 status=LICENSE_STATUS_ISSUED
chain logs mention ErrSovereignLimitExceeded: False
```

Recommendation

Enforce the invariant in `MintLicense` (primary) and `BindLicense` (defense-in-depth):

```
if category == types.LicenseCategory_LICENSE_CATEGORY_SOVEREIGN {
    for _, lic := range k.GetLicensesByOwner(ctx, owner) {
        if lic.Category == types.LicenseCategory_LICENSE_CATEGORY_SOVEREIGN &&
            lic.Status != types.LICENSE_STATUS_REVOKED {
            return 0, types.ErrSovereignLimitExceeded
        }
    }
}
```

Add the same check to `GenesisState.Validate()`. Existing multi-sovereign owners (if any) need an upgrade handler.

Alleviation

[Autheo , 05/03/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/c32a6a41d6b0e126046c84b5fbb591aa45d6b61e

[CertiK , 05/03/2026]: The team heeded the advice and resolved the issue by enforcing the 1-per-owner sovereign invariant at all three layers identified in the recommendation, in commit c32a6a41d6b0e126046c84b5fbb591aa45d6b61e.

ACA-13 ExportGenesis Uses Map Iteration — Produces Non-Deterministic Output Across Validators

Category	Severity	Location	Status
Coding Issue, Volatile Code	Medium	x/emissions/keeper/nft_rewards.go: 293; x/emissions/keeper/state.go: 248	Resolved

Description

`ExportGenesis` in `x/emissions` builds the `LicenseAccruedRewards` slice by ranging over the map returned by `GetAllLicenseAccruedRewards`. Go's map iteration order is randomized, so two validators exporting the same state at the same height produce byte-different genesis files.

This matters for chain exports used in migrations, state-sync manifest generation, and cross-validator export verification (operators often diff exports to confirm identical state). A non-deterministic export breaks those workflows and is a latent correctness risk whenever export output is compared byte-for-byte.

Proof of Concept

Export code (`x/emissions/keeper/state.go:248-267`):

```
allAccrued := k.GetAllLicenseAccruedRewards(ctx)
licenseAccruedRewards := make([]types.LicenseAccruedReward, 0, len(allAccrued))
for licenseID, amount := range allAccrued {  // ← map iteration
    licenseAccruedRewards = append(licenseAccruedRewards,
types.LicenseAccruedReward{
        LicenseId:    licenseID,
        AccruedUtheo: amount,
    })
}
```

Underlying producer (`x/emissions/keeper/nft_rewards.go:293-305`) returns `map[uint64]sdkmath.Int` rather than using a prefix iterator.

Recommendation

Either sort the `licenseAccruedRewards` slice by `LicenseId` before returning, or replace the map producer with a prefix iterator (`storetypes.KVStorePrefixIterator(store, LicenseAccruedRewardsPrefix)`) that emits entries in sorted key order natively.

Alleviation

[Autheo , 05/03/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/79aaf1678d7765d8efd4fc37cfacc6a39316ba36

[CertiK , 05/03/2026]: The team heeded the advice and resolved the issue by replacing the map-based producer with a prefix iterator at both call sites in commit 79aaf1678d7765d8efd4fc37cfacc6a39316ba36.

ACA-14 Staking Reward Inflation: Integer Division BlocksPerYear / 12 Can Create Extra Monthly Cycles

Category	Severity	Location	Status
Logical Issue	Medium	app/staking_mint_inflation.go: 24~30	Resolved

Description

`NewStakingRewardsInflationFn` derives `blocksPerMonth := params.BlocksPerYear / 12` using integer division, then advances a monthly emission cycle with `cycleIndex := uint64(blockHeight) / blocksPerMonth`. When `BlocksPerYear` is not divisible by 12, `12 * blocksPerMonth` is strictly less than `BlocksPerYear`, so more than 12 cycle boundaries can occur within one nominal year of block heights. That can allow the monthly cap (`StakingMonthlyCapUtheo`) to apply more often than intended within the same calendar block span, increasing minted staking rewards relative to the intended 12-month schedule. If `blocksPerMonth` were zero, the code sets it to 1.

Recommendation

Enforce `BlocksPerYear % 12 == 0` in mint params `ValidateBasic`, or replace the cycle math with a formulation that ties monthly windows to the full `BlocksPerYear` without truncation so exactly 12 monthly periods cover one year of blocks.

Alleviation

[Autheo , 05/03/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/e594bfce01ded276e86752912e1b0797bb328604

[CertiK , 05/03/2026]: The team heeded the advice and resolved the issue by replacing the truncated `cycleIndex := uint64(blockHeight) / blocksPerMonth` formula in `NewStakingRewardsInflationFn` (`app/staking_mint_inflation.go`) with `cycleIndex = (uint64(blockHeight) * 12) / params.BlocksPerYear`, which ties monthly windows to the full `BlocksPerYear` without truncation so exactly 12 cycle boundaries fall within one year of blocks regardless of divisibility, and adding a `params.BlocksPerYear > 0` guard to avoid a divide-by-zero on misconfigured params, in commit e594bfce01ded276e86752912e1b0797bb328604.

ACA-15 Relayer Precompile Run Returns Raw Protobuf Bytes Where The ABI Declares Encoded Bytes

Category	Severity	Location	Status
Coding Issue	Medium	x/autheo/keeper/precompiles/relayer.go: 179~248	Resolved

Description

`RelayerContract.Run` dispatches every IBC method through `exec(e, bc.ibcKeeper.<Method>)` and then `return res, err`, where `res` is the raw response bytes produced by the IBC keeper. The compiled relayer ABI declares methods such as `createClient`, `connectionOpenInit`, `channelOpenInit`, `recvPacket`, and others as returning dynamic `bytes`. A dynamic `bytes` return requires the precompile to emit ABI output encoding (offset word plus length word plus padded payload). The current code skips `method.Outputs.Pack(...)` or any equivalent encoding step, so high-level Solidity callers that invoke the precompile through a normal typed interface receive raw Protobuf response bytes and revert during ABI decoding. Low-level `call` consumers that read returndata manually can still recover the payload.

Recommendation

After dispatching the IBC method, pack the response payload using the method's declared outputs, for example `out, err := method.Outputs.Pack(res)` before returning, so every method whose ABI advertises a dynamic `bytes` return produces standard ABI-encoded output. Review all relayer methods to confirm runtime encoding matches the published ABI.

Alleviation

[Autheo , 05/03/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/dac271c082b9f1f28fbadbb32c31ba889044a454

[CertiK , 05/03/2026]: The team heeded the advice and resolved the issue by adding a `method.Outputs.Pack(res)` step at the end of `RelayerContract.Run` in `x/autheo/keeper/precompiles/relayer.go`, so the raw protobuf bytes produced by every dispatched IBC method (`createClient`, `updateClient`, `connectionOpen*`, `channelOpen*`, `channelClose*`, `recvPacket`, `acknowledgement`, `timeout`, `timeoutOnClose`) are wrapped in standard ABI dynamic-`bytes` encoding (offset + length + padded payload) before being returned to the EVM caller, in commit dac271c082b9f1f28fbadbb32c31ba889044a454.

ACA-16 Blocklist Enforcement Does Not Unwrap Authz MsgExec, Allowing Blocklisted Granters To Act Through Grantees

Category	Severity	Location	Status
Coding Issue	Medium	app/app.go: 1208~1213; app/block_address.go: 32~43; app/proposal.go: 145~176	Resolved

Description

The chain implements a purpose-built, age-encrypted compliance blocklist system that enforces sanctions at the consensus layer via `ProcessProposalHandler`. The enforcement logic checks multiple bypass vectors — top-level signers, EVM `To()` destinations, and EIP-7702 `Authority()` / `Address` fields — demonstrating awareness of evasion paths. However, both enforcement points (`BlockAddressesDecorator.AnteHandle` and `ProposalHandler.ValidateTransaction`) only inspect top-level transaction signers and messages, and do not unwrap `cosmos.authz.v1beta1.MsgExec` to evaluate the granter whose authority is being exercised.

This creates a compliance gap: if account A is blocklisted and has pre-existing authz grants to account B (created before A was blocklisted), B can submit `MsgExec` containing inner messages that exercise A's authority. Both enforcement layers see only B (the grantee/signer) and pass the check — A (the blocked granter) is never evaluated. The bypass passes all three enforcement gates in the transaction lifecycle:

1. **CheckTx** — `BlockAddressesDecorator` sees B as signer, MsgExec is not MsgEthereumTx so no EVM-specific checks fire
2. **PrepareProposal** — `ValidateTransaction` same result, tx is included in proposed block
3. **ProcessProposal** — `ValidateTransaction` same result, all validators accept
4. **DeliverTx** — `BlockAddressesDecorator` skips entirely (`ctx.IsCheckTx()` is false), no `ValidateTransaction` call — tx executes with zero blocklist enforcement

This is significant because the blocklist is specifically designed for regulatory/sanctions compliance — a domain where the chain invested in a unique, consensus-embedded encrypted enforcement system. Unlike most Cosmos chains which have no compliance blocklist at all and handle sanctions off-chain, this chain chose to enforce at the protocol level, making gaps in that enforcement directly relevant to its compliance posture.

A blocklisted entity can still perform the following via authz grants to a cooperative grantee:

- **Move funds** via `MsgSend`
- **Stake/unstake** via `MsgDelegate` / `MsgUndelegate`
- **Vote on governance** via `MsgVote`
- **Claim rewards** via `MsgWithdrawDelegatorReward`

`DisabledAuthzMsgs` (app.go:1208-1213) blocks `MsgEthereumTx` and vesting messages from authz wrapping, showing the team considered authz risks for specific message types — but did not extend the same protection to the blocklist

enforcement path.

Proof of Concept

1. Account A grants authz GenericAuthorization to account B for /cosmos.bank.v1beta1.MsgSend
2. Account A is later added to the encrypted blocklist via MsgStoreBlockList
3. Account B submits MsgExec containing MsgSend transferring A's funds
4. CheckTx: BlockAddressesDecorator checks signers — sees only B (not blocked) → passes into mempool
5. PrepareProposal: ValidateTransaction checks signers — sees only B → included in block
6. ProcessProposal: ValidateTransaction checks signers — sees only B → all validators accept
7. DeliverTx: BlockAddressesDecorator skips (IsCheckTx=false), no ValidateTransaction call → executes
8. A's funds are moved via the authz grant despite A being blocklisted

The same pattern works for MsgDelegate (staking A's tokens to earn rewards), MsgVote (A participating in governance), and MsgWithdrawDelegatorReward (A extracting accumulated staking rewards) — all authz-eligible and not covered by DisabledAuthzMsgs.

Recommendation

In both `BlockAddressesDecorator.AnteHandle` and `ProposalHandler.ValidateTransaction`, add a type assertion for `cosmos.authz.v1beta1.MsgExec` when iterating `tx.GetMsgs()` — mirroring the existing type assertion for `*evmtypes.MsgEthereumTx`. For any `MsgExec` found, decode the inner messages and check their signer fields (which represent the granter) against the blocklist. Reject the transaction if the granter is blocked.

Note: `MsgExec` itself only carries the grantee address. The granter is implicit — it is the signer of the inner messages. The inner messages can be decoded via `msgExec.GetMessages()` and their signers inspected.

As an immediate, lower-effort mitigation: add `MsgExec` to `DisabledAuthzMsgs` to prevent any authz-wrapped execution while the blocklist is active. This is a blunt approach that disables authz for all users, but eliminates the bypass completely.

Alleviation

[Autheo , 05/03/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/b4601bc42061b71aaeed4f72ad088267825b2210

[CertiK , 05/03/2026]: The team heeded the advice and resolved the issue by extending both blocklist enforcement points (`BlockAddressesDecorator.AnteHandle` in `app/block_address.go` and `ProposalHandler.ValidateTransaction` in `app/proposal.go`) with a `*authz.MsgExec` type assertion that decodes the inner messages via `GetMessages()` and checks each inner `sdk.LegacyMsg`'s `GetSigners()` (the implicit granter) against the blocklist, rejecting the transaction if any granter is blocked and failing closed when inner messages cannot be decoded, in commit b4601bc42061b71aaeed4f72ad088267825b2210.

ACA-17 OnPacketResult Silently Treats Failed EVM IBC Callbacks As Success

Category	Severity	Location	Status
Coding Issue	Medium	x/autheo/keeper/keeper.go: 301~302	Resolved

Description

`onPacketResult` in `x/autheo/keeper/keeper.go:279-303` invokes the IBC callback contract via `CallEVM` but discards the returned `*evmtypes.EVMResult` and relies only on the Go `error` return:

```
_, _, err = k.CallEVM(ctx, &senderAddr, data, big.NewInt(0), gasLimit)
return err
```

`CallEVM` (`x/autheo/keeper/evm.go:24-53`) only returns a non-nil Go error for setup / gas-preparation failures inside `ApplyMessage`. EVM reverts (out-of-gas under `MaxCallbackGas`, Solidity revert, invalid opcode) are reported via `res.VmError` / `res.Failed()` while `err == nil`. Sibling helpers `CallModuleCRC21` (line 65) and `DeployModuleCRC21` (line 85) in the same file correctly gate on `res.Failed()` — `onPacketResult` is internally inconsistent.

IBC callbacks middleware (`ibc-go/v10/modules/apps/callbacks/internal/process.go`) wraps the callback in a `CacheContext` and commits it whenever the Go error is nil. So a reverted callback produces:

- `CacheContext.writeFn()` is invoked (cache committed).
- `EmitCallbackEvent` records `err=nil`, emitting a success event.
- The IBC ack / timeout lifecycle completes as if the callback executed cleanly.

EVM storage itself is rolled back by the EVM on `VmError`, so user funds in Solidity contracts are safe. The damage is dApp-observability: contracts and off-chain indexers subscribing to the callback event cannot distinguish a successful callback from a silently-failed one, and any non-storage side effects in the cache context (events, logs, module state touched from the callback path) commit despite the revert.

This is a forked function Cronos, but Cronos fixed the pattern in PR #1987:

```
// cronos/x/cronos/keeper/keeper.go (post-fix)
_, res, err := k.CallEVM(ctx, &senderAddr, data, big.NewInt(0), gasLimit)
if err != nil {
    return err
}
if res.Failed() {
    return fmt.Errorf("IBC callback EVM execution reverted: %s", res.VmError)
}
return nil
```

Recommendation

Port Cronos PR #1987 (`4b3cf329`):

```
_, res, err := k.CallEVM(ctx, &senderAddr, data, big.NewInt(0), gasLimit)
if err != nil {
    return err
}
if res.Failed() {
    return fmt.Errorf("IBC callback EVM execution reverted: %s", res.VmError)
}
return nil
```

Add an integration test that registers a reverting callback contract and asserts the caller receives a non-nil error (and the callback event records the failure). Sweep the remaining `CallEVM` call sites in `x/autheo/keeper/` for the same pattern and ensure every caller either checks `res.Failed()` or documents why the result is intentionally discarded.

Alleviation

[Autheo, 05/07/2026]: https://github.com/autheo-blockchain/autheo-chain-core/commit/9030890a474b2ff2f148f1180491566bcc3543da

---

[CertiK, 05/07/2026]: The team heeded the advice and resolved the issue by adding `len(input) < 4 { return baseCost }` to `IcaContract.RequiredGas` and `len(contract.Input) < 4 { return nil, "input too short" }` to `IcaContract.Run` in `x/autheo/keeper/precompiles/ica.go` , mirroring the relayer-precompile fix shipped in rev-1's ACA-06 and closing the explicit ICA gap noted in our prior remediations write-up, in commit 9030890.

ACA-18 | Token Mappings Do Not Verify Contract Code Exists

Category	Severity	Location	Status
Coding Issue	Medium	x/autheo/keeper/keeper.go: 216; x/autheo/keeper/msg_server.go: 85; x/autheo/proposal_handler.go: 15; x/autheo/types/proposal.go: 45	Resolved

Description

`RegisterOrUpdateTokenMapping()` stores a denom-to-contract mapping without verifying that bytecode exists at the target address. The message handler validates only the bech32 sender and `common.IsHexAddress(msg.Contract)`, and the governance route is weaker still: `TokenMappingChangeProposal.ValidateBasic()` returns `nil` and the proposal handler forwards the raw contract string directly into the keeper.

As a result, the chain can authoritatively associate a denom with an EOA, an undeployed address, or a self-destructed contract. dApps and wallets that resolve denoms via `contract-by-denom` then receive a non-functional address; subsequent conversion attempts revert at the EVM layer when `CallModuleCRC21` invokes a method that does not exist on the target.

Proof of Concept

Vulnerable code (`x/autheo/keeper/keeper.go:260-274`, non-source branch — has hex check but no bytecode check):

```
} else {
    if len(msg.Contract) == 0 {
        k.DeleteExternalContractForDenom(ctx, msg.Denom)
    } else {
        if !common.IsHexAddress(msg.Contract) {
            return errors.Wrapf(sdkerrors.ErrInvalidAddress, ...)
        }
        contract := common.HexToAddress(msg.Contract)
        // No ensureContractCode() check
        if err := k.SetExternalContractForDenom(ctx, msg.Denom, contract); err !=
nil {
            return err
        }
    }
}
```

Steps to reproduce:

1. Set `autheo_admin` to a controlled address via gov proposal.
2. As admin, submit `MsgUpdatePermissions` granting `CanChangeTokenMapping` (value `1`) to that address.
3. Generate a random 20-byte Ethereum address and confirm it has no code (`eth_getCode` returns `0x`).

4. Submit `MsgUpdateTokenMapping` mapping a fabricated IBC denom (`ibc/<random 64-hex>`) to that address.

Observed result:

```
Target EOA       : 0xfc9ed96a0fa5b9f41cbc06aebb56b604afafdb47
eth_getCode      : 0x                                  (no bytecode)
Fake denom       : ibc/D70AC768...82F4E5AD17DD
Tx code          : 0                                   (mapping accepted)
contract-by-denom:
  contract       : 0xFc9eD96A0Fa5B9F41CBC06AeBb56B604aFafdB47
  auto_contract  : (empty)
eth_getCode (post): 0x                                 (still no bytecode)
```

The chain now reports a denom backed by a contract that does not exist. Any subsequent conversion of that denom calls `CallModuleCRC21` on the EOA and reverts at the EVM layer.

Recommendation

Add a keeper-level `ensureContractCode()` helper that calls `evmKeeper.Code()` on the target address and rejects mappings whose target has empty bytecode. Invoke it in both branches of `RegisterOrUpdateTokenMapping()` before persisting the mapping. Implement a strict `ValidateBasic()` for the token-mapping governance proposal so the same validation runs through the gov path.

Alleviation

[Autheo , 05/03/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/168f11abf254a46a8c4412158f480a1a037a72e8

[CertiK , 05/03/2026]: The team heeded the advice and resolved the issue by adding an `ensureContractCode()` keeper helper (using `evmKeeper.GetAccount()` and `evmtypes.IsEmptyCodeHash`) that is invoked in both branches of `RegisterOrUpdateTokenMapping()` before persisting a mapping, registering a new `ErrContractNotDeployed` sentinel error, and implementing a strict `TokenMappingChangeProposal.ValidateBasic()` that enforces denom format and hex-address sanity on the governance path in commit 168f11abf254a46a8c4412158f480a1a037a72e8.

ACA-19 | Critical Forked Dependencies Lack Concrete Version Tracking

Category	Severity	Location	Status
Coding Issue	Medium	go.mod: 283, 287, 300, 307, 309	Resolved

Description

Critical components (Cosmos SDK, CometBFT, go-ethereum, Ethermint) are pinned via custom `replace` pseudo-versions. The repository does not provide a local manifest mapping those forks to upstream release baselines and security patch coverage.

This is a governance and risk-management weakness: during advisory response, operators cannot quickly determine whether a known fix is present. The issue is not a single code bug but opaque provenance in a high-impact dependency surface.

Recommendation

It's recommended that the development team maintain a clear and up-to-date patch manifest for all forked or replaced critical dependencies. This manifest should detail the specific upstream versions serving as the baseline, list all applied patches or commits, and indicate the status of relevant security advisories for each dependency. Ensuring this documentation is available and regularly updated will help operators and stakeholders quickly assess patch coverage and security posture during vulnerability management or incident response situations.

Alleviation

[Autheo , 05/03/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/95b9bc865fd97241c4302bf5a1f651c2c5efca3b

[CertiK , 05/04/2026]: The team heeded the advice and resolved the issue by introducing a `FORK_MANIFEST.md` that maps every forked or replaced critical dependency (Cosmos SDK, CometBFT, go-ethereum, Ethermint, and supporting replacements) to its upstream release baseline, enumerates the patches applied by the fork maintainer, links the relevant security-advisory feeds, and documents an update procedure for keeping the manifest current in commit 95b9bc865fd97241c4302bf5a1f651c2c5efca3b.

ACA-20 | NFT Reward Dust Consistently Assigned To Highest Active License ID

Category	Severity	Location	Status
Coding Issue	Minor	x/emissions/keeper/nft_rewards.go: 259	Resolved

Description

`accrueRewardsToLicenses()` distributes the per-block reward by truncating each license's proportional share and handing the remainder to the last license in the slice. Because `getActiveLicenseRewards()` iterates in ascending order of license ID, the last slice element is deterministically the highest-ID active license. That license therefore receives a small but systematic premium every block with mixed-tier holders.

Individual-block magnitude is bounded by `len(licenses) - 1` utheo (integer truncation residue across tiers), so the absolute bonus is small relative to scheduled emissions. Over long horizons it still compounds into a non-trivial wealth transfer to whoever happens to hold the highest-ID active license at any given block.

Biased distribution (`x/emissions/keeper/nft_rewards.go:259-270`):

```
for i, lr := range licenses {
    var share sdkmath.Int
    if i == len(licenses)-1 {
        // Last license receives the dust so that sum == totalMintInt exactly.
        share = totalMintInt.Sub(distributed)          // ← dust sink
    } else {
        shareDec := sdkmath.LegacyNewDecFromInt(totalMintInt).
            Mul(lr.perBlockDec).Quo(totalPerBlockDec)
        share = shareDec.TruncateInt()
    }
    // ...
}
```

`getActiveLicenseRewards` iterates `for id := 1; id < nextID; id++` , so `licenses[len-1]` is always the highest-ID active license.

Recommendation

Rotate the dust recipient deterministically by block height (e.g. `dustIndex = blockHeight % len(licenses)`). This preserves determinism, keeps the total mint exact, and removes the systematic bias toward high-ID license holders. Alternatively, we invite the dev team to clarify the rational of this choice.

Alleviation

[Autheo, 05/05/2026]: https://github.com/autheo-blockchain/autheo-chain-core/commit/aa3cfb10e4c8246e79e6ea254dd2420c8c634df2

[CertiK, 05/06/2026]: The team heeded the advice and resolved the issue by replacing the fixed "last license receives the dust" branch in `accrueRewardsToLicenses` with a rotating recipient indexed by `blockHeight % len(licenses)` — proportional shares are paid to all other licenses in a first pass, and the rotating recipient receives `totalMintInt - distributed` (its proportional share plus the truncation dust) in a second pass — preserving sum-equals-total while removing the systematic premium to the highest-ID active license, in commit aa3cfb1.

ACA-21 | Community Remainder Is Not Recomputed When Block Rewards Reaches Maximum Emission

Category	Severity	Location	Status
Coding Issue	Minor	x/emissions/keeper/block_rewards.go: 37, 83	Resolved

Description

When `MintBlockRewards` enters the cap-clamp branch, `newCommunityRemainder` is computed from the pre-clamp `totalDec` while `communityAmount` is replaced with `remaining`. The pre-clamp remainder is then persisted at line 83, even though the amount actually minted was reduced.

Under current operation this is cosmetic because the cap-reached short-circuit prevents further minting on the next block. However, if a future upgrade raises `EmissionsCap`, the stranded remainder immediately produces an extra integer mint on the next block.

Proof of Concept

Clamp branch (`x/emissions/keeper/block_rewards.go:37-83`):

```
communityAmount, newCommunityRemainder := k.calculateMintAmountWithRemainder(
    params.CommunityPerBlockUtheo,  // could be astronomically large post-HIGH-11
    k.GetCommunityRemainder(ctx),
)
totalToMint := communityAmount

if emittedTotal.Add(totalToMint).GT(params.EmissionsCap) {
    remaining := params.EmissionsCap.Sub(emittedTotal)
    if remaining.IsPositive() {
        communityAmount = remaining   // amount clamped
        totalToMint = remaining
        // newCommunityRemainder is NOT recomputed — keeps pre-clamp value
    }
}
// ... further down ...
k.SetCommunityRemainder(ctx, newCommunityRemainder)   // persists the pre-clamp
remainder
```

A cap raise that re-enables emissions would then use this inflated remainder on the very next block, bypassing the clamp that originally stopped the drain.

Recommendation

Recompute `newCommunityRemainder` when the cap is reached — or reset it to zero so that no fractional accrual survives a cap-exhaustion event. Apply the same treatment to the NFT remainder path in `MintNFTBlockRewards`.

Alleviation

[Autheo, 05/05/2026]: https://github.com/autheo-blockchain/autheo-chain-core/commit/750c95efd9497374adfc830eaf4ec49461f7bb6f

---

[CertiK, 05/06/2026]: The team heeded the advice and resolved the issue by zeroing `newCommunityRemainder` in both clamp branches of `MintBlockRewards` (the per-block `maxPerBlock` clamp and the cap-exhaustion clamp), so the pre-clamp fractional carry no longer survives a clamped block to produce an extra mint on the next block or after governance raises the cap, in commit 750c95e.

ACA-22 | MsgStoreBlockList Blob Has No Size Limit

Category	Severity	Location	Status
Denial of Service, Access Control	Minor	app/proposal.go: 92; x/autheo/keeper/msg_server.go: 134; x/autheo/types/messages.go: 201	Resolved

Description

`MsgStoreBlockList.ValidateBasic()` runs only a bech32 check on the sender and a dummy-identity format probe on the blob. It imposes no upper bound on blob size. The raw blob is then persisted to consensus state under `types.KeyPrefixBlockList` and re-read by every validator on each `ProposalHandler.SetBlockList()` refresh, where it is fully decrypted and JSON-unmarshalled in memory.

Under the standard audit threat model — privileged keys can be compromised — a single `MsgStoreBlockList` transaction forces every validator to persist the blob, allocate `O(blob_size)` transient memory for the decrypted payload, and consume CPU re-processing it. A compromised admin can repeat this in every block (each new blob bypasses the `bytes.Equal(h.lastBlockList, blob)` early-return cache), sustaining chain-wide resource pressure for as long as the admin key is not rotated via governance.

The attack does not cause a chain halt or direct fund loss — validators remain live, and the persistent state is bounded by the latest blob plus IAVL retention. Block-size (`max_bytes`) caps any single blob to ~1 MiB on the devnet configuration. The severity is therefore Medium rather than High: a compromised admin can degrade every validator simultaneously with a single tx, but the impact is reversible (subsequent small blob overwrites) and observable (decrypt-failure logs, sudden state-size growth).

Proof of Concept

Missing size check (`x/autheo/types/messages.go:201-212`):

```
func (msg *MsgStoreBlockList) ValidateBasic() error {
    _, err := sdk.AccAddressFromBech32(msg.From)
    if err != nil {
        return errors.Wrapf(sdkerrors.ErrInvalidAddress, ...)
    }
    // skip heavy operation in Decrypt by early return with errDummyIdentity
    _, err = age.Decrypt(bytes.NewBuffer(msg.Blob), new(dummyIdentity))
    if err != nil && !stderrors.Is(err, errDummyIdentity) {
        return err
    }
    return nil   // ← no len(msg.Blob) check
}
```

`StoreBlockList` (`msg_server.go:140`) then writes the raw blob to store:

`ctx.KVStore(k.storeKey).Set(types.KeyPrefixBlockList, msg.Blob)` .

Live PoC (`PoC/tests/poc_f23_blocklist_blob_size.py`) on a running testnet:

1. Bootstrap `autheo_admin = validator1` via gov proposal.
2. Generate age-encrypted blobs of 1 KiB, 64 KiB, and 512 KiB payloads with `pyrage` (valid age format that passes the dummy-identity probe).
3. Submit each via `autheod tx autheo store-block-list` .

Observed result — all three sizes accepted with `tx code: 0` :

```
[submit size=1KiB]   tx code: 0
[submit size=64KiB]  tx code: 0
[submit size=512KiB] tx code: 0
```

Subsequent submissions would continue scaling up to the CometBFT `max_bytes` per-tx ceiling (~1 MiB on devnet config). Each new blob re-triggers `age.Decrypt` + `io.ReadAll` on every validator via `ProposalHandler.SetBlockList` .

Recommendation

Impose a maximum blob size in `ValidateBasic()` before any other work (for example, 64 KiB — large enough for a realistic sanctioned-address list, small enough that a single message cannot pressure validators). Reject blobs above the cap with a descriptive error. Additionally, rate-limit `MsgStoreBlockList` so a compromised admin cannot submit a new blob on every block.

Alleviation

[Autheo, 05/05/2026]: https://github.com/autheo-blockchain/autheo-chain-core/commit/a65899b37d927571f6a94c8fa0c52353ab69484a

---

[CertiK, 05/06/2026]: The team heeded the advice and resolved the issue by adding a `MaxBlockListBlobSize = 64 KiB` constant in `x/autheo/types/keys.go` and enforcing it as the first check in `MsgStoreBlockList.ValidateBasic()` — before address parsing or `age.Decrypt` — so a compromised admin can no longer force every validator to allocate megabytes per block on a malicious blob, in commit a65899b.

ACA-23 Placeholder Upgrade Handler `v1.6` Pre-Registered On A Pre-Launch Chain — Last-Writer-Wins Silent-Migration Risk

Category	Severity	Location	Status
Coding Issue	Minor	app/app.go: 1042; app/storeloader.go: 22; app/upgrades.go: 14, 18	Resolved

Description

`RegisterUpgradeHandlers` at `app/upgrades.go:13-19` pre-registers a no-op handler for plan name `"v1.6"` (commented `// TBD`) on a chain that has no prior versions to upgrade from.

`UpgradeKeeper.SetUpgradeHandler` is map-insert, which means any other upgrade with name v1.6 could be overwritten/override the existing upgrade handler. When the team plans a real upgrade and reuses the name `"v1.6"`, ordering in `app.New()` determines whether the real handler overrides the current one or vice-versa.

Recommendation

Preferred: delete the placeholder wiring entirely until a real upgrade is planned:

- Remove `RegisterUpgradeHandlers` from `app/upgrades.go`.
- Remove the `storeLoaderOverritten` branch at `app/app.go:1042-1046`; call `app.SetStoreLoader(MaxVersionStoreLoader(qmsVersion))` unconditionally.
- Remove the unused `MaxVersionUpgradeStoreLoader` from `app/storeloader.go` (its closure also has a nil-deref on `storeUpgrades.Renamed` if called with nil).
- Fix the `overritten` → `overridden` typo if the branch is kept for any reason.

Alternative (if the team wants to keep the scaffolding to rehearse the upgrade flow): rename the plan to a sentinel that cannot collide with any real version, e.g. `"scaffold-placeholder-DO-NOT-USE"`.

Alleviation

[Autheo, 05/05/2026]: https://github.com/autheo-blockchain/autheo-chain-core/commit/98ceaf5728fa6475a1eb603cf33d853b8b0d1a9e

---

[CertiK, 05/06/2026]: The team heeded the advice and resolved the issue by deleting the placeholder `v1.6` upgrade handler from `app/upgrades.go` and the unused `MaxVersionUpgradeStoreLoader` from `app/storeloader.go`, simplifying the `app.go` wiring to always install `MaxVersionStoreLoader`, so an empty no-op handler can no longer mask a missed migration on a real chain upgrade, in commit 98ceaf5.

ACA-24 Address Conversion Functions In Event Decoders Silently Produce Invalid Addresses On Parse Failure

Category	Severity	Location	Status
Coding Issue	Minor	x/autheo/events/decoders.go: 135~140, 185~187; x/autheo/keeper/evm_hooks.go: 28~44	Resolved

Description

Two address conversion functions in `decoders.go` silently produce invalid addresses on parse failure, corrupting relayer precompile EVM logs:

1. `convertAddress` (line 185-187): discards bech32 error via `addr, _ = ...` → zero address (`0x0`) emitted as receiver
2. `ConvertAccAddressFromBech32` (line 57-63): returns raw string on failure → keccak-hashed into meaningless indexed topic

The ICS-20 receiver is attacker-controlled from any counterparty chain. The chain has 5 live PostTxProcessing handlers (app.go:788-794) that trigger fund transfers, IBC sends, and license minting from EVM logs — corrupted logs share the same receipt pipeline.

Proof of Concept

1. Attacker sends ICS-20 transfer with receiver = "autheo1invalidchecksum" from counterparty
2. convertAddress: prefix matches, decode fails, error discarded → 0x0 in EVM log
3. ConvertAccAddressFromBech32: non-parseable string → raw string returned → keccak-hashed into bogus topic

Recommendation

Propagate parse errors in both functions instead of silently falling back. Replace `addr, _ = ...` with proper error handling; replace the raw-string fallback with `return nil, err`.

Alleviation

[Autheo, 05/05/2026]: https://github.com/autheo-blockchain/autheo-chain-core/commit/1dfeff13a960738ab002aa4dfda0de93d648c21d

---

[CertiK, 05/06/2026]: The team heeded the advice and resolved the issue by reversing the error handling in `ConvertAccAddressFromBech32` (return wrapped error instead of falling back to the raw string) and replacing the `_` discards inside `convertAddress` for both `ValAddressFromBech32` and `AccAddressFromBech32` with `parsed, err :=` and a wrapped error return, so a malformed address now surfaces to callers instead of producing a wrong-typed event attribute, in commit 1dfeff1.

ACA-25 Relayer Precompile ABI Not Updated For Ibc-Go V10, Breaking EVM Event Emission For Token Packets And Timeout Refunds

Category	Severity	Location	Status
Coding Issue	Minor	x/autheo/events/decoders.go: 88; x/autheo/events/events.go: 19~39, 46~51	Resolved

Description

The relayer precompile ABI was designed for ibc-go v8 but the chain uses ibc-go v10 (go.mod: v10.1.1). Two event types are broken:

1. **FungibleTokenPacket**: ABI declares a `tokens` tuple-array field. ibc-go v10 emits flat `denom` / `amount` attributes. `ConvertTokens` exists (decoders.go:88) but is never wired into `RelayerValueDecoders`. ConvertEvent fails → EVM log silently dropped.

2. **Timeout**: ABI declares `refundDenom` / `refundAmount`. ibc-go v10 emits only `refund_tokens`. ConvertEvent looks for `refund_denom` per the ABI → not found → EVM log silently dropped.

In both cases the IBC operation succeeds at the Cosmos layer. Only the EVM log is missing.

Proof of Concept

1. IBC transfer completes or times out via relayer precompile
2. ConvertEvent iterates ABI's declared fields, looks for matching attributes
3. For FungibleTokenPacket: 'tokens' not found in v10 event → error
4. For Timeout: 'refund_denom' not found (v10 emits 'refund_tokens') → error
5. emitNativeEvents catches error, logs, continues — EVM log not emitted

Recommendation

Regenerate the relayer ABI bindings to match ibc-go v10's event schema. Wire `ConvertTokens` into `RelayerValueDecoders` and update the Timeout event to use `refund_tokens`.

Alleviation

[Autheo, 05/07/2026]: https://github.com/autheo-blockchain/autheo-chain-core/commit/9030890a474b2ff2f148f1180491566bcc3543da

[CertiK, 05/07/2026]: The team heeded the advice and resolved the issue by introducing `contractOwnedByDenom`, `ensureContractNotMapped`, `ensureDenomNotMapped`, and `deleteReverseIfOwned` helpers in `x/autheo/keeper/keeper.go` and rewriting `RegisterContractForDenom` and `GetDenomByContract` around them, so the

forward (`denom → contract`) and reverse (`contract → denom`) indices stay consistent and the auto-deploy / external-mapping paths can no longer collide on the same denom or leave a stale reverse entry pointing at a no-longer-current contract, in commit 9030890.

ACA-26 ICA Precompile SubmitMsgs Silently Truncates Uint256 Timeout To Uint64, Causing Unintended Packet Expiry

Category	Severity	Location	Status
Logical Issue	Minor	x/autheo/keeper/precompiles/ica.go: 178~204	Resolved

Description

The Solidity-facing ABI of the ICA precompile exposes `submitMsgs(string connectionID, bytes data, uint256 timeout)`, so callers can pass any `uint256` timeout. Inside `IcaContract.Run`, the decoded value is forwarded into `MsgSendTx.RelativeTimeout` using `timeout.Uint64()` without any range check. `RelativeTimeout` is a `uint64`, so any value above `2^64 - 1` is silently reduced to its lowest 64 bits (e.g. `2^64 + 1` becomes `1`).

The call still succeeds and returns a valid sequence number, so the caller has no way to detect that the effective timeout is not what was requested. Downstream contracts that record the sequence as "pending" continue to wait on an ICA packet that was actually queued with a near-immediate expiry, leading to avoidable timeout failures and workflow disruption.

Recommendation

Reject timeouts that do not fit into `uint64` so callers receive a deterministic error instead of silent truncation. Alternatively, change the Solidity ABI to use `uint64 timeout` so the type mismatch is resolved at the interface boundary and callers cannot request unrepresentable values.

Alleviation

[Autheo, 05/05/2026]: https://github.com/autheo-blockchain/autheo-chain-core/commit/4a50542ea44c1b88550a288de41e9c14ccd12806

[CertiK, 05/06/2026]: The team heeded the advice and resolved the issue by adding an explicit `timeout.IsUint64()` check in `IcaContract.Run` before constructing the `InterchainAccountPacketData`, so an ABI-declared `uint256` timeout that exceeds `MaxUint64` returns a deterministic error rather than silently truncating to the low 64 bits and producing a packet timeout the caller did not request, in commit 4a50542.

ACA-27 ERC-721 Transfer Handler Reverts Entire EVM Transaction On Uint256 Token IDs, Breaking Standards-Compliant License Contracts

Category	Severity	Location	Status
Logical Issue	Minor	x/license/keeper/evm_hooks.go: 77~100	Acknowledged

Description

ERC-721 specifies `tokenId` as `uint256`, but `ERC721TransferHandler.Handle` in `x/license/keeper/evm_hooks.go` hard-fails any Transfer event whose `tokenId` does not fit in `uint64`. The handler returns an error from the post-transaction hook, which in the Cosmos-EVM pipeline propagates up and reverts the entire EVM transaction, not just the license-sync side effect. The downstream native flow (`OnEvmErc721Transfer`, license module storage) is modeled around `uint64` IDs, so this is a structural boundary between ERC-721 semantics and the native license identifier space rather than a parser bug.

For the currently configured Sovereign / Prime / Core license contracts, IDs may be assigned in the `uint64` range by convention, in which case the condition is not triggered today. However, any future license contract or upgrade that uses `uint256` IDs (for example, hash-derived IDs, chain-prefixed IDs, or any pattern above `2^64 - 1`) will cause every single transfer of those NFTs to revert at the EVM layer, not just the license synchronization.

Recommendation

Align the license synchronization boundary with ERC-721 semantics by explicitly handling `uint256` token IDs instead of reverting on values above `uint64`. If the native license module must remain limited to `uint64`, enforce that constraint at the system boundary in a predictable and documented way so recognized license contracts cannot emit unsupported IDs.

Update the integration design so unsupported token IDs do not unexpectedly revert the entire EVM transaction. Consider either rejecting non-conforming license contracts during configuration, constraining token issuance to the supported range, or treating out-of-range events as unsupported without propagating a hard failure through the post-transaction hook.

Alleviation

[Autheo, 05/05/2026]: By business logic, the data type is sufficient to handle token IDs.

[CertiK, 05/06/2026]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement, since the business logic would not trigger this condition.

ACA-28 Token Mapping Keeper Missing Invariant Guards In SetExternalContractForDenom, SetAutoContractForDenom And GetDenomByContract

Category	Severity	Location	Status
Coding Issue	Minor	x/autheo/keeper/keeper.go: 118~127, 129~146, 187~192	Resolved

Description

The token-mapping keeper in `x/autheo/keeper/keeper.go` is missing three interdependent invariant guards that together allow stale reverse mappings, block idempotent governance updates, and permit conflicting external/auto mappings to co-exist. All three are addressed by a single Cronos upstream patch.

(a) `SetExternalContractForDenom` rejects idempotent self-remap (lines 129-146)

```
func (k Keeper) SetExternalContractForDenom(ctx sdk.Context, denom string, address
common.Address) error {
    _, found := k.GetDenomByContract(ctx, address)
    if found {
        return fmt.Errorf("the contract is already registered: %s", address.Hex())
    }
    ...
}
```

The guard fires when `address` already has ANY reverse mapping, even if that mapping points at the same `denom` being supplied. `GetDenomByContract` does not compare the returned existing denom to the input, so the self-case `(D, A) -> (D, A)` is treated identically to a conflicting remap `(D', A) -> (D, A)`.

Consequences:

- A `MsgUpdateTokenMapping` or `TokenMappingProposal` that re-registers the same `(denom, contract)` pair — e.g., to refresh metadata (symbol / decimals) registered alongside the mapping in `RegisterOrUpdateTokenMapping` at `keeper.go:216-277` — fails with `the contract is already registered`.
- Because `SetDenomMetaData` (line 254) commits before the failing `SetExternalContractForDenom` (line 257), the tx rolls back atomically, leaving metadata unchanged.
- Governance operators must use a two-proposal dance: first `MsgUpdateTokenMapping` with empty `Contract` to delete, then re-register — an avoidable operational burden.

(b) `SetAutoContractForDenom` leaves stale reverse mappings and lacks external-exclusivity (lines 187-192)

```
func (k Keeper) SetAutoContractForDenom(ctx sdk.Context, denom string, address
common.Address) {
    store := ctx.KVStore(k.storeKey)
    store.Set(types.DenomToAutoContractKey(denom), address.Bytes())
    store.Set(types.ContractToDenomKey(address.Bytes()), []byte(denom))
}
```

Compared with the sibling `SetExternalContractForDenom`, this function is missing three safeguards:

1. **No stale reverse cleanup.** If the forward mapping already points at `oldAddress` and is overwritten to a new `address`, the reverse entry `ContractToDenomKey(oldAddress) -> denom` remains in the store. `GetDenomByContract(oldAddress)` then returns a false positive indefinitely.
2. **No external-exclusivity check.** If a denom already has an external mapping (non-source coin), `SetAutoContractForDenom` silently co-exists with it, breaking the upstream invariant that a given non-source denom has exactly one active contract.
3. **No error return.** The function cannot signal an invalid operation to callers.

In practice the runtime caller at `x/autheo/keeper/evm.go:98-110` gates with `if !found` on `GetContractByDenom`, so the forward mapping is not overwritten during normal auto-deployment. The exposure path that remains is genesis import — `x/autheo/genesis.go:41` iterates `AutoContracts` entries without de-duplication or invariant checks, so a malformed genesis (operator error, or a regenesis that replays prior auto-deploy addresses after a key change) can seed the store with contradictory forward/reverse entries that the keeper cannot detect later.

(c) `GetDenomByContract` has no forward cross-check (lines 118-127)

```
func (k Keeper) GetDenomByContract(ctx sdk.Context, contract common.Address) (denom
string, found bool) {
    store := ctx.KVStore(k.storeKey)
    bz := store.Get(types.ContractToDenomKey(contract.Bytes()))
    if len(bz) == 0 { return "", false }
    return string(bz), true
}
```

Consumers of `GetDenomByContract` (precompiles, JSON-RPC, event decoders in `x/autheo/events/decoders.go`) cannot detect that a reverse entry is stale. Combined with (b), this means any legacy-stale reverse mapping is silently returned as authoritative.

Recommendation

Port Cronos PR #1998 (`b91e942b`) as a single atomic change — the three helpers and guards interlock and must ship together:

1. **Introduce `ensureContractNotMapped`** and use it as the uniform guard in both `SetExternalContractForDenom` and `SetAutoContractForDenom`:

```
func (k Keeper) ensureContractNotMapped(ctx sdk.Context, denom string, address
common.Address) error {
    store := ctx.KVStore(k.storeKey)
    bz := store.Get(types.ContractToDenomKey(address.Bytes()))
    if len(bz) == 0 { return nil }
    existingDenom := string(bz)
    if existingDenom == denom { return nil }
    if k.contractOwnedByDenom(ctx, existingDenom, address) {
        return errors.Wrap(types.ErrContractAlreadyRegistered, ...)
    }
    store.Delete(types.ContractToDenomKey(address.Bytes()))
    return nil
}
```

2. **Introduce** `deleteReverseIfOwned` **and** `contractOwnedByDenom` **helpers** for precise reverse-mapping management.

3. **Change** `SetAutoContractForDenom` **to return** `error` and enforce both the external-exclusivity check and `ensureContractNotMapped`:

```
func (k Keeper) SetAutoContractForDenom(ctx sdk.Context, denom string, address
common.Address) error {
    store := ctx.KVStore(k.storeKey)
    if _, found := k.getExternalContractByDenom(ctx, denom); found &&
!types.IsSourceCoin(denom) {
        return errors.Wrap(types.ErrExternalMappingExists, denom)
    }
    if err := k.ensureContractNotMapped(ctx, denom, address); err != nil {
        return err
    }
    store.Set(types.DenomToAutoContractKey(denom), address.Bytes())
    store.Set(types.ContractToDenomKey(address.Bytes()), []byte(denom))
    return nil
}
```

4. **Update** `SetExternalContractForDenom` to retire any existing auto mapping for non-source denoms.

5. **Add the forward cross-check** in `GetDenomByContract` so consumers never see stale reverse entries.

6. **Update call sites** — `x/autheo/keeper/evm.go:107` and `x/autheo/genesis.go:41` — to handle the new error return from `SetAutoContractForDenom`.

7. **Add a genesis-import invariant check** in `x/autheo/genesis.go:41` to detect conflicting `AutoContracts` entries at chain start.

Alleviation

[Autheo, 05/07/2026]: https://github.com/autheo-blockchain/autheo-chain-core/commit/9030890a474b2ff2f148f1180491566bcc3543da

---

[CertiK, 05/07/2026]: The team heeded the advice and resolved the issue by rejecting any address in the precompile range (`addr.Big() < 256`) inside `ensureContractCode` and migrating the deployment check from `evmKeeper.GetAccount` + `IsEmptyCodeHash` to `evmKeeper.Code(ctx, *evmtypes.QueryCodeRequest)` (with the corresponding swap on the `EvmKeeper` interface in `x/autheo/types/interfaces.go`), so a precompile address can no longer be installed as a CRC-21 token mapping and the contract-code presence check goes through the canonical EVM query rather than an indirect account-state path, in commit 9030890.

ACA-29 ICA And Relayer Precompiles Panic On Short Or Malformed Calldata

Category	Severity	Location	Status
Coding Issue	Minor	x/autheo/keeper/precompiles/ica.go: 102~123; x/autheo/keeper/precompiles/relayer.go: 134~177	Resolved

Description

Two sibling precompiles reachable from any EVM caller can panic at the gas-metering pre-pass or the main execution path when given inputs outside the narrow shape they assume. Both are addressed by a single Cronos upstream patch.

(a) ICA precompile panics on sub-4-byte calldata (`x/autheo/keeper/precompiles/ica.go:102-123`)

```
func (ic *IcaContract) RequiredGas(input []byte) uint64 {
    var methodID [4]byte
    copy(methodID[:], input[:4])   // slice bounds out of range when len(input) < 4
    ...
}

func (ic *IcaContract) Run(evm *vm.EVM, contract *vm.Contract, readonly bool)
([]byte, error) {
    methodID := contract.Input[:4]   // slice bounds out of range when len(Input) <
4
    ...
}
```

Both entry points panic on calldata shorter than four bytes. The `copy` is not protective — Go evaluates the `input[:4]` slice expression **before** `copy` runs, so the panic fires on the slice, not the copy.

(b) Relayer precompile `RequiredGas` panics on non-ICS-20 IBC packets and short input (`x/autheo/keeper/precompiles/relayer.go:134-177`)

Four unconditional `panic(err)` sites on the `recvPacket` path, plus a missing length guard:

```
method, err := irelayerABI.MethodById(methodID[:])
if err != nil {
    panic(err)                                          // L143: unknown
selector
}
args, err := method.Inputs.Unpack(input[4:])
if err != nil {
    panic(err)                                          // L148: ABI unpack
}
i, _ := args[0].([]byte)
var msg ibctypes.MsgRecvPacket
if err := bc.cdc.Unmarshal(i, &msg); err != nil {
    panic(err)                                          // L153: proto unmarshal
}
var data ibctransfertypes.FungibleTokenPacketData
if err := ibctransfertypes.ModuleCdc.UnmarshalJSON(msg.Packet.GetData(), &data); err
!= nil {
    panic(err)                                          // L157: assumes ICS-20
}
```

There is also no `len(input) < 4` guard before `copy(methodID[:], input[:4])`.

The hardcoded ICS-20 assumption at L157 creates an **asymmetry** with the `Run` method: `Run` dispatches generically to `bc.ibcKeeper.RecvPacket(...)` which the IBC core router handles for any registered app (ICS-20, ICA, ICS-29 fee, ICS-721 NFT, custom middleware). Both the ICA Controller and ICA Host keepers are registered in the IBC router at `app/app.go:321-322, 635-656`, so non-ICS-20 packets are reachable on-chain via normal IBC operation. Any EVM caller that invokes `recvPacket` on the relayer precompile with a valid non-ICS-20 packet crashes at gas estimation.

Reachability

Both precompiles are reachable from any EOA transaction or internal `CALL` with 0-3 bytes of calldata or malformed payload — no privileged access required. The precompile addresses are live on the chain.

Containment

- Tx path: Cosmos SDK `BaseApp.runTx` recovers panics in a `defer`. The tx aborts with an error and the gas paid is forfeited; the node does **not** halt.
- `eth_call` / `eth_estimateGas`: the ABCI Query path also has deferred recovery, so the RPC returns an error instead of crashing the node.
- Because the panic is deterministic on identical calldata, validators produce identical outcomes — no consensus fork.

Impact

- Failed transactions with forfeited gas on any short or malformed call.
- Relayers and contracts attempting to use the relayer precompile as a generic IBC relay endpoint (for which `Run` is actually capable) are blocked at the `RequiredGas` stage. Any smart-contract integration that routes ICA packets

through the precompile is non-functional.

Upstream fix

Cronos fixed both precompiles in PR #1941 (commit `5cabab48`), titled "fix: remove panic in contracts and add guards for ica and ibc logic." The patch:

- Adds explicit `len(input) < 4` / `len(contract.Input) < 4` guards in both ICA precompile entry points.
- Replaces every `panic(err)` in the relayer `RequiredGas` path with structured logging and a graceful fallback that returns the method's default gas cost via a new `getRequiredGas(DefaultGasRequired, baseCost, intrinsicGas)` helper.

```
// cronos/x/cronos/keeper/precompiles/ica.go (post-fix)
func (ic *IcaContract) RequiredGas(input []byte) uint64 {
    if len(input) < 4 { return baseCost }
    ...
}

func (ic *IcaContract) Run(evm *vm.EVM, contract *vm.Contract, readonly bool)
([]byte, error) {
    if len(contract.Input) < 4 { return nil, errors.New("input too short") }
    ...
}

// cronos/x/cronos/keeper/precompiles/relayer.go (post-fix excerpt)
if inputLen := len(input); inputLen < 4 {
    bc.logger.Error("input length is less than 4")
    return getRequiredGas(DefaultGasRequired, baseCost, intrinsicGas)
}
...
if err := bc.cdc.Unmarshal(i, &msg); err != nil {
    bc.logger.Error("failed to unmarshal MsgRecvPacket", "error", err)
    return getRequiredGas(gasRequiredByMethod, baseCost, intrinsicGas)
}
if err := ibctransfertypes.ModuleCdc.UnmarshalJSON(msg.Packet.GetData(), &data); err
!= nil {
    bc.logger.Error("failed to unmarshal FungibleTokenPacketData", "error", err)
    return getRequiredGas(gasRequiredByMethod, baseCost, intrinsicGas)
}
```

Recommendation

Port Cronos PR #1941 (`5cabab48`) to both precompile files as a single change — the patterns interlock:

1. **ICA precompile — add length guards in both entry points:**

```
func (ic *IcaContract) RequiredGas(input []byte) uint64 {
    if len(input) < 4 { return baseCost }
    ...
}

func (ic *IcaContract) Run(evm *vm.EVM, contract *vm.Contract, readonly bool)
([]byte, error) {
    if len(contract.Input) < 4 { return nil, errors.New("input too short") }
    ...
}
```

2. **Relayer precompile — replace every `panic(err)` with structured log + default-gas fallback:**

```
if inputLen := len(input); inputLen < 4 {
    bc.logger.Error("relayer input length < 4")
    return getRequiredGas(DefaultGasRequired, baseCost, intrinsicGas)
}
...
if err := ibctransfertypes.ModuleCdc.UnmarshalJSON(msg.Packet.GetData(), &data); err
!= nil {
    bc.logger.Error("not an ICS-20 packet; using default relayer gas", "error", err)
    return getRequiredGas(gasRequiredByMethod, baseCost, intrinsicGas)
}
```

3. **Remove the load-bearing ICS-20 assumption** from the relayer `RequiredGas` altogether if feasible — the actual work in `Run` routes through `ibcKeeper.RecvPacket` which is app-agnostic, so `RequiredGas` does not need to decode `Packet.Data` beyond ICS-20-specific gas heuristics (and the heuristic can default gracefully for other app schemas).

4. **Add a precompile fuzz harness** (`go test -fuzz=...`) that calls every precompile with 0-3 byte inputs, unknown method selectors, and payloads from each registered IBC app (ICA Controller, ICA Host, future ICS-721) and asserts no panic. Gate it in CI.

5. Sweep `x/autheo/keeper/precompiles/` for any other call sites that index into raw calldata without a length check; the relayer and ICA precompiles are the two entry points covered by PR #1941 but the same pattern may exist elsewhere.

Alleviation

[Autheo, 05/07/2026]: https://github.com/autheo-blockchain/autheo-chain-core/commit/9030890a474b2ff2f148f1180491566bcc3543da

[CertiK, 05/07/2026]: The team heeded the advice and resolved the issue by capturing the `*MsgEthereumTxResponse` from `CallEVM` in the IBC callback path and returning `fmt.Errorf("IBC callback EVM execution reverted: %s", res.VmError)` whenever `res.Failed()` (so a reverted callback now aborts the IBC packet instead of being silently treated as success), and by changing `SetAutoContractForDenom` to return `error` with both call sites — `x/autheo/genesis.go` `InitGenesis` and `x/autheo/keeper/evm.go` IBC auto-deploy — propagating that error, in commit 9030890.

ACA-30 Relayer Precompile Panics On String-Argument Selectors Due To Unconditional Args[0].([]Byte) Assertion

Category	Severity	Location	Status
Coding Issue	Minor	x/autheo/keeper/precompiles/relayer.go: 195~248	Resolved

Description

`RelayerContract.Run` at `x/autheo/keeper/precompiles/relayer.go:200` performs an unchecked type assertion before the method-name switch:

```
args, err := method.Inputs.Unpack(contract.Input[4:])
...
input := args[0].([]byte)    // panics when args[0] is not []byte
switch method.Name { ... }
```

The registered ABI declares two methods whose first argument is `string`, not `bytes`:

- `registerPayee(string, string, address)`
- `registerCounterpartyPayee(string, string, string)`

For either selector, `Unpack` returns a Go `string` as `args[0]`, so the assertion panics with `interface conversion: interface {} is string, not []uint8`. The dispatch switch also has no cases for these two methods; even with the panic guarded they would return `unknown method`. They are effectively dead code, but the panic fires first and yields an opaque error.

`BaseApp.runTx` recovers the panic into a failed tx (no node halt). Impact is limited to the two non-functional methods. The pattern is present unchanged upstream in Cronos.

Recommendation

Move the type assertion into the bytes-argument cases or use comma-ok form:

```
case CreateClient, UpdateClient /* ...bytes-arg cases... */:
    input, ok := args[0].([]byte)
    if !ok { return nil, fmt.Errorf("expected bytes for %s", method.Name) }
    ...
case RegisterPayee, RegisterCounterpartyPayee:
    return nil, fmt.Errorf("%s not supported via precompile", method.Name)
```

Preferably, remove `RegisterPayee` / `RegisterCounterpartyPayee` from the Solidity interface and the gas table so they never reach dispatch.

Alleviation

[Autheo, 05/05/2026]: https://github.com/autheo-blockchain/autheo-chain-core/commit/f2ae6122b519ec25bacedd0e4a87903f23008721

[CertiK, 05/06/2026]: The team heeded the advice and resolved the issue by short-circuiting the string-argument methods (`RegisterPayee` , `RegisterCounterpartyPayee`) in `RelayerContract.Run` with an explicit "not supported via the relayer precompile" error before the `args[0].([]byte)` assertion, and converting that assertion to the comma-ok form so any other unexpected argument shape returns a typed error rather than panicking the EVM, in commit f2ae612.

ACA-31 TransferTokens Emits Requested EvmDenom Instead Of Rounded Burned/Bridged Amounts

Category	Severity	Location	Status
Coding Issue	Minor	x/autheo/keeper/ibc.go: 93~130; x/autheo/keeper/msg_server.go: 63~81	Resolved

Description

`msgServer.TransferTokens` emits `types.NewTransferTokensEvent(msg.From, msg.To, msg.Coins)` after `IbcTransferCoins` returns nil. For `evmParams.EvmDenom`, `Keeper.IbcTransferCoins` rounds down to 8 effective decimals, skips the coin when `amountToBurn` is zero, and may bridge less than the raw request amount. The event and deferred telemetry still use the original `msg.Coins`, so off-chain indexers, analytics, or any consumer that treats `transfer_tokens` as authoritative can record inflated or phantom transfer amounts. This is an observability and accounting-signal mismatch, not an on-chain balance mint.

Recommendation

Emit the event and telemetry from the effective coins processed by `IbcTransferCoins`, or return a structured list of per-denom actually burned, bridged, and skipped. Do not emit a success transfer for coins that were fully skipped as dust.

Alleviation

[Autheo, 05/05/2026]: https://github.com/autheo-blockchain/autheo-chain-core/commit/db24c238acf336087fc43526ea40238017ed6310

[CertiK, 05/06/2026]: The team heeded the advice and resolved the issue by changing `IbcTransferCoins`'s signature to return `(processedCoins sdk.Coins, error)` accumulating only the coins actually burned and bridged (skipping EVM-denom amounts that round to zero as dust), and updating `MsgServer.TransferTokens` to emit the `transfer_tokens` event from `processedCoins` (and suppress it entirely when nothing survived rounding) so indexers no longer see phantom or inflated amounts, in commit db24c23.

ACA-32 Emissions GetParams Returns Unvalidated Params And Defaults When ParamsKey Is Absent

Category	Severity	Location	Status
Coding Issue	Minor	x/emissions/keeper/state.go: 11~20	Resolved

Description

`Keeper.GetParams` returns `types.DefaultParams()` when `types.ParamsKey` is missing, and returns unmarshalled storage bytes without `params.Validate()` or backfilling of new fields. A missing or partially migrated `ParamsKey` can therefore drive emissions logic with defaults or incomplete structs, masking configuration or upgrade errors. Risk is upgrade and initialization robustness, not a demonstrated unprivileged store write to arbitrary bytes.

Recommendation

Fail closed on missing params at module init or first use, or require explicit genesis/migration to set `ParamsKey`. After `MustUnmarshal`, call `params.Validate()` and normalize legacy fields. Enforce the same invariants in migrations that touch emissions params.

Alleviation

[Autheo, 05/05/2026]: https://github.com/autheo-blockchain/autheo-chain-core/commit/30179c60391d9e1598227c45de0a2ad495200c15

[CertiK, 05/06/2026]: The team heeded the advice and resolved the issue by adding a post-unmarshal `params.Validate()` to `emissions.Keeper.GetParams` that panics on validation failure (silent operation with broken consensus-critical params is worse than a halt), and logging a clear error when the `ParamsKey` is missing entirely so operators can detect the missed migration, in commit 30179c6.

ACA-33 TrustedRelayer Check Conflates Authorized Signer With Voucher Beneficiary, Breaking MsgConvertVouchers

Category	Severity	Location	Status
Coding Issue	Minor	x/autheo/keeper/ibc.go: 21~77; x/autheo/keeper/msg_server.go: 27~60	Resolved

Description

When `params.TrustedRelayer` is non-empty, `ConvertVouchers` requires `msg.Address` to equal that relayer address. The same field is then passed to `ConvertVouchersToEvmCoins`, which debits voucher coins and credits native/EVM balances for that same `from` account. As a result the relayer cannot submit a conversion on behalf of an end-user (the check would require `msg.Address` to be the user, which fails the relayer gate), and an end-user cannot convert their own vouchers (their address is not the trusted relayer). The only consistent use is converting vouchers that are already held by the relayer account. IBC auto-credit paths may still deliver assets to users without this message.

Recommendation

Split authorized relayer from beneficiary: e.g. validate the transaction signer (or a dedicated `relayer` field) against `TrustedRelayer`, and use a separate `beneficiary` / `receiver` bech32 for `ConvertVouchersToEvmCoins`. Alternatively document that only the relayer self-converts and route user flows exclusively through IBC hooks; if that is intended, align params and docs and consider removing the misleading dual use of `Address`.

Alleviation

[Autheo, 05/05/2026]: Fixed in commit https://github.com/autheo-blockchain/autheo-chain-core/commit/616e3156f28b7b4ad96fafda9c8f3fc2b26d369c

[CertiK, 05/06/2026]: The team heeded the advice and resolved the issue in commit 616e3156f28b7b4ad96fafda9c8f3fc2b26d369c. MsgConvertVouchers no longer overloads trusted_relayer against the conversion account, so holders can convert their own vouchers again; the dedicated relayer param was removed rather than split.

ACA-34 X/License DestEvmTokenMap Index Not Serialized In Genesis Import/Export Enabling Duplicate License Mints On Genesis Import

Category	Severity	Location	Status
Coding Issue	Minor	x/license/keeper/genesis.go: 8~70; x/license/keeper/keeper.go: 208~218, 352~387; x/license/types/keys.go: 43	Resolved

Description

The `DestEvmTokenMap` reverse index at `x/license/types/keys.go:43` (prefix `0x0b`) maps `(evmContract, evmTokenId) → licenseId`. It is written once at mint time by `setDestEvmTokenMap` (`x/license/keeper/keeper.go:215`) and read by `GetLicenseIdByEvmToken` at `keeper.go:358`, which `OnEvmErc721Transfer` (`keeper.go:370-387`) uses as a duplicate-mint guard.

`ExportGenesis` / `InitGenesis` at `x/license/keeper/genesis.go` do not serialize this index, and the `GenesisState` proto has no field for it. `InitGenesis` also cannot reconstruct it from the exported `LicenseRecord`s because the record only carries the Hyperlane-side identifiers (`OriginChain`, `BridgeContract`, `TokenId`), not the local `(evmContract, evmTokenId)` pair.

Consequence: after a state export / import (chain upgrade via genesis, state sync from a freshly-exported genesis, or a hard fork), `GetLicenseIdByEvmToken` returns `(0, false)` for every pre-existing EVM-mapped license. The next ERC-721 `Transfer` event on the same token hits the `!found` branch in `OnEvmErc721Transfer` and mints a NEW license with a fresh licenseID, duplicating the underlying entitlement. Both the original license (still keyed by its old ID in the `Licenses` export) and the newly minted license would now exist on-chain, doubling rewards / bindings for the same EVM token.

Impact bounded by the operational rarity of genesis export / import flows, but severity is correctness-critical when they do occur.

Recommendation

Add Genesis Import/Export for that mapping.

1. Add a `DestEvmTokenEntry { bytes evm_contract; uint64 evm_token_id; uint64 license_id; }` repeated field to `GenesisState` in the license module proto.
2. In `ExportGenesis`, iterate the `DestEvmTokenMapPrefix` subrange and populate the new field.
3. In `InitGenesis`, loop the new entries and call `setDestEvmTokenMap` for each.

4. Add an integration test (`x/license/keeper/genesis_test.go`) that mints an EVM-mapped license, exports genesis, re-imports it, and asserts that `GetLicenseIdByEvmToken` still resolves the original license and that a repeated ERC-721 Transfer for the same token does not mint a duplicate.

Alternative: persist `(evmContract, evmTokenId)` on the `LicenseRecord` itself so the forward licenses export carries the information needed to rebuild the reverse index in `InitGenesis` without adding a new top-level field.

Alleviation

[Autheo , 05/05/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/19f6cac99a64ee5ed5395bcb6b32a2ae1f465416

[CertiK , 05/05/2026]: `GenesisState` adds `dest_evm_token_map` (`DestEvmTokenEntry`); `ExportGenesis` fills it via `GetAllDestEvmTokenMap` , and `InitGenesis` restores the reverse index with `setDestEvmTokenMap` so `GetLicenseIdByEvmToken` survives export/import—19f6cac99a64ee5ed5395bcb6b32a2ae1f465416.

ACA-35 Sovereign License Loops Use First-Match-Wins Semantics In Both Unjail And EditValidator Checks

Category	Severity	Location	Status
Coding Issue	Minor	x/license/keeper/msg_server.go: 277~292, 330~356; x/license/module.go: 27~32; x/licensedstaking/keeper/jail_interceptor.go: 446~494; x/licensedstaking/keeper/msg_server.go: 103~147	Resolved

Description

Two sibling loops over sovereign licenses use first-match-wins semantics. Either branch inside the loop returns immediately on the first Sovereign encountered, without examining remaining entries:

1. `checkSovereignLicenseForUnjail` at `x/licensedstaking/keeper/jail_interceptor.go:446-494` — on first Sovereign, returns `ErrUnauthorized` if REVOKED, returns `nil` otherwise.

2. `EditValidator` **override** at `x/licensedstaking/keeper/msg_server.go:120-139` — on first Sovereign, returns `ErrUnauthorized` if owner mismatches or status is REVOKED, sets `foundSovereign=true` and breaks otherwise.

Both are safe under two currently-enforced invariants:

- **Sovereign** `MaxBindings=1` (`x/license/module.go:28-32`), enforced in `BindLicense` at `x/license/keeper/msg_server.go:333-356`. At most one Sovereign binding per validator.
- **Sovereign binds only to owner's own validator address** (`x/license/keeper/msg_server.go:277-292`): `valAddr = sdk.ValAddress(ownerAddr)`, and any explicit `msg.ValidatorAddress` must equal the owner-derived address. A sovereign binding on validator V is structurally owned by V's operator.
- **Revoke deletes the binding before writing REVOKED status** (`x/license/keeper/msg_server.go:680-685`), so a REVOKED Sovereign never appears in `GetLicenseIdsByValidator`.

Under these invariants neither loop can observe multiple Sovereigns, foreign-owned Sovereigns, or REVOKED Sovereigns — so the first-match branches never hide a second valid entry.

Note that `MsgUpsertLicensePolicy` (`x/license/keeper/msg_server.go:27-91`) is reachable via gov authority and can raise Sovereign `MaxBindings`. If that ever happens — or if a future refactor changes revoke ordering or allows sovereign transfers to non-owner validators — both loops begin returning order-dependent decisions. In the unjail path, an attacker who can seed a REVOKED sovereign ahead of a valid one in iteration order blocks unjail. In EditValidator, the same pattern blocks commission/description/min-self-delegation edits.

Recommendation

In both loops, iterate the full license list and prefer a matching non-revoked Sovereign if any exists; reject only when no valid

Sovereign is found:

`checkSovereignLicenseForUnjail`:

```
var sawRevoked bool
for _, lid := range licenseIds {
    license, _ := k.licenseKeeper.GetLicense(ctx, lid)
    if license.Category != SOVEREIGN { continue }
    if license.Status == REVOKED { sawRevoked = true; continue }
    return nil
}
if sawRevoked { return ErrUnauthorized }
return ErrNoSovereignLicense
```

`EditValidator` **override:** replace the two `return ErrUnauthorized` short-circuits with `continue`; only return `ErrUnauthorized` at the end of the loop if no owner-matching non-revoked Sovereign was found.

Alleviation

[Autheo , 05/05/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/47494bcecc636b45220d9791355f04e0e8d0663f

---

[CertiK , 05/05/2026]: `checkSovereignLicenseForUnjail` now scans all bound licenses, skipping revoked sovereigns until a non-revoked one is found; `EditValidator` uses `continue` instead of failing on the first mismatched/revoked sovereign —47494bcecc636b45220d9791355f04e0e8d0663f.

ACA-36 EditValidator O(N) License Scan Spammable Via Unlimited Permissionless Prime/Core Bindings

Category	Severity	Location	Status
Coding Issue	Minor	x/license/keeper/msg_server.go: 290~356; x/license/module.go: 28~46; x/licensedstaking/keeper/msg_server.go: 103~147	Acknowledged

Description

`EditValidator` in `x/licensedstaking/keeper/msg_server.go:103-147` linearly iterates `GetLicenseIdsByValidator(valAddr)` and calls `GetLicense(id)` per iteration to locate the operator's Sovereign license. The loop breaks on the first Sovereign match, so in normal operation the cost is O(index-of-sovereign); in the pathological case (Sovereign last or absent) it walks every binding.

Two design choices combine to make this griefable:

1. **Prime/Core default `MaxBindings = 0`** (`x/license/module.go:38,45`). Zero means unlimited — any number of Prime/Core licenses can bind to the same validator.
2. **Permissionless bind target.** `BindLicense` (`x/license/keeper/msg_server.go:309-322`) only requires that `msg.Owner` own the license and that the target validator is bonded and not jailed. **The validator's operator does not consent.**

Consequently, any Prime/Core license holder can repeatedly bind their licenses to any bonded validator's address. Each binding increases the cost of every subsequent `EditValidator` call made by the legitimate operator — each edit pays one prefix iterator scan + N `GetLicense` KV reads before reaching (or failing to reach) the operator's Sovereign.

This finding shares its root cause with the emissions BeginBlocker finding (`emissions-beginblocker-unbounded-license-scan`), but the attack surfaces differ: F36 is block-level performance; this is per-operator griefing at message dispatch. A single fix — cap Prime/Core `MaxBindings` or add a category-indexed lookup — closes both.

Recommendation

Pick one of (preferably both):

1. **Add per-tier supply / binding caps.** Change the default `CategoryPolicy.MaxBindings` for Prime/Core from `0` (unlimited) to a conservative value appropriate for the product tokenomics, and re-enforce in `BindLicense`.
2. **Index bindings by category.** Maintain a secondary store keyed by `(valAddr, category, licenseID)` so `EditValidator` can look up Sovereign bindings in O(1):

```
store.Set(types.ValidatorCategoryBindingKey(valAddr, SOVEREIGN, licenseID),
[]byte{})
```

Update `BindLicense`, `RebindLicense`, `DeleteBinding`, `RevokeLicense`, and `TransferLicense` to keep the index in sync.

3. **Require target-validator consent** on Prime/Core `BindLicense` calls (e.g., a separate approval list set by the operator via a new msg) so a validator is never forced to carry third-party binding state.

The emissions BeginBlocker (F36) benefits from the same remediation — the active-license index proposed there closes both surfaces.

Alleviation

[Autheo, 05/05/2026]: Acknowledged and deferred. This is a known O(N) scan over validator bindings in EditValidator. The root cause, unlimited Prime/Core MaxBindings with permissionless bind targeting, is understood. There is no fund loss or chain halt risk.

[CertiK, 05/06/2026]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement. The issue is present, but the risk is low enough to be deferred to later fix.

ACA-37 Token Mapping Governance Proposals Lack Input Validation And Contract-Code Existence Check

Category	Severity	Location	Status
Coding Issue	Minor	x/autheo/keeper/keeper.go: 217~259; x/autheo/proposal_handler.go: 16~30; x/autheo/types/proposal.go: 44~48	Resolved

Description

Three layered gaps allow a `TokenMappingChangeProposal` to install a malformed mapping with no rejection:

1. `TokenMappingChangeProposal.ValidateBasic` (`x/autheo/types/proposal.go:44-48`) is a `// TODO` stub returning nil.
2. The proposal handler (`x/autheo/proposal_handler.go:16-30`) only checks `IsValidCoinDenom`; it never calls `c.ValidateBasic()`.
3. The source-coin branch of `RegisterOrUpdateTokenMapping` (`x/autheo/keeper/keeper.go:257`) calls `common.HexToAddress(msg.Contract)` without `IsHexAddress` first — invalid hex silently zero-pads to `0x000...000`. The non-source branch (line 265) does check `IsHexAddress` but neither branch verifies that EVM code exists at the target.

A passing proposal with an empty or malformed `Contract` for a source-coin denom commits a mapping pointing at the zero address. Subsequent ICS-20 conversions for that denom call into `0x000...000` and fail. Governance-gated, no unprivileged exploit path; the gap is defense-in-depth — a typo in a proposal cannot be rejected at submission time and only becomes visible after the vote passes.

Cronos fixed all three layers in PR #1997 (commit `29391d80`): non-stub `ValidateBasic`, handler invocation, and an `ensureContractCode` keeper helper called from both branches. This is the prerequisite for PR #1998 (cleanup on remap), already tracked under MINOR-F17F18.

Proof of Concept

Stub — `x/autheo/types/proposal.go:44-48`:

```
func (tcp *TokenMappingChangeProposal) ValidateBasic() error {
    // TODO
    return nil
}
```

Handler skips it — `x/autheo/proposal_handler.go:20-30`:

```
if !types.IsValidCoinDenom(c.Denom) { ... }
// no c.ValidateBasic() call
return k.RegisterOrUpdateTokenMapping(ctx, &msg)
```

Source-coin branch silently zero-pads — `x/autheo/keeper/keeper.go:257` :

```
if err := k.SetExternalContractForDenom(ctx, msg.Denom,
common.HexToAddress(msg.Contract)); err != nil {
```

Reproduction: submit `TokenMappingChangeProposal{Denom: <source-coin>, Contract: ""}` . After it passes, `query autheo contract-by-denom <denom>` returns `0x0000...0000` .

Recommendation

Port Cronos PR #1997 (`29391d80`):

1. Implement `TokenMappingChangeProposal.ValidateBasic` to require a valid hex `Contract` for source denoms and reject malformed (non-empty, non-hex) `Contract` for non-source denoms.
2. Call `c.ValidateBasic()` in the proposal handler before constructing `MsgUpdateTokenMapping` .
3. Add `ensureContractCode(ctx, contract)` querying `evmKeeper.Code` and rejecting empty bytecode; call it from both branches of `RegisterOrUpdateTokenMapping` . Also add `IsHexAddress` to the source-coin branch.

Port #1997 before #1998 — input validation prevents bad state from being created; #1998 cleans up legacy state.

Alleviation

[Autheo, 05/07/2026]: https://github.com/autheo-blockchain/autheo-chain-core/commit/9030890a474b2ff2f148f1180491566bcc3543da

[CertiK, 05/07/2026]: The team heeded the advice and resolved the issue by adding `c.ValidateBasic()` to the per-coin loop in the `MsgConvertVouchers` proposal handler in `x/autheo/proposal_handler.go` , so coins delivered through the gov-proposal route are now subject to the same denom-format and non-negative-amount validation as those delivered through the message-server path, in commit 9030890.

ACA-38 Sovereign License Lifecycle Transitions Skip Validator Status Checks (Bind + Reinstate)

Category	Severity	Location	Status
Coding Issue	Minor	x/license/keeper/msg_server.go: 711~760	Acknowledged

Description

Two Sovereign-license lifecycle handlers omit validator-status checks. Together they let a license reattach to a permanently-dead validator.

(a) `BindLicense` **Sovereign branch (** `msg_server.go:276-292` **)** sets `valAddr = sdk.ValAddress(ownerAddr)` and writes the binding without `GetValidator` / `IsBonded` / `IsJailed` / `IsTombstoned` . Prime/Core (lines 311-322) check all four; Sovereign does not.

(b) `ReinstateLicense` **(** `msg_server.go:740-760` **)** logs a warning when the validator is jailed but never checks tombstone, and does not clear `LastBoundValidator` .

Compose: gov reinstates a license whose validator was tombstoned → owner re-binds via the Sovereign branch (no status check) → license sits BOUND on a permanently-dead validator. Activation skips (tombstone is permanent jail), so no rewards flow, but the state is stuck until governance revokes again.

Proof of Concept

(a) `msg_server.go:276-292` :

```
case SOVEREIGN:
    valAddr = sdk.ValAddress(ownerAddr)   // no GetValidator / IsBonded /
IsTombstoned
```

(b) `msg_server.go:740-760` :

```
if validator.IsJailed() { Logger.Warn(...) }
// no IsTombstoned check
license.Status = ISSUED
```

Recommendation

(a) Add `GetValidator` + `IsTombstoned` check in the Sovereign bind branch. Decide explicitly whether bind-while-jailed is allowed.

(b) In `ReinstateLicense` , if `validator.IsTombstoned()` , clear `license.LastBoundValidator` (or refuse).

Alleviation

[Autheo, 05/05/2026]: Acknowledged. The stuck-state scenario (BOUND license on a tombstoned validator) carries no economic risk, a tombstoned validator produces no blocks, accrues no rewards and the only consequence is a governance revoke being required to clean up.

[Certik, 05/06/2026]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement, since the issue will not cause any loss of funds or broken state.

ACA-39 Blocklist Validation Skips Cosmos Bank, IBC, And Authz Recipients

Category	Severity	Location	Status
Coding Issue	Minor	app/proposal.go: 145~215	Resolved

Description

`ValidateTransaction` checks signers and EVM-only fields (`MsgEthereumTx.To` , EIP-7702). All other Cosmos msg types pass through. `bank.MsgSend.ToAddress` , `MsgMultiSend.Outputs` , `ibctransfer.MsgTransfer.Receiver` , and `authz.MsgExec.Msgs` are never inspected — blocklisted addresses freely receive Cosmos/IBC funds, and authz-wrapped sends with a non-blocklisted outer signer bypass the gate.

Proof of Concept

`proposal.go:178-211` :

```
for _, msg := range tx.GetMsgs() {
    if _, ok := msg.(*evmtypes.MsgEthereumTx); ok { /* check */ }
    // no handling for bank/IBC/authz/multisend
}
```

Recommendation

Replace the type-switch with a recursive scanner that extracts every Cosmos address reference (and unwraps `authz.MsgExec.Msgs`). Cover bank send/multisend recipients, IBC receivers, ICA inner msgs.

Alleviation

[Autheo , 05/05/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/635f498298a05355d8cf3596c37b6986180c6175

[CertiK , 05/05/2026]: New `checkBlockedRecipients` (`app/blocklist_recipients.go`) scans `MsgSend` , `MsgMultiSend` , `MsgTransfer` , and one-level `MsgExec` inners against the blocklist; it's invoked from `ValidateTransaction` and `BlockAddressesDecorator` —635f498298a05355d8cf3596c37b6986180c6175.

ACA-40 Emission Param `Enabled` Is Not Consistently Applied To Token Mint Pathways

Category	Severity	Location	Status
Volatile Code	Informational	app/staking_mint_inflation.go: 20; proto/emissions/v1/params.proto: 12; x/emissions/keeper/block_rewards.go: 16; x/emissions/keeper/nft_rewards.go: 34	Resolved

Description

Staking token mints bypasses emission module's param (Enabled).

Recommendation

It's recommended that the design and documentation clearly define the responsibilities of each module involved in token emissions, specifying which module controls each emission pathway. Ensure the semantic meaning and expected behavior of the Emission module's Enabled flag are unambiguous and consistently enforced across all relevant emission paths, including/excluding staking rewards. Update comments and user-facing documentation to reflect this clarified behavior, and ensure the code aligns with this documented logic so that emission gating is consistently and predictably applied.

Alleviation

[Autheo , 05/05/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/895519fd0c60af6802ab0753fee44852fed948ab

[CertiK , 05/05/2026]: `NewStakingRewardsInflationFn` (`app/staking_mint_inflation.go`) now returns zero inflation when `emissionsParams.Enabled` is false, matching the existing gates in `MintBlockRewards` / `MintNFTBlockRewards`

ACA-41 | OnRecvVouchers Conversion Failures Are Silently Swallowed Without Event Emission

Category	Severity	Location	Status
Coding Issue	Informational	x/autheo/keeper/keeper.go: 194~209; x/autheo/middleware/conversion_middleware.go: 205~226	Resolved

Description

`IBCConversionModule.convertVouchers` at `x/autheo/middleware/conversion_middleware.go:205-226` calls `autheokeeper.OnRecvVouchers(...)` and unconditionally returns `nil`. `OnRecvVouchers` at `x/autheo/keeper/keeper.go:194-209` runs `ConvertVouchersToEvmCoins` inside a `CacheContext`; on error, the cache is discarded and only `k.Logger(ctx).Error(...)` is called:

```
func (k Keeper) OnRecvVouchers(ctx sdk.Context, tokens sdk.Coins, receiver string) {
    cacheCtx, commit := ctx.CacheContext()
    err := k.ConvertVouchersToEvmCoins(cacheCtx, receiver, tokens)
    if err == nil {
        commit()
    } else {
        k.Logger(ctx).Error(fmt.Sprintf("Failed to convert vouchers to evm tokens
...""))
    }
}
```

The cache-and-rollback pattern is intentional — a failed EVM conversion must not poison the successful IBC receive. Propagating the error upward would turn the IBC ack into an error-ack and force a refund on the source chain, undoing a legitimate transfer. The user still holds the IBC voucher in their bank balance and can retry conversion via `MsgConvertVouchers`.

The observability gap: no on-chain event is emitted on the failure path. A `Logger.Error` line is node-local, not consensus-observable, not indexed, not queryable. Relayers, indexers, and users cannot detect auto-conversion failures without inspecting individual node logs. This behavior is inherited byte-identically from upstream Cronos and has not been changed there.

Recommendation

Emit a typed event on auto-conversion failure so off-chain consumers can detect stuck vouchers without scraping node logs:

```
func (k Keeper) OnRecvVouchers(ctx sdk.Context, tokens sdk.Coins, receiver string) {
    cacheCtx, commit := ctx.CacheContext()
    err := k.ConvertVouchersToEvmCoins(cacheCtx, receiver, tokens)
    if err == nil {
        commit()
        return
    }
    k.Logger(ctx).Error(...)
    ctx.EventManager().EmitEvent(types.NewVoucherConversionFailedEvent(receiver,
tokens, err))
}
```

Retain the cache-and-rollback semantics; do not propagate the error into the IBC ack.

Alleviation

[Autheo , 05/05/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/76bffbdbf2adf608a203b80af2649ff49095cacf

---

[CertiK , 05/05/2026]: `OnRecvVouchers` still rolls back failed `ConvertVouchersToEvmCoins` in a cache context but now emits `voucher_conversion_failed` (receiver, amount, error) on the parent `ctx` for indexers.

ACA-42 | Payable Relayer Precompile Can Permanently Trap EVM-Native Token

Category	Severity	Location	Status
Coding Issue	Informational	x/autheo/events/bindings/cosmos/precompile/relayer/i_relayer_functions.abigen.go: 34; x/autheo/keeper/precompiles/relayer.go: 179 ~248	Resolved

Description

Every method in the relayer precompile ABI (`i_relayer_functions.abigen.go:34`) is declared with `stateMutability: "payable"` — `createClient` , `updateClient` , `recvPacket` , `acknowledgement` , `timeout` , `registerPayee` , etc. go-ethereum's `Call` path transfers the caller's value to the precompile address (`0x0000000000000000000000000000000000000065`) before `Run` executes (`core/vm/evm.go:233`). Ethermint's `StateDB.AddBalance` delegates to the bank module, so the precompile address genuinely holds balance in a bank account keyed by its 20-byte address.

`RelayerContract.Run` at `x/autheo/keeper/precompiles/relayer.go:179-248` never inspects `contract.Value()` and never rejects nonzero value. No sweep, withdraw, refund, or admin-callable recovery path exists for `0x65` . Any wei sent with a successful call is permanently stranded at the precompile address.

Impact is self-inflicted only. There is no attacker primitive that forces a third party's ETH (EVM-native token) into the precompile — EVM `CALL` -with-value debits the caller's balance, not the victim's. The Solidity compiler also refuses to compile a non-payable call with value. Realistic triggers are therefore limited to: (a) a user pasting the precompile address into a wallet and attaching value, or (b) a buggy wrapper contract that indiscriminately forwards `msg.value` .

This behavior is inherited byte-identically from upstream Cronos and has not been changed there. The same pattern exists in the sibling precompiles (`bank.go` , `ica.go`), which are equally payable and equally unguarded.

Recommendation

Add an explicit non-payable guard at the top of `Run` in each precompile that does not consume value:

```
if contract.Value().Sign() != 0 {
    return nil, errors.New("precompile is non-payable")
}
```

Returning an error here reverts the whole frame, which unwinds the value transfer (the pre-Run `Transfer` is inside the same snapshot), so the caller is not charged.

Alternatively, drop `"stateMutability": "payable"` from the Solidity interfaces and regenerate the ABI bindings, so the Solidity compiler refuses at compile time to attach value to these calls.

Apply the same fix to sibling precompiles (`x/autheo/keeper/precompiles/bank.go` , `x/autheo/keeper/precompiles/ica.go`) — they share the same defect class.

Alleviation

[Autheo , 05/05/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/41e642a349da02229d3b0e64105b42b6fa0e6a9e

---

[CertiK , 05/05/2026]: `RelayerContract.Run` (and sibling `bank` / `ica` precompiles) now rejects nonzero `contract.Value()` with an error so the EVM reverts and the pre-Run transfer unwinds—41e642a349da02229d3b0e64105b42b6fa0e6a9e.

ACA-43 Emissions GetEmittedTotal Treats Unparsable EmittedTotal Bytes As Zero

Category	Severity	Location	Status
Coding Issue	Informational	x/emissions/keeper/state.go: 36~48	Resolved

Description

`Keeper.GetEmittedTotal` reads `types.EmittedTotalKey` and parses the value with `sdkmath.NewIntFromString`. On parse failure it returns `sdkmath.ZeroInt()`, conflating corrupted non-numeric state with a legitimate zero total. Downstream cap and emissions accounting that depends on the cumulative total can then behave as if the counter reset. No unprivileged path to write malformed bytes to this key is established in the finding; impact is state-decoding and operational robustness.

Recommendation

On `NewIntFromString` failure, return an error from a refactored API, or panic/halt the BeginBlocker path, or at minimum emit a high-severity log and keep a last-known-good cache. Distinguish absent key, valid zero, and parse failure. Validate `EmittedTotal` during genesis export/import and upgrades.

Alleviation

[Autheo , 05/05/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/25c2174118c38e9844699de79e4ac91ae8294c36

[CertiK , 05/05/2026]: `GetEmittedTotal` (and other global Int/Dec store reads in `state.go`) now panics on parse failure instead of returning zero, so corrupted bytes cannot masquerade as a reset counter—25c2174118c38e9844699de79e4ac91ae8294c36.

ACA-44 ERC-721 Transfer On License Contracts Reverts Entire Tx When Token ID Exceeds Uint64

Category	Severity	Location	Status
Coding Issue	Informational	x/autheo/keeper/evm_hooks.go: 38~41; x/license/keeper/evm_hooks.go: 87~91; x/license/types/params.go: 22~40	Acknowledged

Description

`x/license/keeper/evm_hooks.go:87-91` extracts the ERC-721 token ID from `topics[3]` and rejects values that do not fit in `uint64`:

```
tokenIdBig := topics[3].Big()
if !tokenIdBig.IsUint64() {
    return fmt.Errorf("tokenId too large: %s", tokenIdBig.String())
}
```

Because the hook runs inside `PostTxProcessing` (`x/autheo/keeper/evm_hooks.go:38-41`), returning an error reverts the entire EVM transaction — not just the license-accounting side. The hook only fires when the emitting contract matches one of the three governance-set tier addresses (`SovereignLicenseContract`, `PrimeLicenseContract`, `CoreLicenseContract`). The license module stores IDs as `uint64` throughout, so widening the decoder alone would not help.

Operational footgun: if governance ever points a tier-contract param at an ERC-721 that mints IDs ≥ 2^64 (e.g., hash-derived or timestamp-based IDs common in NFT projects), every on-chain transfer of that collection reverts. Users lose gas, marketplaces break, and secondary trading of the NFT collection is bricked until governance resets the param. No unprivileged attacker path exists.

Not a live defect — current tier contracts are operator-controlled and use small sequential IDs. Filed as an operator-configuration hazard.

Recommendation

Log the oversized-ID event and return nil instead of erroring, so an unsupported ERC-721 does not brick transfers of an unrelated user's NFT:

```
if !tokenIdBig.IsUint64() {
    k.Logger(ctx).Warn("license tokenId exceeds uint64, ignoring transfer",
"contract", contractHex, "tokenId", tokenIdBig.String())
    return nil
}
```

Alternatively, widen internal license IDs to `*big.Int` / `sdk.Int` if arbitrary NFT ID spaces must be supported. Document in operator guidance that tier contracts must use ID schemes bounded by uint64.

Alleviation

[Autheo, 05/05/2026]: By business logic, the data type is sufficient to handle token IDs.

[CertiK, 05/06/2026]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement, because the business logic will ensure this does not happen.

ACA-45 Revoked Prime/Core Delegators Keep Receiving Standard Delegator Rewards When Unbonding Queue Is Full

Category	Severity	Location	Status
Coding Issue	Informational	x/license/keeper/msg_server.go: 486~508, 879~946	Acknowledged

Description

`RevokeLicense` at `x/license/keeper/msg_server.go:486-508` invokes `autoUndelegateForRevoke` as best-effort. When the owner's unbonding queue has already filled its SDK default `MaxEntries=7` slots, `autoUndelegateForRevoke` (`msg_server.go:908-919`) returns `ErrUnbondingQueueFull` without touching the delegation. The caller logs, emits `license_revoke_owner_undelegate_failed`, and **proceeds with the revocation** regardless (`msg_server.go:685`: `license.Status = REVOKED`; line 680: binding deleted).

The source comment at lines 487-490 is explicit: *"Governance must not be blocked by the revoked license owner having a full unbonding queue."*

Effect by license category

Sovereign (owner = validator). `RevokeLicense` jails the validator at line 651. A jailed validator earns nothing; the owner's bonded stake earns zero native rewards from that block forward. License NFT emissions stop (REVOKED filter in `x/emissions/keeper/nft_rewards.go:140`). No residual reward window.

Prime/Core (owner = delegator of some Sovereign-backed validator V). Revocation does NOT jail V — V's own Sovereign license is untouched. The revoked delegator's stake stays bonded to V, and `x/distribution` keeps paying the normal delegator share each block because V is healthy. Only license NFT emissions stop. The delegator can then gradually undelegate manually as queue slots free (each unbonding entry matures in the ~21-day `UnbondingTime`), collecting normal delegator rewards in the meantime. Worst-case window: one full `UnbondingTime` (~21 days).

Impact bound

Prime/Core residual earnings ≈ bonded principal × native staking APR × (residual days / 365). At typical 10-20% APR, ≤ 21 days → roughly 0.6-1.2% of principal earned while the license is already marked REVOKED. Redelegation is globally disabled in the licensedstaking wrapper (`x/licensedstaking/keeper/msg_server.go:436-441`), so the window cannot be extended beyond one unbonding cycle. The governance voting period is publicly visible, so a prepared Prime/Core holder can pre-fill 7 one-share unbonding entries for negligible gas before the revoke tx executes.

Recommendation

The current behaviour is the documented intent. If the product later decides the Prime/Core residual-rewards window is unacceptable, two options are available:

1. **Redirect** — intercept the revoked delegator's `x/distribution` outflows to a module-held escrow (or the community pool) until their stake has fully exited V. The delegator keeps principal-security of the unbonding period but earns nothing

while REVOKED.

2. **Force exit** — after emitting the `license_revoke_owner_undelegate_failed` event, schedule a keeper-driven retry each `EndBlocker`: as soon as one of the 7 queue slots matures, auto-undelegate the remaining shares. The total residual window shrinks from ~21 days to the time it takes one slot to free. Also, license revoke triggered undelegations could ignore the unbonding max entries to bypass the limit.

Otherwise, document in operator guidance that Prime/Core revocation halts NFT emissions immediately but may leave native delegator rewards flowing to the revoked delegator for up to ~21 days when they have pre-filled their unbonding queue.

Alleviation

[Autheo, 05/05/2026]: It can be controlled through UnbondingTime or MaxEntries.

[Autheo, 05/06/2026]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement, since this issue can be mitigated by adjusting unbondingtime and maxentries.

ACA-46 Dormant BeginRedelegate License-Rebind Path Has Latent Defects If Re-Enabled

Category	Severity	Location	Status
Coding Issue	Informational	x/license/keeper/msg_server.go: 1058~1114; x/licensedstaking/keeper/msg_server.go: 436~547	Acknowledged

Description

`BeginRedelegate` at `x/licensedstaking/keeper/msg_server.go:436-441` is globally disabled whenever `licenseKeeper != nil` and returns `ErrRedelegateDisabled`. A parked replacement, `beginRedelegateLicenseSync` at lines 449-547, is marked `//nolint:unused,deadcode` and is reachable only by removing the kill-switch.

If the dormant path is re-enabled without changes, three defects would be activated:

1. **Src-jailed bypass.** `srcWasJailedBefore` is snapshotted (line 464) and used only to filter out *new* implicit-jail signals after the redelegate call. When src is **already** jailed at the time of the redelegate, `RebindLicense` (line 500) still moves licenses to dst and sets them `ACTIVE`, and the `!srcWasJailedBefore` guard at line 505 skips the ACTIVE→BOUND downgrade branch. Licenses inherited from an already-jailed source become active on destination without re-verification.

2. **Dst jail/tombstone not checked.** `RebindLicense` at `x/license/keeper/msg_server.go:1058-1114` performs no validation on `dstVal` — it does not verify the destination validator exists, is not jailed, and is not tombstoned. Currently the upstream SDK `BeginRedelegate` rejects unknown dst before `RebindLicense` runs, but a jailed-but-valid dst still receives licenses that are unconditionally set `ACTIVE`.

3. **Sovereign escape.** No category filter. The doc comment on the disabled wrapper (line 433-435) explicitly flags the risk: *"a sovereign operator could incorrectly rebind their license to a different validator."* The dormant handler moves any category including Sovereign, defeating the `checkSovereignLicenseForUnjail` gate.

No current runtime exposure — the kill-switch is active. Filed here as a future-work hazard: re-enabling the dormant path (e.g., to support a product requirement for redelegation) must first fix all three defects, or replace the path entirely with a sequence of Undelegate + Delegate calls that route through the existing license-sync hooks (`OnDelegationRemovedOrZero`, `OnDelegationCreatedOrIncreased`, jail interceptor).

Recommendation

Before re-enabling `BeginRedelegate`, either:

Option A — replace with an Undelegate + Delegate composition. Let the existing hooks handle the license state machine. The sibling `Undelegate` / `Delegate` paths already correctly demote ACTIVE → BOUND on exit and re-promote on new delegation, with jail-aware guards and sovereign-specific cascades.

Option B — harden the dormant handler if direct redelegation is required:

1. Reject redelegation when src is jailed or tombstoned.
2. Reject redelegation for Sovereign licenses (or force an Unbind + Rebind cycle with cooldown).
3. Re-check dst in `RebindLicense`: if `dstVal.IsJailed()` is true, set the rebound license status to `BOUND` instead of `ACTIVE`.
4. Remove the `!srcWasJailedBefore` gate on the downgrade branch — the post-rebind state should depend only on dst's current jail state.
5. Add an integration test covering jailed-src and jailed-dst redelegation scenarios.

Until one of these is done, keep the kill-switch in place and remove the dead code to avoid accidental re-enable.

Alleviation

[Autheo, 05/05/2026]: BeginRedelegate is currently a no-op. We'll follow the recommendation once it's enabled.

[CertiK, 05/06/2026]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement, since BeginRedelegate is currently a no-op.

ACA-47 | Licensedstaking ValidateGenesis Drops Upstream SDK Validator Invariants

Category	Severity	Location	Status
Coding Issue	Informational	x/licensedistribution/module.go: 49~52; x/licensedstaking/module.go: 60~68	Resolved

Description

`licensedstaking/module.go:60-68` overrides `ValidateGenesis` to call only `data.Params.Validate()`, dropping the SDK's `validateGenesisStateValidators` checks: (1) no duplicate consensus keys, (2) no jailed-and-bonded, (3) no zero `DelegatorShares`. `staking.InitGenesis` does not re-check these — corrupt genesis boots silently. `SetValidatorByConsAddr` blind-overwrites duplicate consensus keys; jailed-bonded validators occupy bonded-pool liquidity without consensus participation; zero-shares bonded validators panic on first `TokensFromShares` call.

Proof of Concept

`licensedstaking/module.go:62-67`:

```
return data.Params.Validate()   // upstream calls
stakingtypes.ValidateGenesis(&data)
```

Recommendation

`return stakingtypes.ValidateGenesis(&data)`.

Alleviation

[Autheo , 05/05/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/14e7e50cf7069988200bbbc4f09c71f381f9be71

[CertiK , 05/05/2026]: `licensedstaking`'s `ValidateGenesis` now delegates to `staking.ValidateGenesis(&data)` instead of params-only validation, restoring duplicate-consensus-key, jailed/bonded, and zero-shares checks—14e7e50cf7069988200bbbc4f09c71f381f9be71.

ACA-48 Keeper Token Mapping Trusts AutheoAdmin Without On-Chain Contract-Semantics Validation

Category	Severity	Location	Status
Coding Issue	Informational	x/autheo/keeper/evmhandlers/send_to_ibc.go: 94~115; x/autheo/keeper/keeper.go: 129~146; x/autheo/keeper/msg_server.go: 84~120; x/autheo/keeper/permissions.go: 32~60	Resolved

Description

`SetExternalContractForDenom` at `x/autheo/keeper/keeper.go:129-146` persists a `denom ↔ contract` association without validating that the contract address actually hosts ERC-20 code or matches the declared decimals / symbol. `SendToIbcHandler.handle` at `x/autheo/keeper/evmhandlers/send_to_ibc.go:94-115` then trusts the mapping when deciding which denom to mint or release in response to an EVM event, so a malformed or malicious mapping directly controls bridged-asset accounting.

Write paths are governance-gated:

1. `MsgUpdateTokenMapping` via `msg_server.go:84-97`, gated by `HasPermission(..., CanChangeTokenMapping)`.
2. `TokenMappingChangeProposal` governance handler.

`HasPermission` (`x/autheo/keeper/permissions.go:32-60`) accepts the `AutheoAdmin` param address OR any address the admin has granted `CanChangeTokenMapping` via `SetPermissions`. `UpdatePermissions` is restricted to `AutheoAdmin` (`msg_server.go:105-120`). `AutheoAdmin` itself is updated via `MsgUpdateParams` (gov authority).

Net trust model: governance is the root, but operationally a single `AutheoAdmin` key and any delegates it grants can re-point any ERC-20 contract to any denom, producing arbitrary mint/release. This is a known operational trust boundary rather than a defect, and is covered in greater depth by two existing findings:

- `findings/findings/MEDIUM-22-token-mapping-proposal-validatebasic-nil.json` — missing input validation on the governance proposal path.
- `findings/findings/MEDIUM-03-missing-contract-code-validation.json` — missing keeper-level `ensureContractCode()` check.

Filed here as an informational cross-reference so reviewers reading the aiflow set see the connection.

Recommendation

Apply the combined remediation from the existing findings:

1. Harden `TokenMappingChangeProposal.ValidateBasic` (per MEDIUM-22).
2. Add an `ensureContractCode` check in `SetExternalContractForDenom` (per MEDIUM-03).

Separately, consider multi-signature or timelock on `AutheoAdmin` key operations, or replace the single-admin permission system with pure governance for token-mapping changes.

Alleviation

[Autheo , 05/05/2026]: ACA-48 is fixed by the ACA-18 implementation:

ensureContractCode blocks EOAs/undeployed addresses in both admin and governance write paths and TokenMappingChangeProposal.ValidateBasic now validates denom and hex address instead of returning nil.

[CertiK , 05/05/2026]: Cross-reference remediated in 168f11abf254a46a8c4412158f480a1a037a72e8: `ensureContractCode` runs before `SetExternalContractForDenom`; `TokenMappingChangeProposal.ValidateBasic` validates denom and contract hex.

ACA-49 E2ee InitGenesis Skips GenesisState.Validate; Malformed X25519 Keys Stored Verbatim

Category	Severity	Location	Status
Coding Issue	Informational	x/e2ee/keeper/keeper.go: 30~70	Resolved

Description

`InitGenesis` calls `registerEncryptionKey` directly without `state.Validate()`. `registerEncryptionKey` validates the bech32 address only, not the X25519 key bytes. Programmatic genesis builds that bypass `AppModuleBasic.ValidateGenesis` store malformed keys verbatim; later `age.ParseX25519Recipient` calls fail at runtime.

Proof of Concept

`keeper.go:55-65`:

```
for _, key := range state.Keys {
    k.registerEncryptionKey(ctx, key.Address, key.Key)   // no state.Validate; no key parse
}
```

Recommendation

Call `state.Validate()` at the top of `InitGenesis`. Add `age.ParseX25519Recipient` to `registerEncryptionKey`.

Alleviation

[Autheo , 05/05/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/302897b3bd12bfab9500eaa65ba942dd26146ca0

[CertiK , 05/05/2026]: `InitGenesis` now calls `state.Validate()` before writing keys, and `registerEncryptionKey` rejects non-parseable material via `age.ParseX25519Recipient` before persisting—302897b3bd12bfab9500eaa65ba942dd26146ca0.

ACA-50 | Mempool TxReplacement Int64 Overflow Allows Fee-Bump Bypass

Category	Severity	Location	Status
Coding Issue	Informational	app/app.go: 419~423	Resolved

Description

`TxReplacement` computes `op*threshold/100` on int64 with no overflow check. For `op > MaxInt64/threshold`, the multiplication wraps negative and the comparison succeeds for any `np`, defeating the fee-bump policy. Reachable only when operator sets `feeBump > 0`.

Proof of Concept

`app.go:419-423`:

```
threshold := 100 + feeBump
return np >= op*threshold/100
```

`op = 1<<60`, `feeBump = 20` → `op*120` wraps under int64 → bound negative → any `np` accepted.

Recommendation

Use `math/big`:

```
lhs := new(big.Int).Mul(big.NewInt(np), big.NewInt(100))
rhs := new(big.Int).Mul(big.NewInt(op), big.NewInt(100+feeBump))
return lhs.Cmp(rhs) >= 0
```

Alleviation

[Autheo , 05/05/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/785cf63961bd5d91c3866433265fc5f76946b98b

[CertiK , 05/05/2026]: `TxReplacement` now uses `math/big` for the fee-bump ratio in `app/app.go`, fixing the `int64` overflow bypass—785cf63961bd5d91c3866433265fc5f76946b98b.

ACA-51 | E2ee Keygen Silently Overwrites Existing Identity

Category	Severity	Location	Status
Coding Issue	Informational	x/e2ee/client/cli/generate.go: 30~55	Resolved

Description

`KeygenCommand` calls `kr.Set(krName, ...)` without a pre-existence check. Re-running destroys the prior key.

Proof of Concept

`generate.go:38-43`:

```
k, _ := age.GenerateX25519Identity()
kr.Set(krName, []byte(k.String()))   // no kr.Get pre-check
```

Recommendation

```
if _, err := kr.Get(krName); err == nil && !overwriteFlag {
    return fmt.Errorf("identity %q exists; use --overwrite", krName)
}
```

Alleviation

[Autheo , 05/05/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/autheo-blockchain/autheo-chain-core/commit/774cf835e7559763ba8430c2d3fd18dab6b1466b

[CertiK , 05/05/2026]: The team heeded the advice and resolved the issue by adding a `kr.Get(krName)` pre-check in `KeygenCommand` (`x/e2ee/client/cli/generate.go`) before generating and storing a new age identity: if a record already exists and `--overwrite` is not set, the command returns a clear error instead of calling `kr.Set` blindly, and a boolean `--overwrite` flag allows intentional replacement, in commit 774cf835e7559763ba8430c2d3fd18dab6b1466b.

APPENDIX | AUTHEO - CHAIN AUDIT

Audit Scope

autheo-blockchain/autheo-chain-core
app/app.go
x/license/module.go
x/license/keeper/evm_hooks.go
x/license/keeper/keeper.go
x/license/keeper/msg_server.go
x/license/types/params.go
x/licensedstaking/keeper/msg_server.go
app/block_address.go
app/proposal.go
app/staking_mint_inflation.go
x/autheo/events/decoders.go
x/autheo/events/events.go
x/autheo/keeper/keeper.go
x/autheo/keeper/msg_server.go
x/autheo/keeper/precompiles/ica.go
x/autheo/keeper/precompiles/relayer.go
x/autheo/middleware/conversion_middleware.go
x/emissions/abci.go
x/emissions/keeper/msg_server.go

autheo-blockchain/autheo-chain-core
x/emissions/keeper/nft_rewards.go
x/emissions/keeper/state.go
x/emissions/keeper/block_rewards.go
x/emissions/types/genesis.go
x/emissions/types/keys.go
x/emissions/types/nft_emissions_params.go
x/emissions/types/params.go
x/license/keeper/genesis.go
x/license/types/errors.go
x/license/types/keys.go
x/licensedistribution/module.go
x/licensedstaking/module.go
x/licensedstaking/keeper/jail_interceptor.go
x/autheo/keeper/precompiles/utils.go
x/autheo/types/errors.go
x/autheo/types/params.go
x/emissions/module.go
x/emissions/keeper/keeper.go
x/emissions/keeper/grpc_query.go
x/emissions/types/codec.go
x/emissions/types/errors.go
x/emissions/types/expected_keepers.go

autheo-blockchain/autheo-chain-core
x/emissions/types/tx.go
x/license/client/cli/query.go
x/license/client/cli/tx.go
x/license/keeper/nft_stub.go
x/license/keeper/query_server.go
x/license/keeper/staking_adapter.go
x/license/types/codec.go
x/license/types/expected_keepers.go
x/license/types/inbound.go
x/license/types/msg.go
x/licensedistribution/keeper/keeper.go
x/licensedistribution/keeper/msg_server.go
x/licensedistribution/types/codec.go
x/licensedistribution/types/errors.go
x/licensedistribution/types/expected_keepers.go
x/licensedistribution/types/keys.go
x/licensedistribution/types/tx.go
x/licensedstaking/expected_keeper.go
x/licensedstaking/keeper/keeper.go

Finding Categories

Categories	Description
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Access Control	Access Control findings are about security vulnerabilities that make protected assets unsafe.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Denial of Service	Denial of Service findings indicate that an attacker may prevent the program from operating correctly or responding to legitimate requests.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevate Your Web3 Journey

CertiK is the largest Web3 security platform combining formal verification with audits and comprehensive security solutions.