Blaize.Security

E Money
NETWORK

EMONEY NETWORK
BLOCKCHAIN SECURITY AUDIT

June 7th, 2024 / V. 1.0

Table of Contents

Executive Summary

During the audit, we examined the security of the **EMoney** chain (EMC chain) - Cosmos / Evmos based chain. Our task was to find and describe any security issues in the protocol. This report presents the findings of the security audit of the **EMoney** chain conducted between **February 2nd, 2024 and March 11th, 2024.** The second iteration of the audit connected to the review of the protocol upgrade, Evmos upgrade and issues patching occurred between **April 29th, 2024 and June 6th, 2024.**

Blaize.Security team audited the EMC Chain - Cosmos-based chain with EVM support (via the Evmos module). The audit's goal was to review the implementation of Lock and Regulation modules, check the integrity of the network codebase, and review it against inconsistencies, security issues, vulnerabilities, or business logic flaws. The security team worked simultaneously in three main directions: system analysis, review of each module and comparison with the available specification, line-by-line code review, and simulation testing of the protocol on the local network - network launched in the private mode with few validators simulated locally. Such a local environment was used for several manual and exploratory testing rounds.

The audit team examined the system architecture, ensuring the correct relationship between protocol modules, their interfaces, type relations, and used data structures. The team reviewed the upgrades implemented over the native Cosmos and Evmos modules and their influence on standard transactions (both Cosmos and EVM). The security team reviewed the available configuration and settings and the initial setup of the protocol.

During the testing stage, the team reviewed available tests, conducted local testing of modules (especially focusing on the access control and correct implementation of restrictions), and performed a simulation of the internal network with several rounds of testing, simulating different genesis conditions, different transaction orders (both Cosmos and EVM).

The security team discovered several issues:

- several violations of the genesis record validations
- issues with the storage export
- issues with the access control flow (in terms of authority delegation and new validators onboarding)
- outdated Evmos used
- several best practices and code quality violations (like todos, unused code, etc.)
- several missing queries and edge cases (referring to the implemented business logic)

The EMC team verified or resolved most of the issues. Also, the security team provided patching as well - resolving several issues connected to the genesis state validation and Evmos upgrade. However, auditors should note that some issues marked as Info are not resolved - and they are marked as such connected to the business logic. So it is highly recommended that the team pay attention to them during further development. However, it should be noted that the chain's current state is alpha-testnet (upgraded to the beta-testnet after the audit). Therefore, some security measures are less strict, but they still require attention in the future. Also, it should be noted that the set of native tests requires improvements, and the code contains several TODOs (some of which are inherited from the Evmos modules). Thus, further codebase improvements are recommended.

The EMC chain successfully passed the security audit, and the security team noted the high level of cooperation from the EMC side and the willingness to make the necessary codebase improvements. Thus, several patches were provided by the security team and accepted by the EMC team, including recommendations on the chain upgrade procedure. However, the security team recommends adding more documentation, especially natspec comments, and improving the state of native tests.

From all points of view, the protocol shows high compliance with the security standards. **Note**: the result is applicable for the codebase itself, testnet will share the same level of security only after the upgrade.

Auditing Strategy and Techniques Applied/Procedure

Blaize.Security auditors start the audit by developing an auditing strategy - an individual plan where the team plans methods, techniques, approaches for the audited components. That includes a list of activities:

MANUAL AUDIT STAGE

- Manual line-by-line code by at least 2 security auditors with crosschecks and validation from the security lead;
- Protocol decomposition and components analysis with building an interaction scheme, depicting internal flows between the components and sequence diagrams;
- System analysis for components/modules co-dependencies;
- Business logic inspection for potential loopholes, deadlocks, backdoors;
- Order-dependency and time-dependency of operations
- Math operations and calculations analysis, formula modeling;
- Review of dependencies, 3rd parties, and integrations;
- Storage structure and stored data structure review (including read/write operations);
- Review global settings for the usage/mis-using within the protocol (check for necessity, ambiguity and mis-using) and the default values;
- Basic validations of parameters of functions/methods/messages (or their analogs)
- Vulnerabilities analysis against several checklists, including internal Blaize.Security checklist;
- Code quality, documentation, and consistency review.

FOR ADVANCED COMPONENTS:

- Cryptographical elements and keys storage/usage audit (if applicable);
- Review against OWASP recommendations (if applicable);
- Blockchain interacting components and transactions flow (if applicable);
- Review against CCSSA (C4) checklist and recommendations (if applicable);

TESTING STAGE:

- Development of edge cases based on manual stage results for false positives validation;
- Integration tests for checking connections with 3rd parties;
- Manual exploratory/simulation tests over the locally deployed protocol;
- Checking the existing set of tests and performing additional unit testing;
- End-to-end testing of complex systems;

In case of any issues found during audit activities, the team provides detailed recommendations for all findings.

Audit Rating

Score: 9.3 /10

	RATING
Security	**9.8**
Logic optimization	9.9
Code quality	9.5
Testing suite	5.0
Documentation	8.0

Security: General mark for the security of the protocol.
The main mark for the audit qualification.

Logic optimization: Evaluation of how optimal the implementation is, including presence of extra/unused code, uncovered/extra cases, gas (or its analog) optimization, memory management optimization, etc
Code quality: Evaluation of best practices followed, code readability, structure and convenience of further development
Testing suite: Availability of the native tests suite, level of logic coverage, checks of critical areas being covered.
Documentation: Availability and quality of the documentation, coverage of core functionality and user flows: whitepaper, gitbook, readme, specs, natspec, comments in the code and other possible forms of documentation.

SECURITY RATING CALCULATION

Approximate weight of unresolved issues.
Critical: -3 points
High: -2 points
Medium: -1 points
Low: -0.25 points
Informational: -0.1 point

Note: additional concerns, violated checklist items (including standard vulnerabilities), and verified backdoors may influence the final mark and weight of certain issues.

Starting with a perfect score of 10:

Critical issues: 2 issues (2 resolved): 0 points deducted

High issues: 2 issues (2 resolved): 0 points deducted.

Medium issues: 4 issues (3 resolved, 1 verified): 0 points deducted

Low issues: 0 issue (0 resoled): 0 points deducted

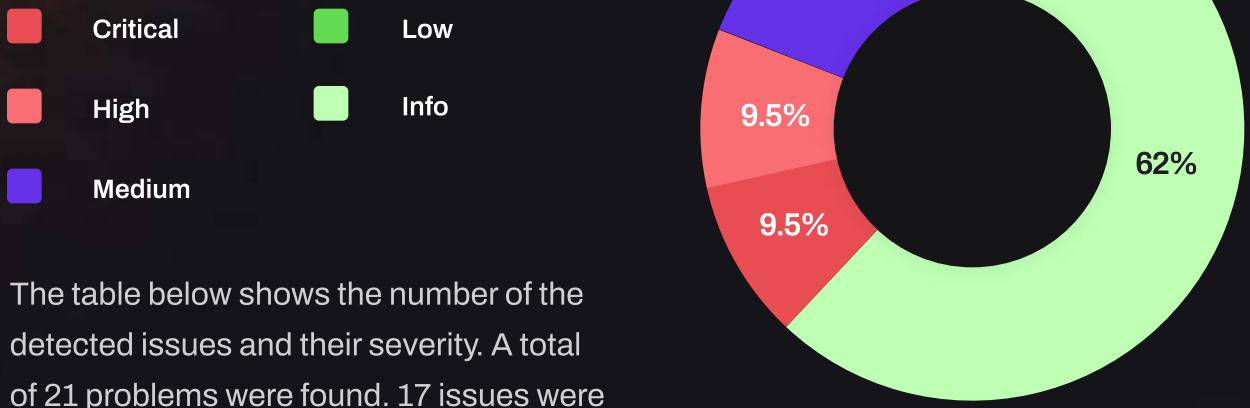
Informational issues: 13 issues (6 resolved, 3 verified): -0.5 points deducted for unresolved issues

Other: 0.2 points deducted for low amount of native tests and coverage with documentation

Security rating = 10 -0.5 - 0.2 = 9.3

Technical Summary

THE GRAPH OF VULNERABILITIES DISTRIBUTION:



The table below shows the number of the detected issues and their severity. A total of 21 problems were found. 17 issues were fixed or verified by the Customer's team.

	FOUND	FIXED/VERIFIED
Critical	2	2
High	2	2
Medium	4	4
Low	0	0
Info	13	9

SEVERITY DEFINITION

CRITICAL

The system contains several issues ranked as very seriousand dangerous for users and the secure work of the system. Requires immediate fixes and a further check.

HIGH

The system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge data or financial leak. Requires immediate fixes and a further check.

MEDIUM

The system contains issues that may lead to medium financial loss or users' private information leak. Requires immediate fixes and a further check.

LOW

The system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Requires fixes.

INFO

The issue has no impact on the contract's ability to operate, yet is relevant for best practices. Or this status can be assigned to the issues related to the suspicious activity or substandard business logic decisions which cannot be classified without the comments from the team (and can be re-classified on the later audit stages).

Issues reviewed by the team can get the next statuses:
Resolved: issue is resolved by an appropriate patch or changes in the business logic
Verified: the team provided sufficient evidences that the issue describes desired behavior
Unresolved: neither path nor comments provided by the team, or they are not sufficient to resolve the issue
Acknowledged: the team accepts the misbehavior and connected risks

Audit Scope

Language/Technology: **Golang**

Blockchain: **EMoney Chain (Cosmos / evmos)**

The scope of the project includes:

- x\lock\client*.go
- x\lock\keeper*.go
- x\lock\types*.go
- x\regulation\client*.go
- x\regulation\keeper*.go
- x\regulation\types*.go

The audited codebase (before the chain renaming) is based on the original EMoney repository: https://github.com/ScallopBank/chain

- ccc188c1306bc3c2c3020e7dffb13905c23d2298

During the audit the codebase was transferred into the re-branded repository: https://github.com/EMoney-Network/EMoneyChain

The source code is located in the branch: **main**

Final commit:

- d13ff46b50bb1454c0f764e9772c9226f16c790b

Final patch from the security team is located in branch: **evmos-update**

- 4f7bd3a1e0e64f9ecaca9ce0eb1b3e12c7d8f81d

Additional patch from the security is located in branch: **evmos-update**

- 996b643ce9f6dc27ba57e8a0a3bfd770fb33fcbc

Protocol overview

EMC is an EVM blockchain network built using Cosmos-SDK and Evmos. It implements two core features:

- KYC Regulation.
- Lock of tokens.

1. KYC Regulation

By default, EMC doesn't allow users who haven't passed KYC to send transactions. This functionality is implemented in the x/regulation directory. The flow of the regulation is implemented as follows:

- Specify regulation authorities in the genesis.
- Authorities are allowed to transact by default.
- Authority can allow or disallow users to transact.
- If allowed, the user can send transactions to the chain.
- An authority can renounce its regulation rights and transfer them to another user, who then becomes a regulation authority.

2. Lock of tokens

EMC introduces an extension of the existing Staking module of Cosmos-SDK. The extension is that before validators can stake tokens to the StakingKeeper (the basic Cosmos SDK Keeper that allows users to become network validators), they must become vested validators sponsored by other users.

The flow of the lock of tokens is implemented as follows:

- Specify the initial vested validators and locks in the genesis.
- Specify lock parameters such as Denom, amount, and duration.
- Any user who has enough EMC coins is able to lock them in order to register any other user as a vested validator.
- A validator, registered as vested, can stake coins in the StakingKeeper module and become a network validator.
- After the lock duration period, the user is able to withdraw locked coins.
- Users can then lock coins for a new validator.
- The validator, marked as vested, can stake/unstake coins at StakingKeeper anytime.

ROLES AND RESPONSIBILITIES

Lock Module

1) Vested Validator

A vested Validator is a user for whom another user vests coins. When someone initiates a lock tokens transaction, an address is specified, which will become a vested validator.

Validators are set through the function: x/lock/keeper/keeper.go:SetVestedValidator using the type `VestedValidatorsKey` as key (from x/lock/types/keys.go: VestedValidatorsKey).

When validators are created:

- Vested validators are specified during the genesis block and are set in x/lock/keeper.genesis.go:InitGenesis.
- When someone sends a lock tokens transaction in x/lock/keeper/keeper.go:Vest (Lock module) x/lock/keeper/msg_server.go: LockTokens.

Note! Once a user is registered as a vested validator, he can't be unregistered.

Responsibilities.

- Stake coins to the StakeKeeper module to become validator of the network.
- Behave as a conscientious validator of the network.

2) Regular Users

Regular users can vest coins to add new validators. The amount of coins they have to vest is set during genesis in x/lock/keeper/genesis.go: InitGenesis. When a lock tokens transaction is sent, a lock of coins is created for the user. Coins are sent from the user to the module. One user can have only one lock. After a lock period, the user can withdraw coins and lock them again for a new validator.

Note! The user must complete KYC to send transactions. The user can have only one lock simultaneously. After a lock is withdrawn, the user can create a new lock for a new vested validator.

Responsibilities.

- Have a specified amount of coins.
- Withdraw coins after a lock period.

ROLES AND RESPONSIBILITIES

Regulation module

1) Authority

Authorities are a group of users responsible for allowing/disallowing users to transact into EMC. Authorities are set during the genesis and can't be added later. However, an authority can renounce its role in favor of a new authority.

When authorities are added:

- During the genesis, a list of initial authorities is specified in x/regulatory/keeper/genesis.go: InitGenesis.
- Authority can be transferred through a request from authority to another user in x/regulatory/keeper/msg_serve.go: TransferAuthority().

Note! By default, authorities are allowed to transact to the EMC.

Responsibilities.

- Allow users who passed KYC to transact to the EMC chain. x/regulatory/keeper/msg_server.go: AllowAddress
- Disallow users from transacting to the EMC chain. x/regulatory/keeper/msg_server.go: DisallowAddress

2) Unspecified users

Unspecified users are accounts that aren't allowed to transact yet.

Responsibilities.

- Complete KYC in order to be allowed to transact into the EMC.

3) Allowed users.

Allowed users are users who have passed KYC and are allowed by authorities to transact into EMC. They can send Cosmos, EVM, and EMC transactions.

CONFIGURATION AND SETTINGS

Lock module

1. LockAmount

LockAmount is the amount of EMC coins that must be locked to register a user as a vested validator.
LockAmount has the type Coins. During the genesis, it is set as follows:

- Denom: EMC
- Amount: 10000

LockAmount is immutable and can only be set during genesis.

2. LockDuration

LockDuration is the duration specified in seconds, indicating the time tokens are locked.
LockDuration has the type time. Duration. It is set as follows:

- Duration: 604800 seconds

LockDuration is immutable and can only be set during genesis.

Initial staking parameters

According to the initial parameters of the network, the validator set contains up to 100 validators, and the unbinding time is 1814400 seconds.

DEPENDENCIES

E-Money Chain is implemented using Evmos 8.2.0 and Cosmos SDK 0.45.8, which runs on top of the CometBFT consensus engine.

Such integrations allow both Cosmos and Ethereum transactions to be executed on the network, including deployment and interaction with Solidity contracts. Cosmos transactions can be executed using CLIs, gRPCs (including CosmJS and grpcurl), and REST endpoints, while Ethereum transactions can use a familiar set of tools, including Ethers and Web3.

Ethereum blockchain protocol version (hard fork) is the Merge NetSplit (Paris upgrade) (Sep 2022). That is, using the latest versions of Solidity is available by setting the target EVM version of the `solc` compiler to the "Paris" version. However, the opcodes added after `0.8.19` are unavailable.

Note: After the patching prepared by the security team, the evmos version was upgraded up to the 12.1.6 version.

LIST OF VALUABLE ASSETS

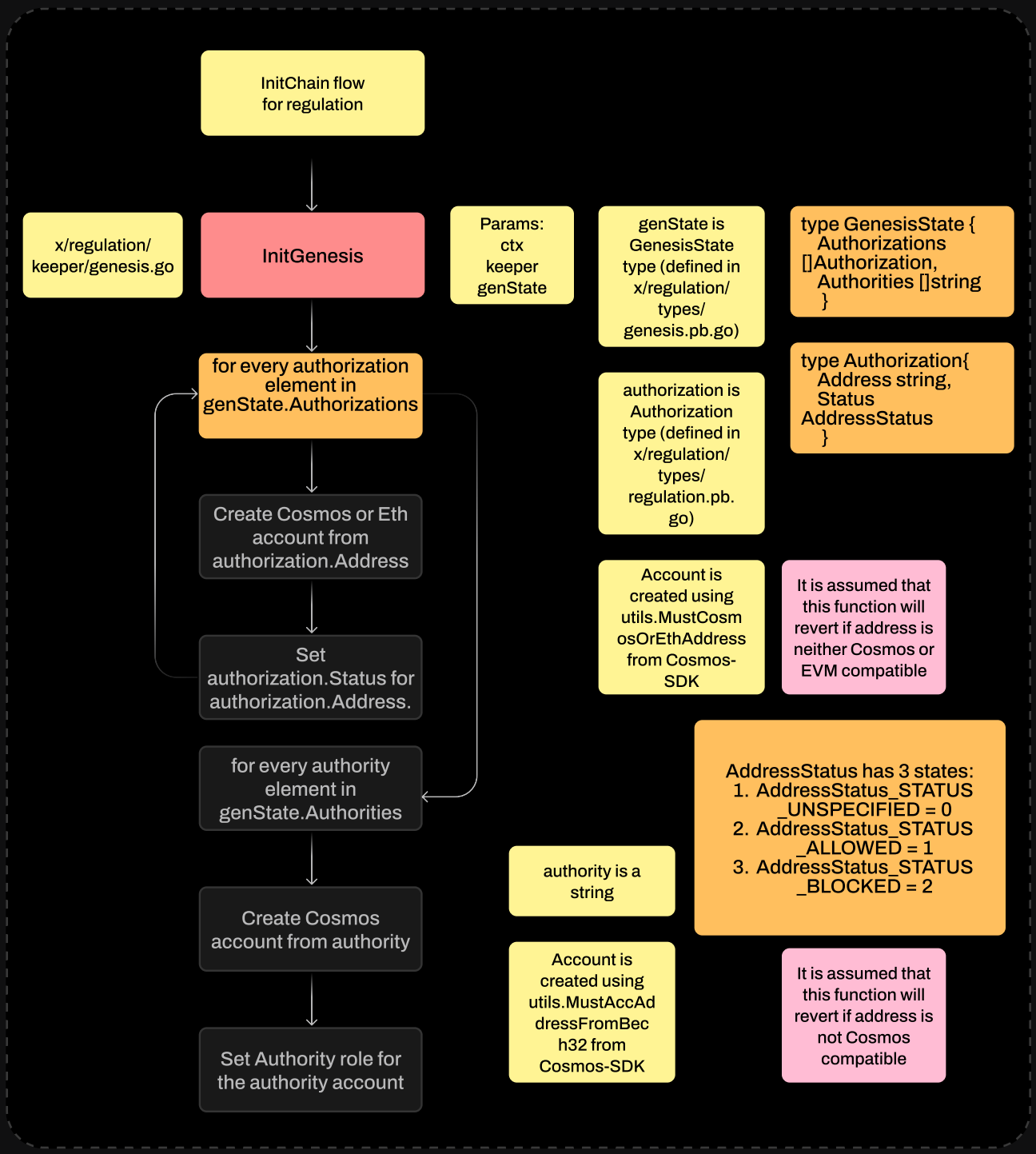
EMC coins.

Every time a user initiates a lock transaction, a certain amount of EMC coins is sent from the user to the LockKeeper module. Coins are locked for a certain period of time, after which they become available for withdrawal.

Only users themselves have access to their coins.

REGULATION

InitChain Flow



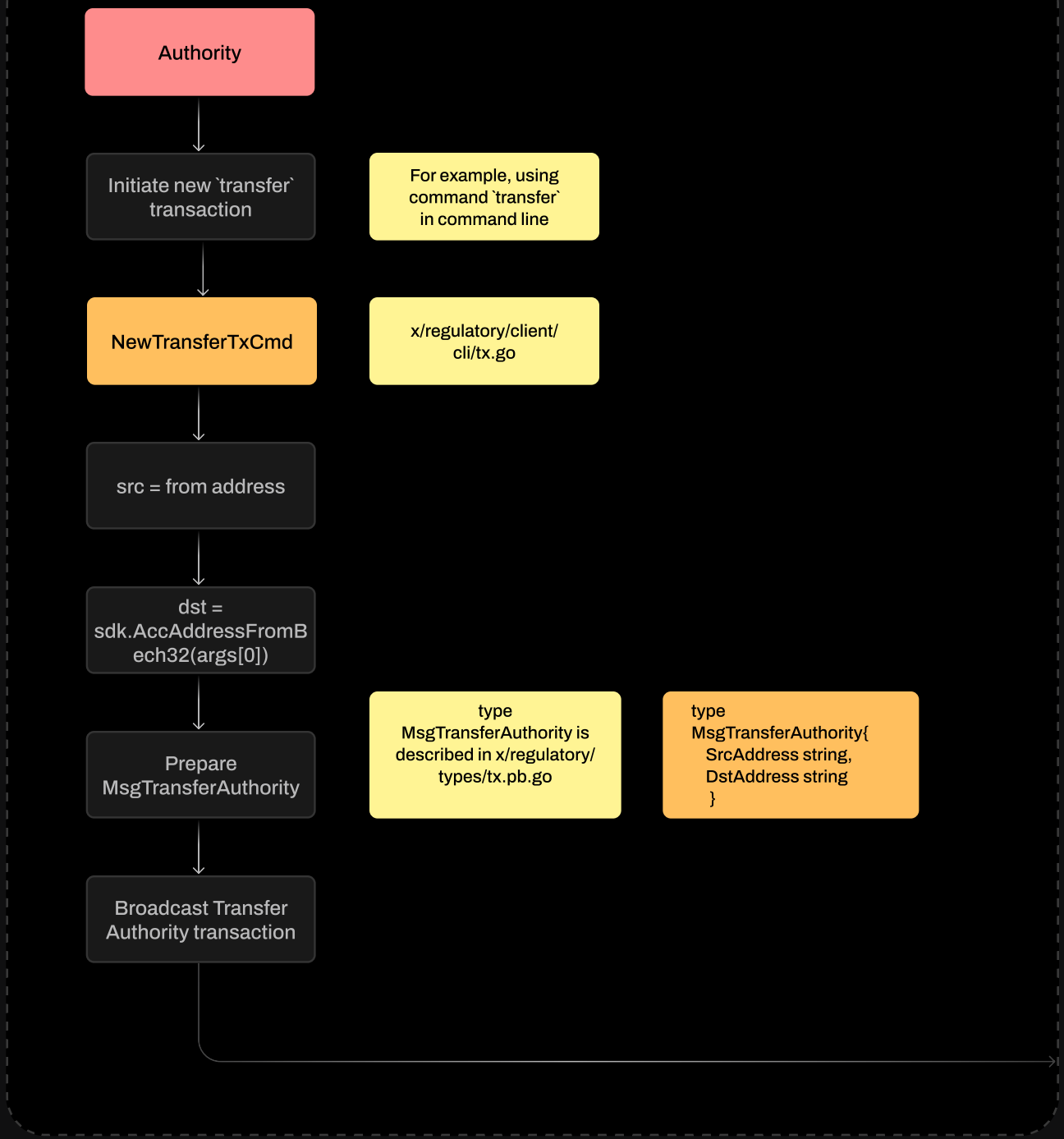
REGULATION

Authority users are set during genesis. New authhortiy users can't be added, however authority rights can be transferred. Thus, number of authorities should always be same as at the genesis.

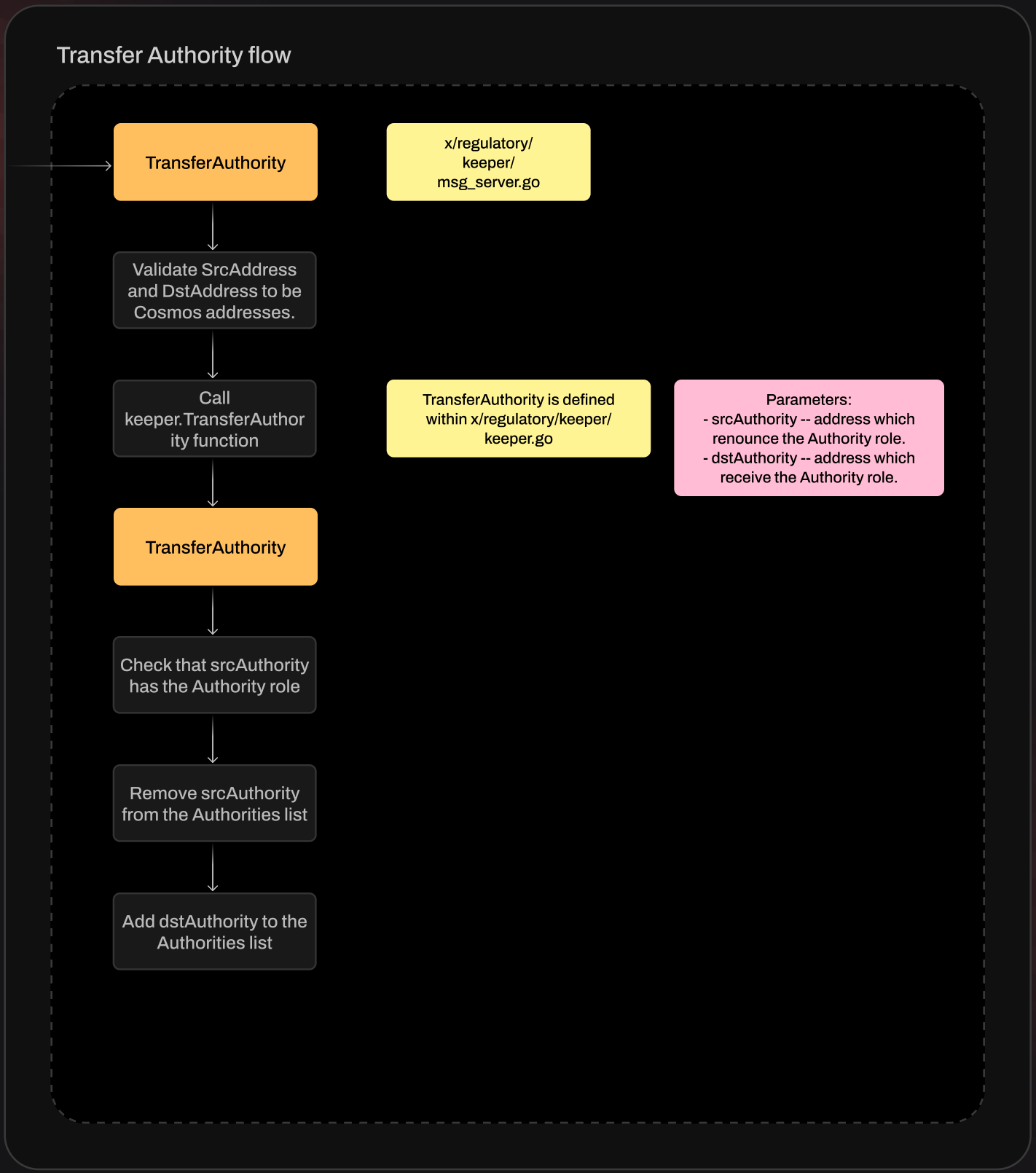
ExportGenesis flow

REGULATION

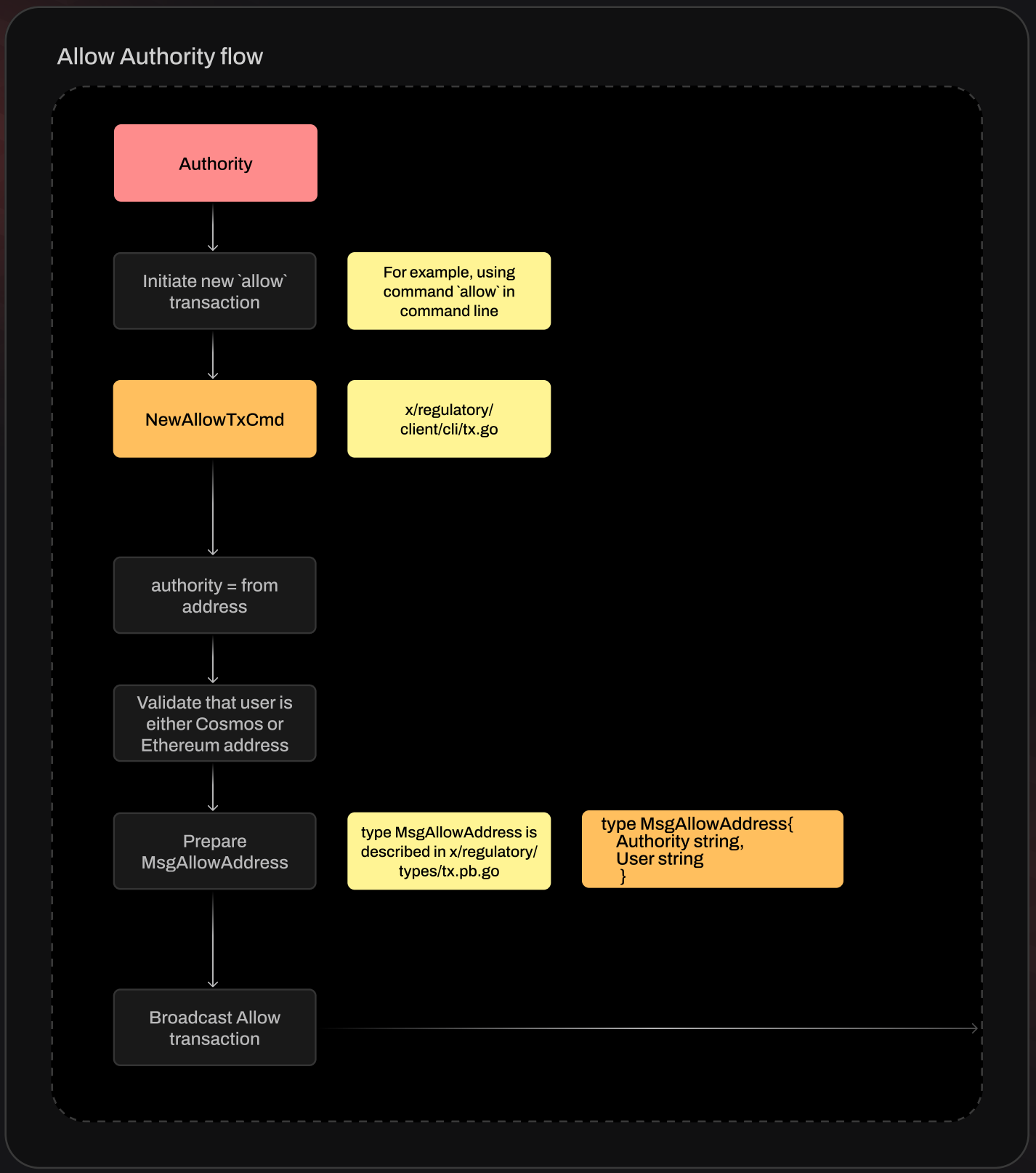
Transfer Authority flow



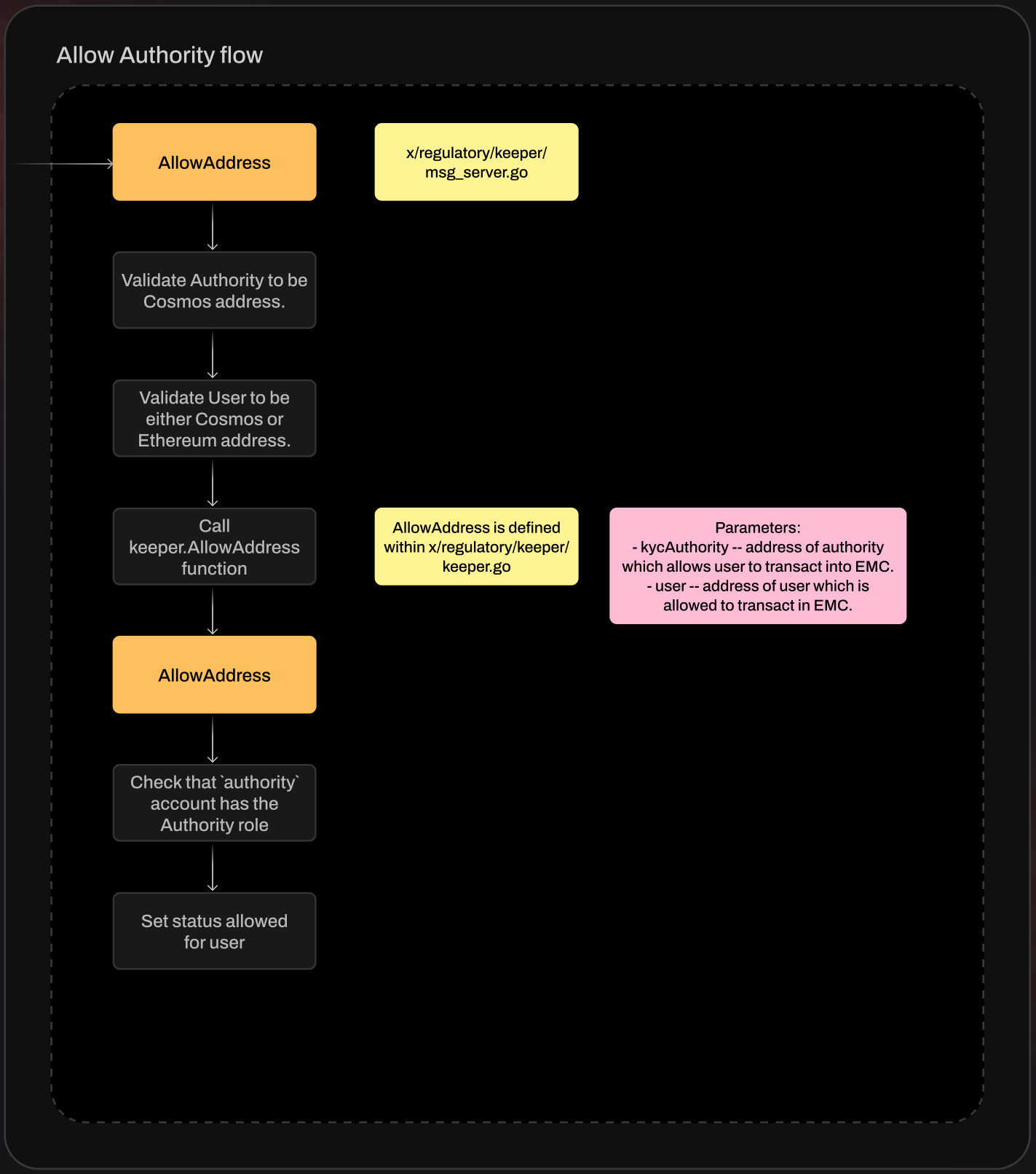
REGULATION


Transfer Authority flow
TransferAuthority
x/regulatory/
keeper/
msg_server.go
Validate SrcAddress and DstAddress to be Cosmos addresses.
Call keeper.TransferAuthority function
TransferAuthority is defined within x/regulatory/keeper/ keeper.go
Parameters:
- srcAuthority -- address which renounce the Authority role.
- dstAuthority -- address which receive the Authority role.
TransferAuthority
Check that srcAuthority has the Authority role
Remove srcAuthority from the Authorities list
Add dstAuthority to the Authorities list

REGULATION

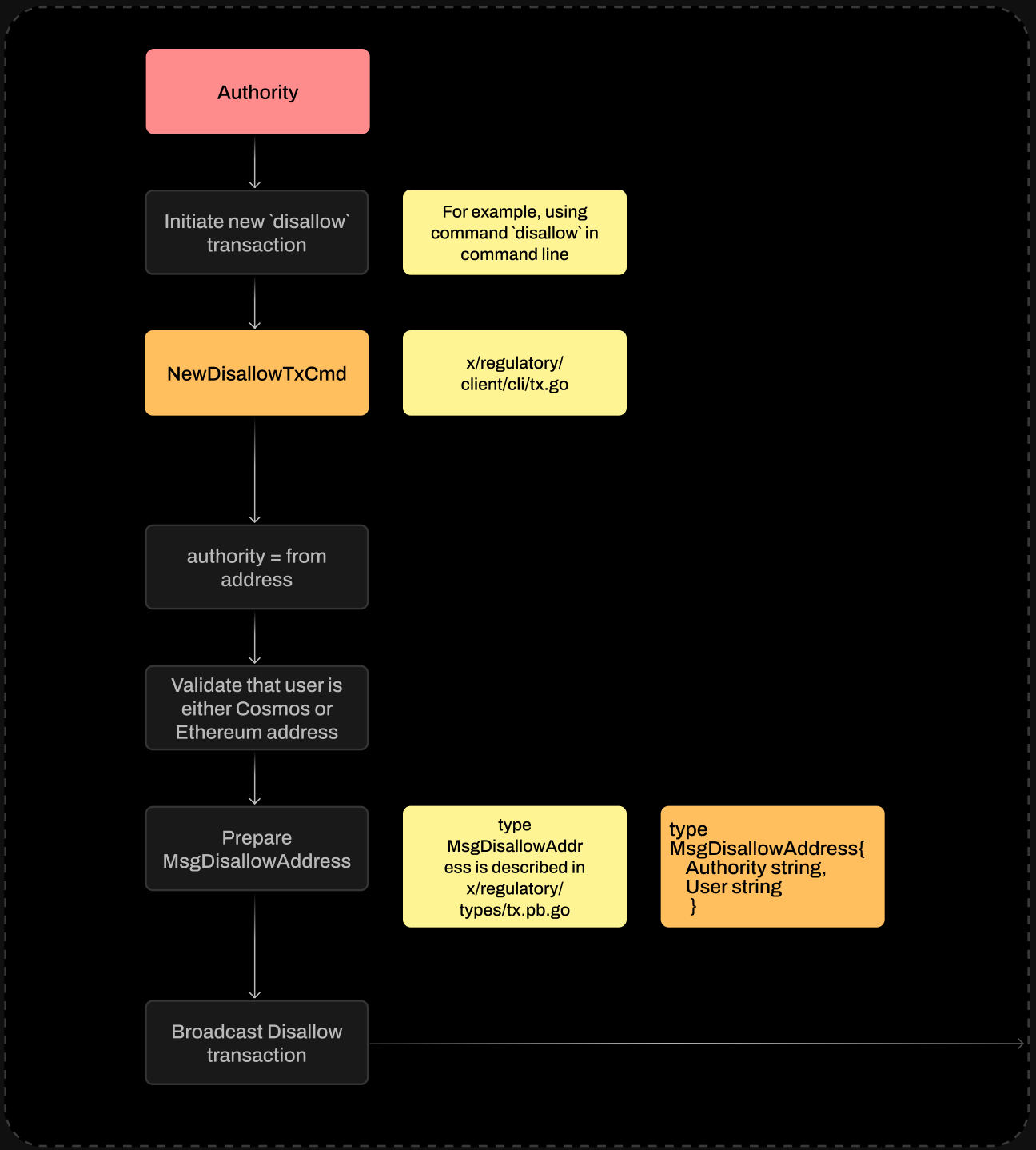

Allow Authority flow
Authority
Initiate new `allow` transaction
For example, using command `allow` in command line
NewAllowTxCmd
x/regulatory/ client/cli/tx.go
authority = from address
Validate that user is either Cosmos or Ethereum address
Prepare MsgAllowAddress
type MsgAllowAddress is described in x/regulatory/ types/tx.pb.go
type MsgAllowAddress{ Authority string, User string }
Broadcast Allow transaction

REGULATION


Allow Authority flow
AllowAddress
x/regulatory/keeper/
msg_server.go
Validate Authority to be
Cosmos address.
Validate User to be
either Cosmos or
Ethereum address.
Call
keeper.AllowAddress
function
AllowAddress is defined
within x/regulatory/keeper/
keeper.go
Parameters:
- kycAuthority -- address of authority
which allows user to transact into EMC.
- user -- address of user which is
allowed to transact in EMC.
AllowAddress
Check that `authority`
account has the
Authority role
Set status allowed
for user

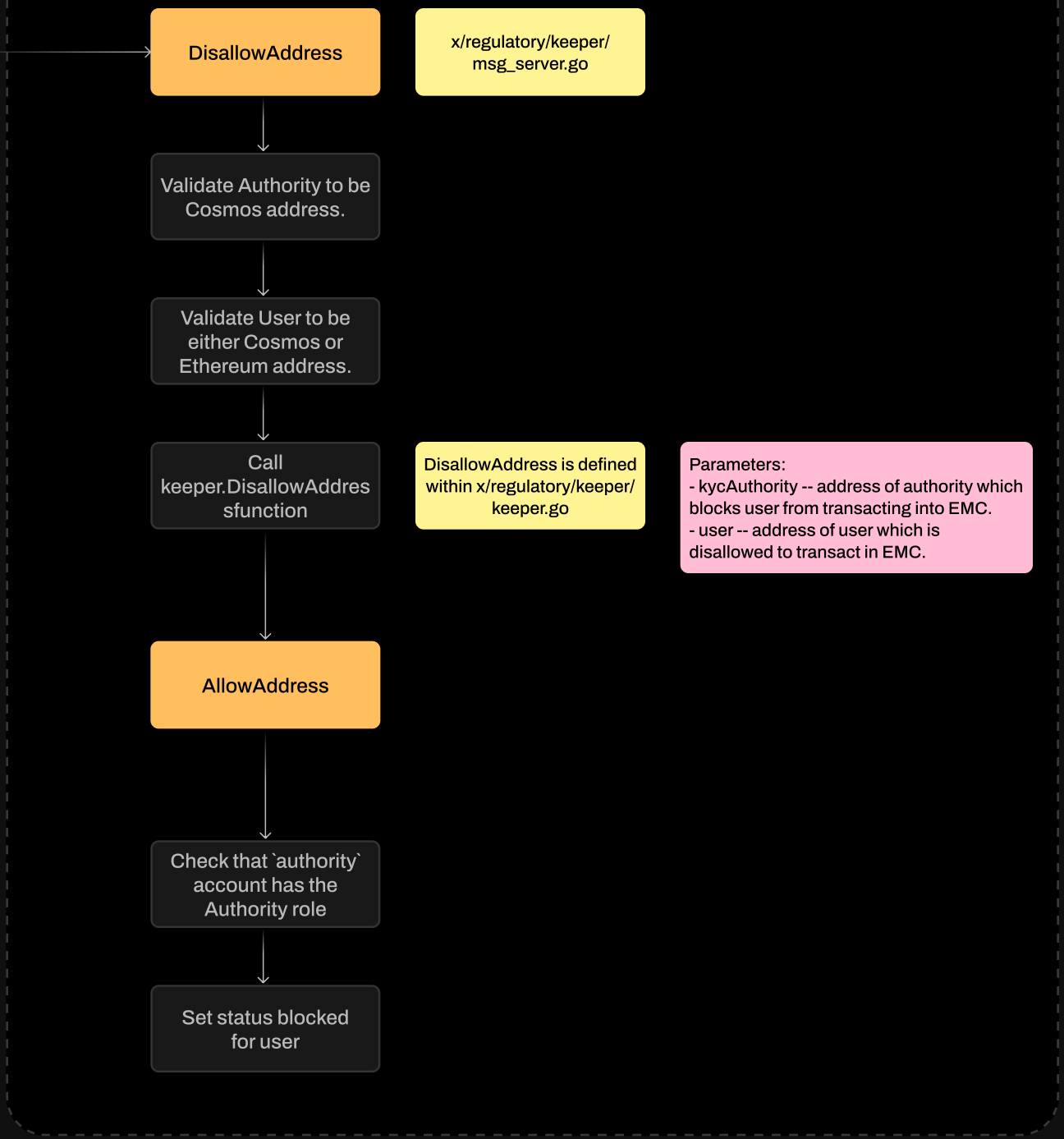
REGULATION

Disallow Authority flow



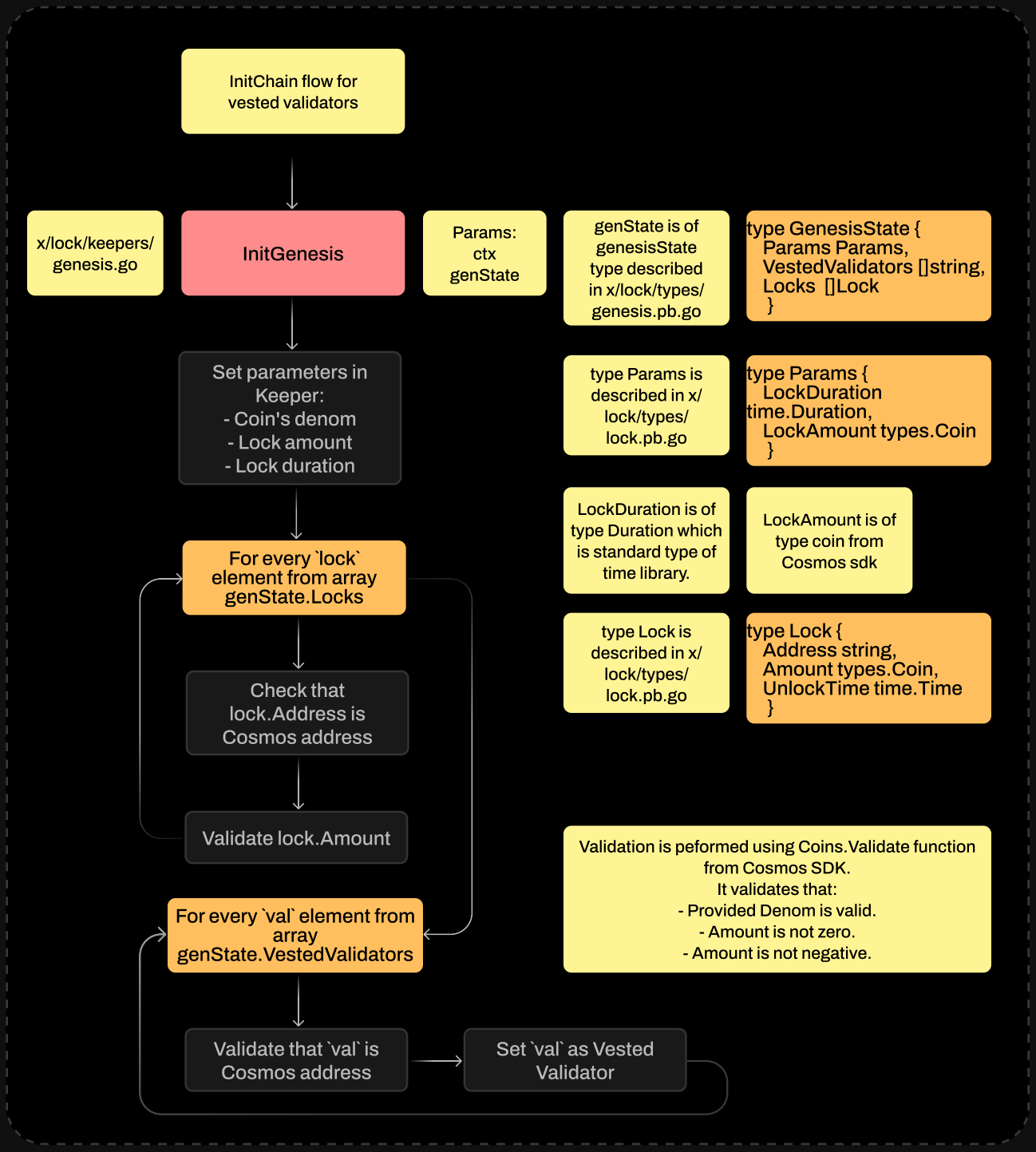
REGULATION

Disallow Authority flow



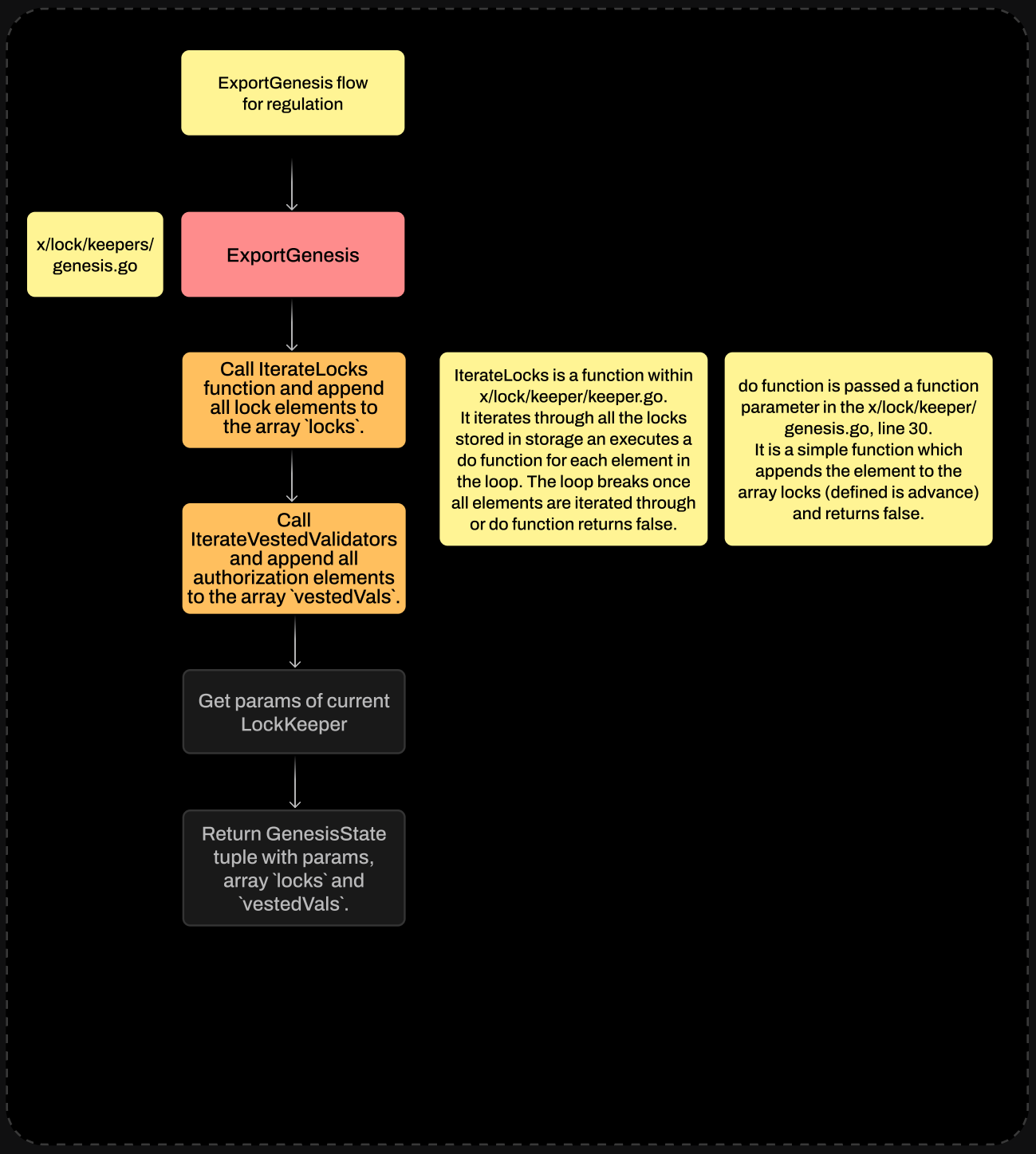
LOCK

InitChain Flow



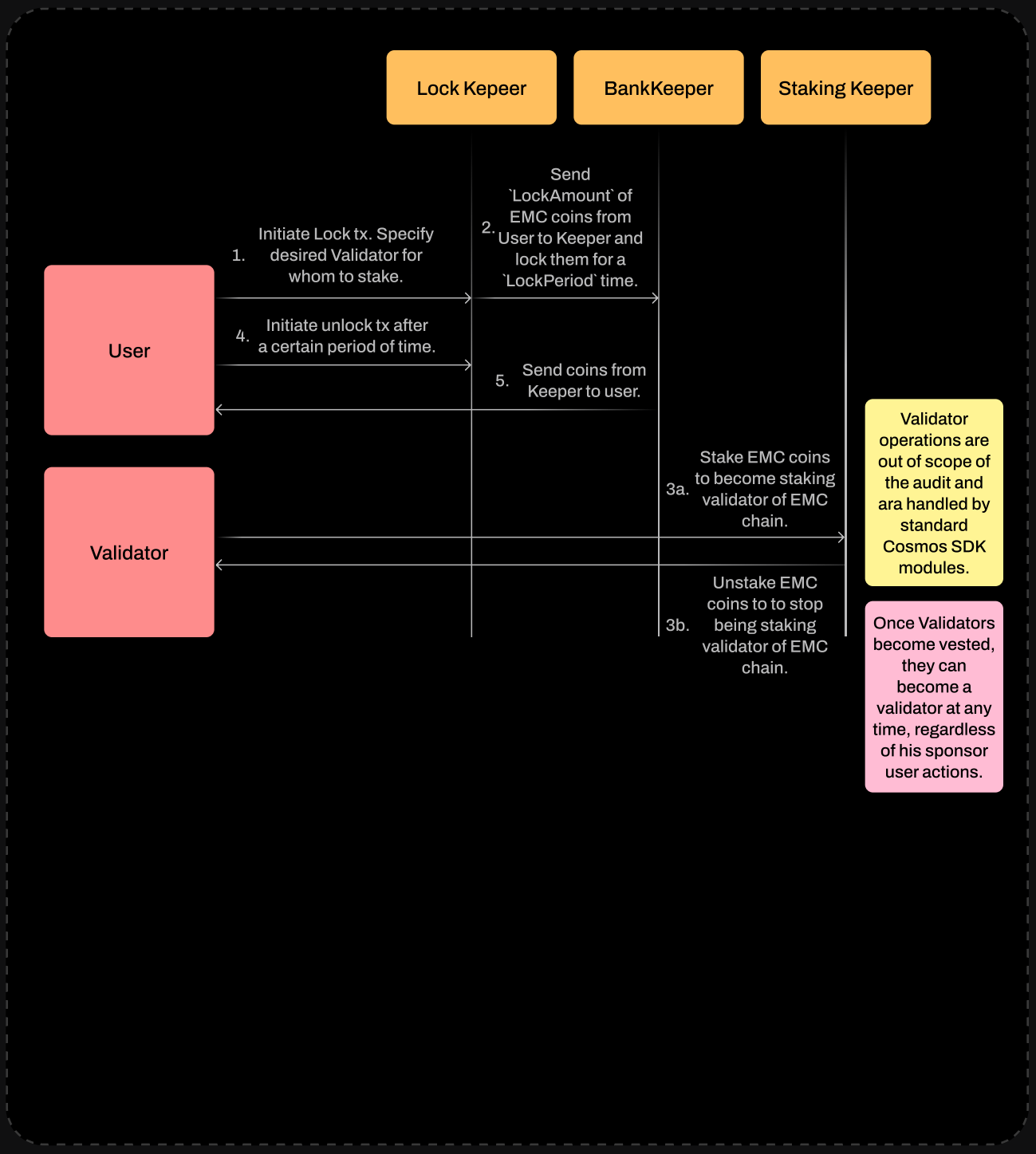
LOCK

ExportGenesis flow



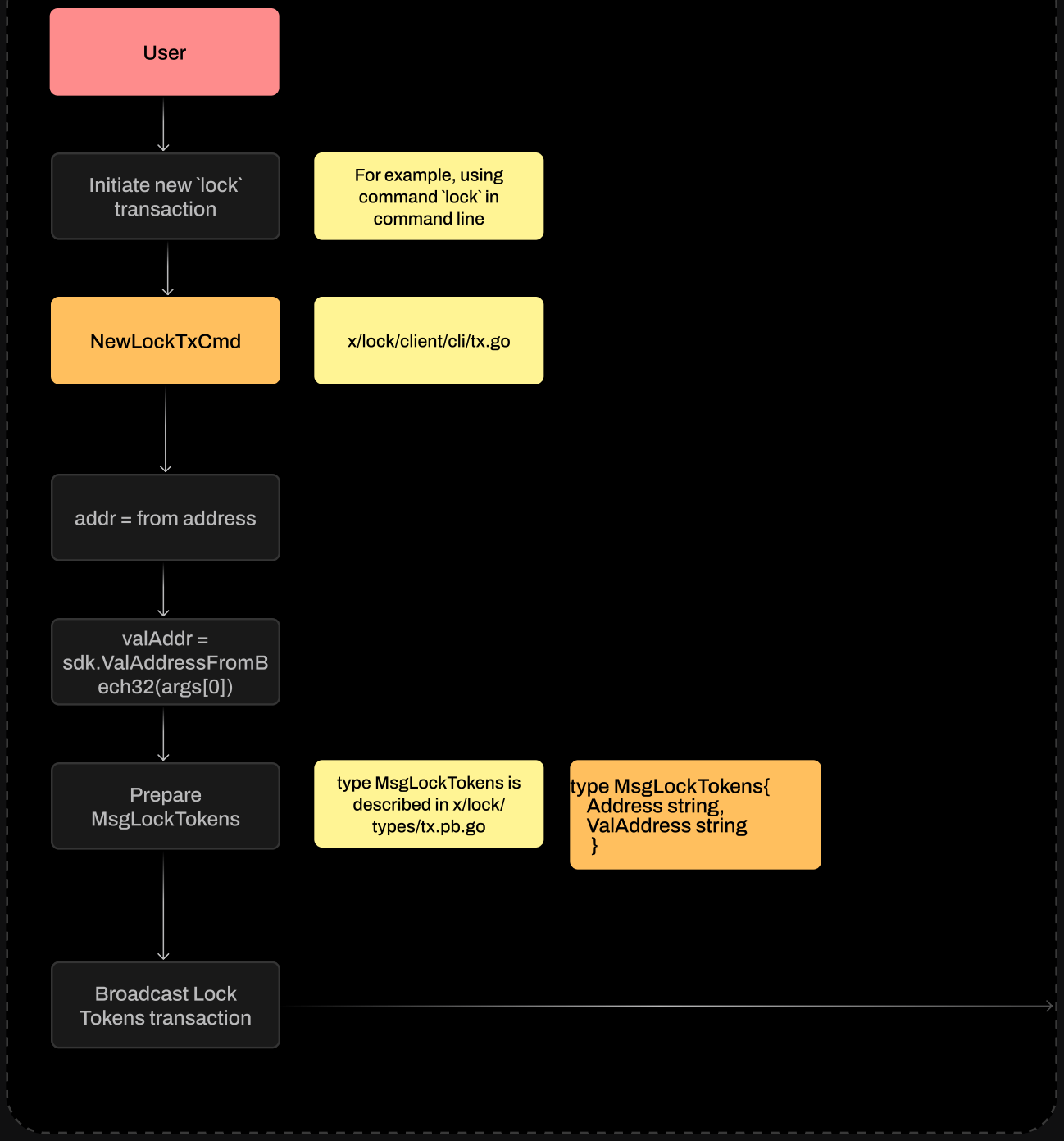
LOCK

Flow of Lock Module

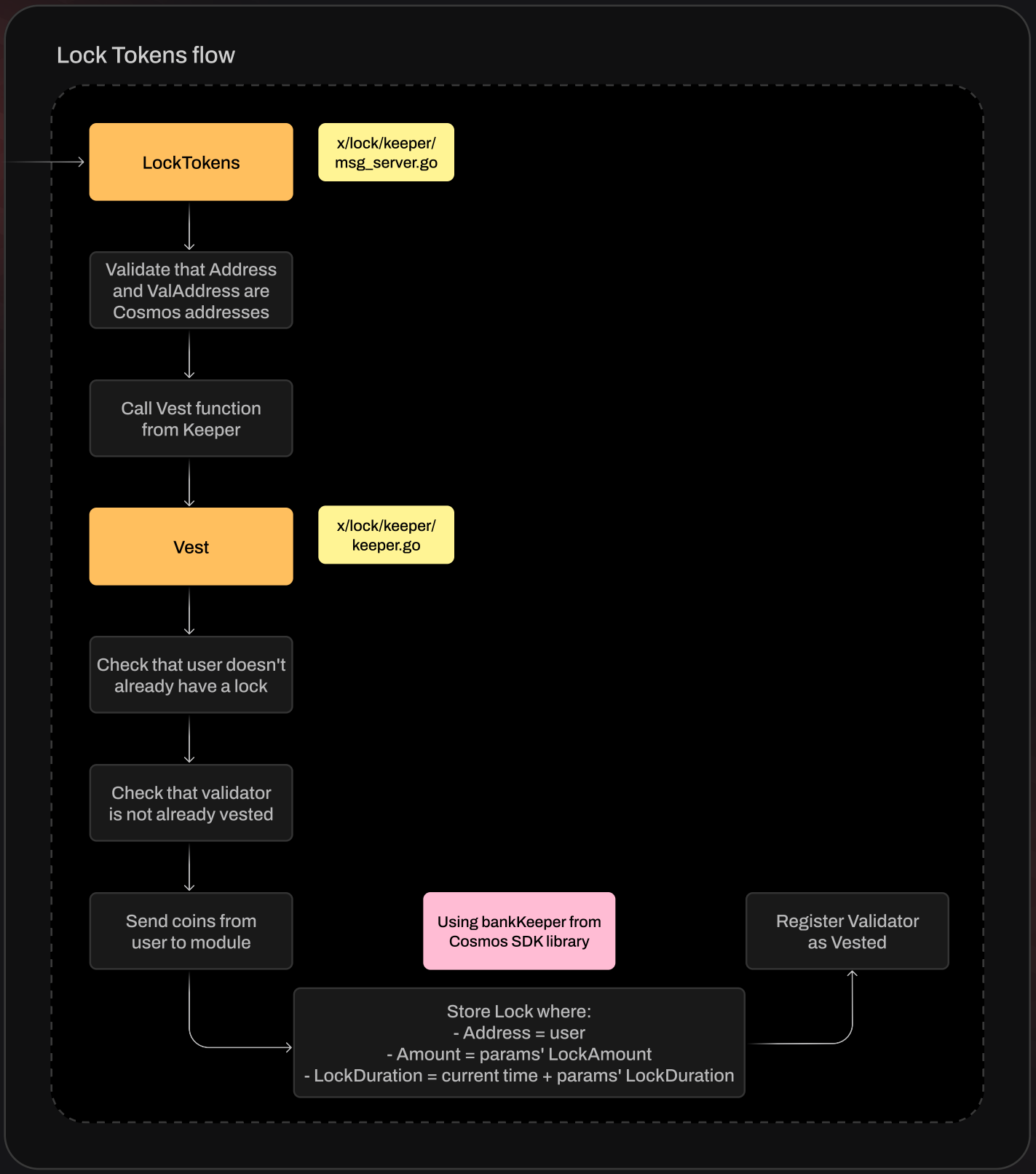


LOCK

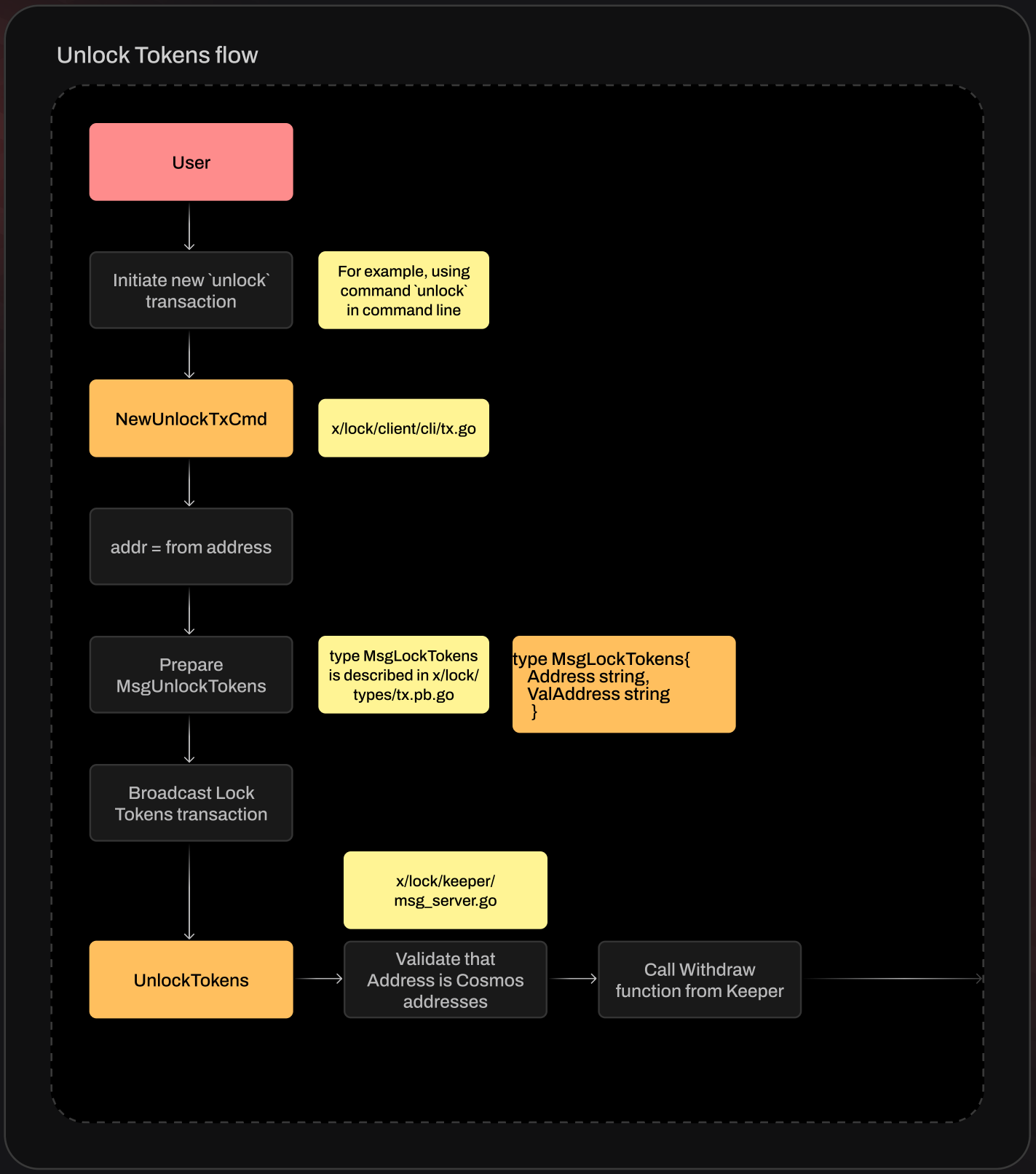
Lock Tokens flow



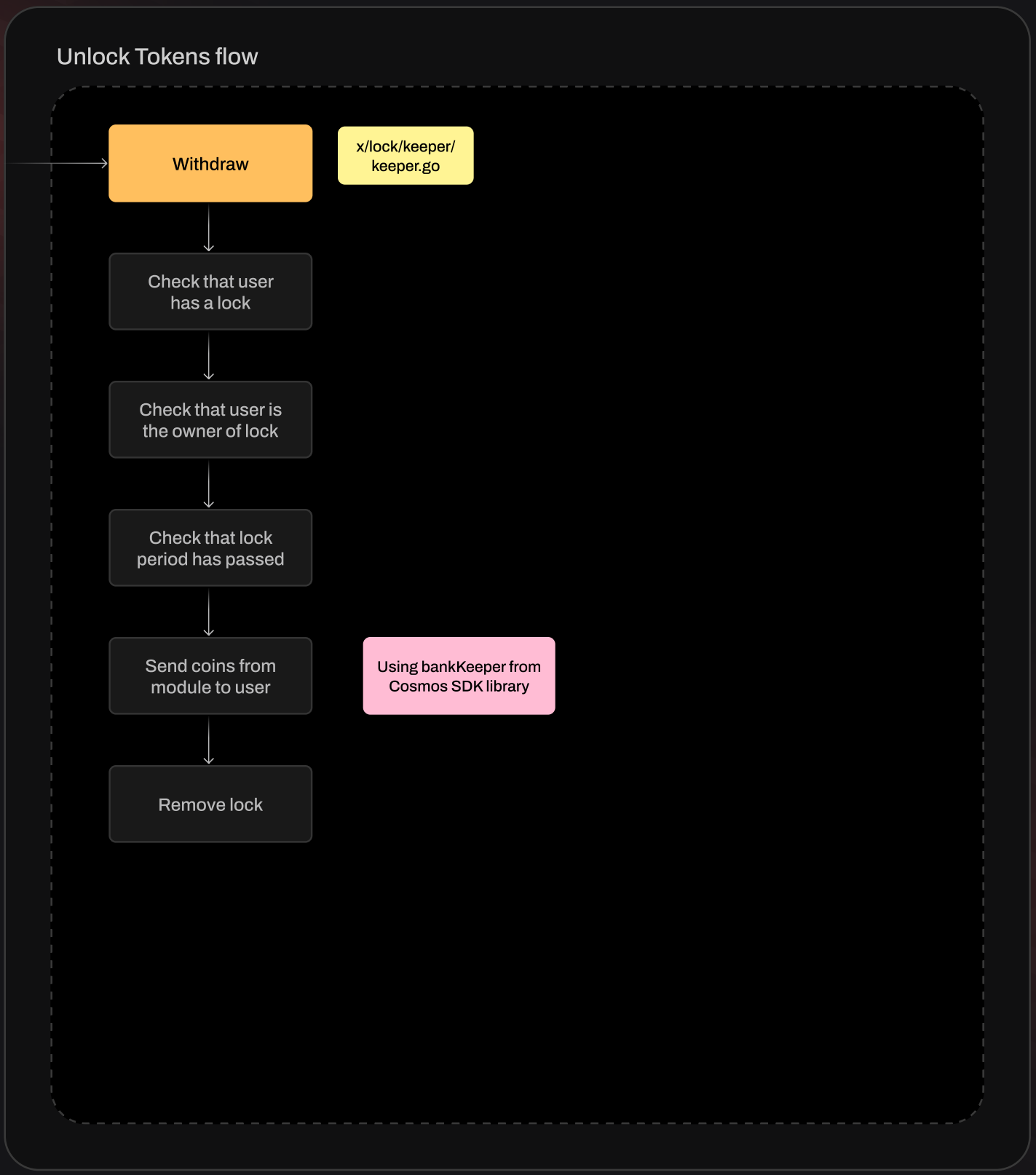
LOCK


Lock Tokens flow
LockTokens
x/lock/keeper/
msg_server.go
Validate that Address
and ValAddress are
Cosmos addresses
Call Vest function
from Keeper
Vest
x/lock/keeper/
keeper.go
Check that user doesn't
already have a lock
Check that validator
is not already vested
Send coins from
user to module
Using bankKeeper from
Cosmos SDK library
Register Validator
as Vested
Store Lock where:
- Address = user
- Amount = params' LockAmount
- LockDuration = current time + params' LockDuration

LOCK


Unlock Tokens flow
User
Initiate new `unlock` transaction
For example, using command `unlock` in command line
NewUnlockTxCmd
x/lock/client/cli/tx.go
addr = from address
Prepare MsgUnlockTokens
type MsgLockTokens is described in x/lock/types/tx.pb.go
type MsgLockTokens{ Address string, ValAddress string }
Broadcast Lock Tokens transaction
x/lock/keeper/msg_server.go
UnlockTokens
Validate that Address is Cosmos addresses
Call Withdraw function from Keeper

LOCK


Unlock Tokens flow
Withdraw
x/lock/keeper/
keeper.go
Check that user
has a lock
Check that user is
the owner of lock
Check that lock
period has passed
Send coins from
module to user
Using bankKeeper from
Cosmos SDK library
Remove lock

Complete Analysis

STANDARD CHECKLIST / VULNERABLE AREAS

Storage structure and data modification flow	Pass
Access control structure, roles existing in the system	Pass
Public interface and restrictions based on the roles system	Pass
General Denial Of Service (DOS)	Pass
Entropy Illusion (Lack of Randomness)	N/A
Order-dependency and time-dependency of operations	Pass
Accuracy loss, incorrect math/formulas other violated operations with numbers	Pass
Validation of function parameters, inputs validation	Pass
Asset management, funds flow and assets conversions	Pass
Signatures replay and multisig schemes security	Pass
Asset Security (backdoors connected to underlying assets)	N/A
Incorrect minting, initial supply or other conditions for assets issuance	N/A
Global settings mis-using, incorrect default values	Pass
Violated communication between components/modules, broken co-dependencies	Pass
3rd party dependencies, used libraries and packages structure	Pass
Single point of failure	Pass
Centralization risk	Pass
General code structure checks and correspondence to best practices	Pass
Language-specific checks	Pass

PROTOCOL RELATED CHECKS

Cosmos transactions: validator staking, delegation of tokens;	Pass
Evmos Ethereum transactions;	Pass
Integration of Staking with Lock module, dependency on locking of tokens in order to launch a new validator;	Pass
Adding/Removing of validator in the network;	Pass
Check correctness of parameters initialization during genesis, adding of initial locks and validators;	Pass
Check that an appropriate amount of coins is locked on module's balance during Lock Tokens transactions;	Pass
Genesis export;	Pass
Check correctness of genesis initialization, adding of initial allowed users and regulation authorities;	Pass
Check the correctness of Allow/Disallow transactions;	Pass
Check that allowed users can send both Cosmos and Ethereum transactions while disallowed can't;	Pass
Check the correctness of transfer authority transactions;	Pass
General Lock module implementation review	Pass
General Regulation module implementation review	Pass
Upgrade procedure	Pass
Correct addresses generation and conversion	Pass

DISCOVERED ISSUES

CRITICAL-1 Resolved

SAME DENOM IS REGISTERED TWICE WHICH PREVENTS SOURCE CODE FROM BEING BUILT

./cmd/emcd/main.go: line 16.
./cmd/config/config.go: line 55.

Whenever the code is built or the chain is started, an error "denom emc already registered" occurs, preventing the code's compilation. The cause of the error is in the RegisterDenom function from Cosmos SDK. This function has a check which validates the uniqueness of the passed denom. The function tries to register two denoms: DisplayDenom and BaseDenom. However, the value of each denom is the same and equal to the "emc". Due to such implementation, the function reverts when registering denom the second time.

RECOMMENDATION:

Rename the second denom.

POST-AUDIT.

Denoms have different names now.

CRITICAL-2 Resolved

GENESIS LOCKS ARE CREATED WITHOUT SENDING COINS TO THE MODULE.

./x/lock/keeper/genesis.go: InitGenesis()

During the genesis creation, the chain sets locks specified in the genesis file. However, coins are not sent from the related vesters to the module like they are in the Vest() function (./x/lock/keeper/keeper.go: Vest(), line 33). As a result, when users create locks and actually send coins, vesters specified in the genesis can withdraw these coins (which actually belong to other users).

Issue is marked as critical since it can lead to the loss of funds of users.

RECOMMENDATION:

Send coins from vesters to the module during genesis or ensure that genesis vesters cannot withdraw coins from the module's balance.

POST-AUDIT.

Coins are sent to the module during genesis now.

HIGH-1 Resolved

AUTHORITY ROLE CAN BE TRANSFERRED TO USERS WHO ARE ALREADY AUTHORITIES.

./x/regulation/keeper/keeper.go: TransferAuthority().

When an authority transfers the role, he specifies the address that will receive it instead. However, the address is not validated as not already being an authority. As a result, the source authority just renounced the role without actually transferring it to another account.
The issue is marked as high since it is possible to reduce the number of authority users (up to 1 left in the edge case) without the possibility of restoring the number of authorities. Even though authorities are probably validated accounts, leaving such a backdoor may be unsafe for the network, especially since there are no options to add new authorities.

RECOMMENDATION:

Validate if the destination authority already has the role.

POST-AUDIT.

Address which receives the authority role is now checked against being an authority.

HIGH-2 Resolved

EXPORT GENESIS RETURNS ONLY A SINGLE ELEMENT FROM THE STORE

./x/lock/keeper/genesis.go: ExportGenesis(), line 30, 36.
./x/lock/keeper/keeper.go: IterateLocks() line 103, IterateVestedValidators() line 125.
./x/regulation/keeper/genesis.go: ExportGenesis(), line 24, 29.
./x/regulation/keeper.go: IterateAuthorizations() line 71, IterateAuthorities() line 129.

All locks, vested validators, authorities, and authorizations are appended to the corresponding arrays and exported when genesis is exported. To append them, a function named do() is passed to the `Iterate` functions. The function returns false, which indicates that the iteration should not be stopped. However, in the `Iterate` functions, the loop stops once the false is returned, whereas it should only be stopped if the true is returned.
Consider the function IterateLocks() as an example. The current code process function do() as follows:

*** if **!do**(v) { break; } ***

Where, it should actually enter the body of if statement only if do() returns true, thus the code should look like:

*** if **do**(v) { break; } ***

As a result, right now only a single element of locks, vested validators, authorities and authorizations are exported which may significantly complicate the process of exporting the genesis of the network.

RECOMMENDATION:

Iterate through all the elements in the store. Break the loop in the `Iterate` functions only if do() returns the true.

MEDIUM-1 Verified

COMPROMISED AUTHORITIES CAN'T BE BLOCKED.

./x/regulation/keeper/keeper.go.

If the authority's private key is compromised, the attacker can perform malicious actions like blocking all the other network users or transferring the role to another account. As a result, such authority may prevent a network from operating correctly or at least keep a DDoS for some time.
Issue is marked as medium, since such vulnerability is only possible in case of key leaking and the malicious user will still require funds to send transactions - though it still represents a risk for the whole network.

RECOMMENDATION:

Consider adding a mechanism where a majority of authorities can block a malicious authority.

POST-AUDIT.

The EMC team verified that this is not the issue for the protocol.

MEDIUM-2 Resolved

AMOUNTS AND DURATIONS OF LOCKS CREATED DURING GENESIS ARE NOT VALIDATED

./x/lock/keeper/genesis.go: InitGenesis().

During the genesis, an array of initial locks is passed in the InitGenesis() function, where locks are set in the LockKeeper. However, the values of the amount and unlock time are not validated. Specifically,

- The amount is not validated against the amount set in the parameters.
- The duration is not validated against the duration set in the parameters.

As a result, genesis locks' amounts and durations are arbitrary.
The issue is marked as medium since initial locks are specified in genesis by the network's creator; however, such a backdoor might still be unsafe if it represents arbitrary data.

RECOMMENDATION:

Validate amounts and durations of locks set during genesis.

POST-AUDIT.

The issue was patched by the security team in the original repository: https://github.com/ScallopBank/chain/pull/18

Patch was transferred to the final repository.

MEDIUM-3 Resolved

VALIDATIONS FROM VEST() FUNCTION ARE OMITTED DURING GENESIS

./x/lock/keeper/genesis.go: InitGenesis().

Some essential lock validations from the Vest() function are omitted during the genesis. Specifically,

- Validation that specified lock. The address doesn't already have a lock created (for example, in case there are duplicate addresses in the array genState.Locks.). Such validation is present in the keeper.go: Vest, line 24.
- Validation that the element from genState.VestedValidators isn't already a vested validator (in case of duplicates in the array). Such validation is present in the keeper.go: Vest, line 28.

RECOMMENDATION:

Perform the validations which are specified in function Vest() during genesis.

POST-AUDIT.

The issue was patched by the security team in the original repository: https://github.com/ScallopBank/chain/pull/18

Patch was transferred to the final repository.

MEDIUM-4 Resolved

LENGTH OF LOCKS AND VALIDATORS SET DURING GENESIS MIGHT MISMATCH

./x/lock/keeper/genesis.go: InitGenesis().

The length of the locks and validators array is not validated to be equal during the genesis. Thus, the number of locks can be less than the number of initial vested validators.
Issue is marked as medium since it is currently assumed that for each vested validator, a lock must be created. However, the current implementation of InitGenesis() function allows to omit it.

RECOMMENDATION:

Validate that length of validators and locks is equal in the IniGenesis().

POST-AUDIT.

The issue was patched by the security team in the original repository: https://github.com/ScallopBank/chain/pull/18

Patch was transferred to the final repository.

INFO-1 Resolved

OUTDATED COSMOS SDK AND EVMOS VERSION.

The version of Cosmos SDK is specified as 0.45.8. However, several new versions have been released since this version. For example, the latest version of the 0.45.x line is 0.45.16, which contains several significant security features. Also, the 0.45.x line is no longer supported by Cosmos SDK developers, which is why Cosmos SDK recommends migrating to the latest line, which is currently 0.50.x.
The version of Evmos is set to 8.2.0 which is also considered outdated. It is recommended to migrate to the latest version (currently 16.0.3) which contains all the newest features and bug fixes.

RECOMMENDATION:

Update to the latest version within the line with which the code has been developed (0.45.16). Consider migration to the latest line of Cosmos SDK as it is frequently updated and contains new features and important security fixes.
Update the version of Evmos to the latest one.

POST-AUDIT:

The security team provided the patching of the network with the upgrade of the Evmos up to 12.1.6 version.

INFO-2 Unresolved

ABSENCE OF EVENTS.

./x/lock/keeper/msg_server.go: LockTokens(), UnlockTokens().
./x/regulation/keeper/msg_server.go: TransferAuthority(), AllowAddress(),

According to the Cosmos SDK documentation, msgServer's methods usually emit events to keep track of historical data, however no events are emitted in the methods of msgServers of Lock and Regulation methods.

RECOMMENDATION:

Add events in all the methods of msgServers.

INFO-3 Resolved

USER IS NOT CHECKED FOR NOT ALREADY BEING ALLOWED/ DISALLOWED.

./x/regulation/keeper/keeper.go: AllowAddress(), DisallowAddress().

When a user is allowed or disallowed from sending transactions with the corresponding methods, as a result, this may be confusing for the chain when the same user is allowed or disallowed without corresponding errors.

RECOMMENDATION:

Validate that the user is not already allowed/disallowed.

INFO-4 Resolved

STATUS OF AUTHORITY CAN BE SET TO DISALLOWED

./x/regulation/keeper/keeper.go: DisallowAddress().

The function allows the setting of the status of authority to be disallowed. Though the authority user will still be able to transact into EMC, he might lose such ability if he transfers the role. Also, leaving an ability to disallow authority users is confusing. Since it has no impact, it may imply that such an attempt should throw an error.

RECOMMENDATION:

Add a validation so that an attempt to disallow an authority will revert. The same check will apply to any change in the authority status.

POST-AUDIT:

Authority account cannot be disallowed now.

INFO-5 Unresolved

MULTIPLE TO-DO COMMENTS

Code contains several to-do comments across multiple files. In a general case to-do comment may imply that the logic is unfinished or is not in the final state and some of critical checks may be missing. Therefore it is highly recommended to review each of the comments and resolve it and finish the functionality (or remove the obsolete comments)

RECOMMENDATION:

Validate that all the necessary features are implemented in the code.

INFO-6 Resolved

UNUSED PARAMETER

./x/regulation/keeper/genesis.go: InitGenesis, parameter keeper.

Parameter is passed in the method, however it is never used in the body of the method.

RECOMMENDATION:

Remove the unused parameter.

INFO-7 Resolved

COMMENTED CODE.

./x/regulatory/client/genesis.go: AddAddressToGenesis().

The file contains commented code which may imply that the logic is unfinished. It is recommended not to leave commented code in a pre-production code.

The issue is marked as Info, though it can be re-evaluated based on the significance of the commented code.

RECOMMENDATION:

Remove or uncomment commented code.

INFO-8 Resolved

EMPTY FILE

./x/lock/keeper/vesting.go
File is empty which might imply an unfinished logic.

RECOMMENDATION:

Remove empty file or validate if it's necessary for future development.

INFO-9 Verified

HIGH DEPENDENCY ON THE OLD CODE VERSION

Current implementation of the EMC chain highly depends on the original implementation of the chain: github.com/ScallopBank/chain

Since the EMC chain implementation is an upgrade over the original Scallop chain, auditors understand the necessity of checking the upgrade before merging into the original chain. However, the current dependency structure is confusing, which may cause issues during the merging process and require an additional review.

It is highly recommended to have the full integrity of the codebase and work on the finalized version of the code.

RECOMMENDATION:

Verify the dependency on the original codebase, perform necessary merge operations OR remove the unnecessary dependency on the outdated codebase to keep the integrity of the developed module.

POST-AUDIT:

The team has verified that the issue is not applicable to the current codebade.
The final repository has a clean dependencies list.

INFO-10 Verified

VESTED VALIDATOR CAN REMOVE LOCK AFTER BECOMING VALIDATOR.

x\lock\keeper\keeper.go: Withdraw()
There are no restrictions for VestedValidtor during the withdrawal process. Thus, the lock acts as a temporary deposit for becoming a validator, but after that step, it can be withdrawn with no preconditions (after the appropriate lock duration). Even more, VestedValidator is not removed from the validators list after the withdrawal.
The issue is marked as Info, as it refers to the business logic decision and cannot be classified without confirmation from the team. However, auditors see it as substandard behavior and require confirmation from the team.

RECOMMENDATION:

Verify that the module presents desired behavior:

- vested validator can remove his lock (after the appropriate duration) without loosing the privilege of being the network validator
- vested validator is never deregistered even after the withdrawal

POST-AUDIT:

Verified by the EMoney team:

- validators remain as such after lock If they hold voting power as bonded token
- and so - validators are de-regestered only if the bonded tokens are unbonded

INFO-11 Unresolved

HIGH DEPENDENCY ON THE OLD CODE VERSION

The authority can mark the vested validator as disallowed, so the user will lose the ability to perform transactions - and will not be able to withdraw his lock.
The issue is marked as Info, as it refers to the business logic decision and cannot be classified as a security issue without confirmation from the team. However, auditors see it as a sub-standard behavior and require a double confirmation from the team.

RECOMMENDATION:

Verify that it is a desired behavior or implement the lock rescue/penalty functionality for locks of disallowed users

INFO-12 Verified

NO QUERY FOR GETTING THE LIST OF AUTHORITIES.

./x/regulation/client/cli/query.go
There is no query for getting the list of authorities or checking whether the given address is authority. Due to this fact, it might be difficult to get the current list of validators in the runtime.

RECOMMENDATION:

Verify that the module presents desired behavior:

- vested validator can remove his lock (after the appropriate duration) without loosing the privilege of being the network validator
- vested validator is never deregistered even after the withdrawal

RECOMMENDATION:

Verified by the EMC team as not required functionality at the moment.

INFO-13 Unresolved

TX FEES REQUIRE MORE COINS THAN LOCKED

Based on the current setup of the network, each lock requires 10k EMC coins to lock. Meanwhile, each transaction requires 20k EMC coins as gas fees. Thus, unlocking 10k coins becomes pointless and unprofitable as unlocked coins won't even cover gas fees.

RECOMMENDATION:

Validate if lock amount and gas fees won't be changed for the mainnet. Consider making unlocking more meaningful for the users.

POST-AUDIT:

The EMoney team verified, that the issue will be resolved in the next development iterations during transition to the next testnet version.

Disclaimer

The information presented in this report is an intellectual property of the customer, including all the presented documentation, code databases, labels, titles, ways of usage, as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else's requirements and be fully secure, complete, accurate, and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions, and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool, which helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.