CERTIK

Audit Report

Produced by CertiK

for e-Money

April 16th, 2020

DISCLAIMER

ABOUT CERTIK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK’s mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that project is checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and verifications on the project, in turn creating a more secure and robust software system.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance’s BGBP and Paxos Gold to decentralized oracles such as Band Protocol and Tellor.

CertiK customizes its engineering tool kits, while applying cutting-edge research on smart contracts, for each client on its project to offer a high quality a delivery. As utilizes technologies from blockchain and smart contracts, CertiK team will continue to support the project as a service provider and collaborator.

For more information, visit https://certik.io

EXECUTIVE SUMMARY

Application Summary	
Name	e-Money Ledger
Commit SHA	20c81b329b09553cefbaa30c76f3ae4c78726cf4
Type	interest-bearing, currency-backed tokens
Platforms	Cosmos SDK v0.37

Engagement Summary	
Dates	02/12/2020 — 03/13/2020
Method	Whitebox Analysis
Consultants Engaged	3
Level of Effort	6 calendar weeks

Vulnerability Summary	
Major Functional Issues	1
Minor and Informational Issues	23
Semantic Issues	28

ENGAGEMENT OBJECTIVES

Verify the soundness of the implementation, while ensuring its logic meets the specifications and intentions of our client, underlying NGM's utility as a token — in order to:

- provide an estimate of the overall security posture of the system;
- evaluate the difficulty of system compromise from an attacker;
- identify design-level risks to the security of the system;
- identify implementation flaws that illustrate systemic and extrinsic risks;
- provide recommendations for best practices that could improve e-Money's security posture;

- document architectural risks to the system in the form of a threat model and data-flow analysis of the prioritized system components;
- provide a reference architecture that the community may use to evaluate the coverage of the security assessment, and to begin building a baseline of security relevant settings and considerations for the system.

AUDIT METHODOLOGY

The key to building and maintaining highly secure and reliable systems is simplicity — a good system will have nothing to take away, rather than nothing to add. Each component requires securing, updating and debugging in perpetuity, which is an order of magnitude more complicated than building it in the first place. There's a common saying that if you engineer the most clever system you can imagine, you're by definition unqualified to maintain it. Like open source projects which reject code not because it's bad, but because it may not be worth maintaining, the CertiK team carefully examined each module in the e-Money stack while weighing its benefits against the extra complexity it introduces. We began by studying e-Money's most recent white papers in order to synthesize its full scope of features, validate their correctness from an economic perspective by reasoning mathematically about their underlying theoretical clauses, and qualify — according to the functional categories listed below — the model's robustness:

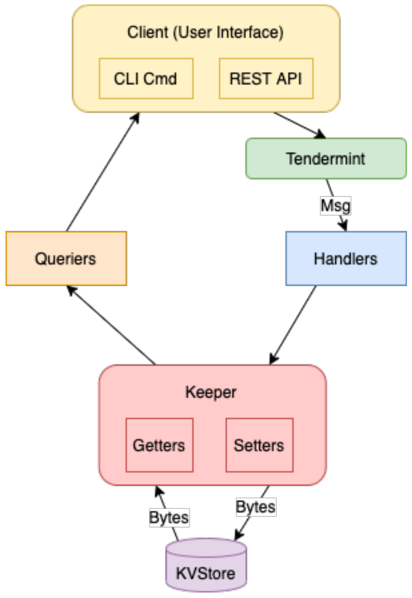
- **Prevention** - properly hardening systems with guardrails to prevent incidents from happening in the first place, just like sturdy walls and secure locks are the best defense against burglary;
- **Detection** - like with a good home alarm system detecting an incident is the next best thing after preventing it, especially in areas performing validation, decoding, type conversion, or where assumptions are being made;
- **Response** - since `error` is only a type in Go, when it is returned it does not change a program's execution flow on it's own like a `panic` statement would, so properly responding to security incidents if/when they arise through system rollback/lockdown is crucial;
- **Monitoring** - implementation of a perceptive transaction lifecycle monitoring cache to encourage prevention and detection as described above.

CertiK found e-Money's theoretical model, as well as its `0.6.0`-release Cosmos SDK implementation, to be well-designed and executed cleanly, demonstrating a good command over relevant best practices. While CertiK cannot comment on the final mainnet performance of our clients' end-products, the technological architecture and economic reasoning behind it were determined to be sound overall. From an operational perspective, after running `bdd_test.go`, which spins up a 4-node chain using docker-compose network and runs through various scenarios, we found no concerns while examining what happens when a malignant validator (or several of them) changes the default P2P consensus configuration (such as the `timeout_commit` option).

SYSTEM OVERVIEW

Tendermint enables the network to handle 1/3 malicious nodes and still maintain byzantine fault tolerance. This is because validators are required to bond (lock up NGM) the native currency of e-Money and can have their stake slashed for bad behavior (ex. proposing bad blocks). Validators take care of messaging (observation, ordering, and transport) and authentication replicating state on external blockchains via:

- *A* **Client** —> validator runs this for verifying consensus transcripts;
- *its* **Connections** —> driven by the Keeper to track associated state between chains;
- their **Channels** —> the flow of assets between via liquidity providers in the market is controlled by issuers, who are in turn controlled by the Authority.



ARCHITECTURE

NETWORKING LAYER

This layer, handled by the Tendermint library, propagates transactions that happen on one localized state machine across all other state machines (nodes) in the network to be processed in the consensus layer.

CONSENSUS LAYER

This layer comprises the algorithm which is responsible for ensuring that state stored on every state machine is the same after a transaction happens (i.e., machines can't fake transactions that never existed).

INCENTIVE LAYER

This layer drives consensus, and is liable to suffer from Censorship attacks and transaction withholding (resulting in reduction of perceived uptime, reduction of rewards from transaction feeds).

APPLICATION LAYER

Together the consensus and incentive layer govern this layer, which is responsible for defining the possible state transitions and updating the state of a state machine after a transaction occurs. The design of e-Money is intentionally centralized as the project is developed by a private company and self-funded. e-Money is both centralized in the sense of staking token distribution, but also by way of the "Authority" module which controls: adding & removing Issuers of new currency-backed tokens, white & blacklisting of tokens. The token white & blacklist can only be configured at genesis.

Commission charged on staking rewards is the main revenue source. Currency-backed tokens are used for fees. Transaction fees can be paid in any token, not just their staking token (NGM). The business model does not rely on additional fees related to issuance or redemption. A seller of currency-backed tokens will thus be paid a part of the bank reserve, that is proportional to the amount of tokens sold out of the total issuance.

In current Tendermint code, the block proposer decides transaction ordering (irrespective of gas prices). Front running can occur with malicious validators. Practically speaking, due to the (centralized) NGM token distribution, e-Money.com will be "cherry-picking" validators through delegation and will be keeping an eye out for inappropriate behavior. Hence we make the conclusion that a Byzantine network takeover by malicious nodes is out of scope, since the majority of NGM staked will be controlled by e-Money. The e-Money A/S treasury tokens will be released linearly over 3 years.

There are no mechanisms to pause the system, revert/undo transactions, modify balances or manage upgrades. There is no on-chain governance. In the event of a hard fork, the company solely decides which fork to redeem tokens from. Bank balances and the resulting calculation of exchange rates is done off chain. Bank balances and rates (the exchange rate will vary depending on the interest accrued on the reserve) will be displayed prominently on the web site along with proof-of-funds. Once the versioning upgrade module that is being upstreamed into `cosmos-sdk` has matured, it will be added to e-Money.

The size of the bank reserve and number of currency-backed tokens will vary independently of each other, the only constant is that ownership across collateral and tokens is proportional. In other words 10% of tokens can be sold to receive 10% of the bank reserve. The supply of currency-backed tokens is continuously increased through inflation, which is immediately distributed as rewards to stakers of the NGM token. This constitutes the markup and is a central incentive for owning and staking NGM tokens.

A staked NGM token represents a proportional claim on all future staking rewards consisting of currency-backed tokens. The value of each NGM token thus scales with the value of the bank reserve, which maintains a constant buy-side on exchanges. This ensures that the market value of the token tracks its underlying currency consistently.

SUMMARY AND RECOMMENDATIONS

As stated in the original report, e-Money's theoretical model and implementation are well-designed and executed cleanly, demonstrating a good command over the relevant best practices.

Regarding the [Major] finding (page 18 on the original report), **the team has addressed it properly** as seen on the commits here: https://github.com/e-money/em-ledger/commit/5c5c655bf32d4cf6ee54942796228eb83b9df116
And here: https://github.com/e-money/em-ledger/commit/795109c2497513957dfe2a9ca084279e34c9bb90

Additionally, the team has also addressed most of the issues in the report (all info and semantics that don't impact the system) here:
https://github.com/e-money/em-ledger/commit/808f428afa5b8967e5269b39f4d742864a064009

Given the that the audit was targeting both **the architecture and the codebase** of the e-Money project (both the theoretical model and its actual implementation), the audit **gives us the ability to rate the project with a high degree of confidence**. The issues that were not addressed are **all non-system impacting**, and given the ethics of the team we are certain that will be addressed in coming commits, to improve the quality and readability of the codebase in time.

FINDINGS & RECOMMENDATIONS

ADDRESSED ISSUES

- *https://github.com/e-money/em-ledger/commit/808f428afa5b8967e5269b39f4d742864a064009*

`./x/authority/types/errors.go`

- Line 25: `CodeMissingDenomination` should be `CodeNotAuthority.`

`./x/inflation/internal/types/keys.go`

- It makes more sense to use strings that are more descriptive of their underlying variables
  - If not, set each variable to `ModuleName` for consistency (unlike line 25)

`./x/authority/types/msgs.go`

- `ValidateBasic` checks that all messages are invalidating if the decoded struct members are empty, and there is no explicit check, for example if a valid address has been assigned in the way that,
  ```
  if !msg.GasPrices.IsValid() {
      return sdk.ErrInvalidCoins(msg.GasPrices.String())
  }
  ```
  - There is no such validation in the handler as well.

`./x/issuer/keeper/keeper.go`

- This error message needs to be improved to specify which denomination specifically doesn't belong to the issuer: `k.logger(ctx).Info("Issuer operation attempted by non-issuer", "address", address)`

`./x/issuer/client/cli/tx.go`

- The function `getCmdSetInflation` is using the args value with `denom := args[1]` although all the rest of the args, in all functions, are validated with parsers or functions that return an error or with the `msg.ValidateBasic()` function.
  - In this case (line 39) the code fails to check the validity of the `denom` variable.
  - Validation of the `denom` variable or the usage of the `msg.ValidateBasic()` needs to be implemented.

`./x/issuer/types/errors.go`

- `ErrDenominationAlreadyAssigned` can be made more informative like the others in the this file: denomination (which?) is already under control of an issuer (whom?).

`./x/authority/keeper/keeper_test.go`

- Line 65 in `CreateIssuer`, additionally to `util.ValidateDenom` there should be a check for there being no duplicate denoms, perhaps by implementing a `util.ValidateDenoms`.

`./util/denom_utils.go`

- Implement a `util.ValidateDenoms`, which calls `ValidateDenom` internally while also checking for duplicates.

`./x/inflation/internal/types/events.go & errors.go`

- There are several events and error codes not used in the project, and should be commented out.

`./hooks/auth/proxykeeper.go`

- It seems this comment should be referring to the auth module
  - "expected interface is copied from the bank module"

`./hooks/bank/bankproxy_test.go`

- One store is mounted although three keys are declared (the other two are only passed as params to instantiate the bank proxy keeper)
  - Mounting
    - `ms.MountStoreWithDB(authCapKey, sdk.StoreTypeIAVL, db)`
  - Declaring
    - `keyParams  = sdk.NewKVStoreKey("params")`
    - `tkeyParams = sdk.NewTransientStoreKey("transient_params")`

`./app.go`

- Another difference is that `MakeCodec()` should `amino.Seal()` prior to returning the codec

- *https://github.com/e-money/em-ledger/commit/795109c2497513957dfe2a9ca084279e34c9bb90*

`./x/issuer/keeper/keeper_test.go`

- Line 294, the issuer keeper needs to be initialized with the keyIssuer `StoreKey` not the `keySupply` one

`./x/inflation/internal/keeper/keeper.go`

- Line 60, remove TODO.

`./x/inflation/internal/types/inflation_state.go`

- Remove TODO on line 22, in our opinion this is unnecessary and it suffices to start accumulating at 0 from genesis

OBSERVED ISSUES

`./app.go`

- *SEMANTIC*
  - Put towards top of the file near AppModuleBasic

    -
      ```
      type AppModule struct {
              AppModuleBasic
              keeper Keeper
      }
      ```

`./x/market/types/types_test.go`

- *MINOR*
  - There is no type test involving `ExecutionPlan`
  - In `TestOrders1`, there should be an expected error on the attempt to add another order to one of the addresses with the same id "A"

- *SEMANTIC*
  - It would make a lot of sense in `TestSerialization` if the assertions for `Source` and `Destination` were not simply `> 0` , but reflecting the actual initialization values of 100 and 120, respectively

`./x/market/types/types.go`

- *SEMANTIC*
  - Order struct fields should have corresponding comments, like they are laid out here `./docs/market.md`
  - It is worth noting in a comment, especially, that orders at the same price will be ordered by `OrderId`, with the lowest matched first.
  - The commented out function on line 166 that is kept for legacy purposes should be removed, or promoted to a `TODO`
  - The statement on line 78 should come before the statement on line 77 because it is unclear if `append` will act on a copy of `i` or a reference to it.
  - Line 224, it seems unnecessary to use a `for` loop to traverse only two orders.
  - The type `Orders` should be renamed to `AllOrders`, to prevent confusion with `Instrument.Orders`

- Though Line 294 in the `ContainsClientOrderId` function would likely perform properly due to the comparator `OrderClientIdComparator`, it is better to use the `o.GetOrder` utility function inside `ContainsClientOrderId`
- It is inconsistent for `account.Orders` to be a `treeSet` and `instrument.Orders` to be a `bTree`

- *MINOR*
  - Line 94, `(*is)[index+1:]` may go outside the bounds of the length of the `Instruments` list

- *INFO*
  - The `orderID` roughly corresponds to a sense of timestamp because the id is always increasing as orders come in one after another.
  - `SourceRemaining` is used for the purpose of not locking down `Source - SourceFilled` in the user's balance, but to let the user retain sovereignty over spending the balance as long as when it exceeds `SourceRemaining`, `SourceRemaining` will also decrease to reduce how much `Destination` is able to be bought after user's balance is spent
  - Orders are optimized for liquidity, do not touch the account balance until they are matched -- thus makers can place multiple orders based on the same `Source`.
    - When the balance of the owner account changes, `SourceRemaining` is adjusted accordingly and any un-tradable orders are canceled.

`./x/market/keeper/keeper.go`

- *MINOR**
  - There should be an emergency pause functionality for protocol administrators to disable trades while allowing users to withdraw liquidity.

- *MINOR*
  - Line 406 is problematic in that the underlying implementation for `store.Delete` in `iavl/store.go` for a `MutableTree` actually returns a boolean as a second parameter, indicating whether the removal was successful or not (perhaps the key was not found in the store).
  - Line 333, `SourceFilled.GTE(newOrder.Source.Amount)`
    - the destination amount is fixed, less than source amount of tokens will be paid if a better price exist in the market.
    - There must at least be a check for whether the replacing order's `destination > destinationFilled` of the original order, rather than checking that source of the replacing `order > sourceFilled` of the original order the check.

    - Instead of setting the replacing order's destination to be the same as that of the original order, set it to be the difference between the replacing order's destination and origin order's `destinationFilled`.

- *SEMANTIC*
  - Type-based validation from Line 128 to Line 144 can be exported upstream to the handler as `NewOrderSingle` is already too long of a function and in need of refactoring
    - The balance check on Line 142 is redundant as the case will be covered by the subsequent one that uses `GetAccountSourceDemand`
  - Line 22, comment should expand on why those gas constants specifically were chosen
  - Line 411, if instrument is not found return an appropriate error
  - There is a questionable inconsistency regarding the fact that in the slashing, market, and inflation modules `k.cdc` is used
  - In issuer and authority the codec is not stored in the keeper struct, but rather `types.ModuleCdc`
  - Move `types.EmitCancelEvent(ctx, *order)`
  - To `market/handler.go` and there, get rid of // TODO Emit events.
  - The `if` statement on line 325 related to // Verify that instrument is the same
    - Should instead be a small utility function in `market/types.go`

- *INFO*
  - EUR -> JPY may be executed by going through the passive orders JPY -> USD, USD -> EUR.
    - An important property of this functionality is that nothing from "middle asset" ("USD" in the example) must end up on the balance of the seller.
      - The "middle asset" is settled to the account at line 233
      - From a user perspective they never see the middle balance, but it can in fact be observed using the generated events.
  - To avoid any "dust" accounts we took great care to ensure that the same amount is then immediately sold.
  - Orders that contain a restricted denomination, and are not created by an allowed account, cannot become passive.
    - Line 382, only allowed addresses can add a passive order. If an account balance changes, all orders created by that account are adjusted accordingly.
  - Hook for auth keeper is added on Line 57
  - Line 147, an account cannot introduce "phantom liquidity" for the same instrument, i.e. it can not create a 20M eEUR sell wall with a 10.000 eEUR balance.
  - The rule is that the sum of all source amounts, for instrument, cannot exceed the source balance of the owner account. An instrument is uniquely identified by it's (source, destination) denoms and containing orders for that instrument. Instrument (A, B) is not the same as instrument (B, A).

  - An account owner is allowed to exhaust their account balance in multiple orders with same source, but different destination. This is intended for market makers, who might want to offer 100k EUR->USD, 100k EUR->CHF, 100k EUR->GBP with just 100k EUR on their account.
  - An account can create orders based on the same balance multiple times, eg. the same 10.000 eEUR can be part of orders against eCHF and eGBP simultaneously.
- Order matching happens sequentially, so there is always an order that is matched first (no race conditions).
  - Once that first order reduces the source account balance, any "conflicting" orders (same source denom) will be adjusted automatically as a consequence of the change to account balance.

`./x/market/module.go`

- *SEMANTIC*
  - Are there purposely no REST routes for the market? // TODO Tx routes
  - Instead of calling the `market/keeper/abci.go` one-liner via `keeper.BeginBlocker(ctx, am.k)`
    It is simpler to directly invoke the functionality therein via `am.k.initializeFromStore(ctx)`

`./x/inflation/accrual.go`

- SEMANTIC
  - Rather than simply calling returning without error, it would be more informative to at least log for provenance according to the if statement which triggered the return.
  - The following statements should happen after
    `if mintedCoins.IsZero() { return }`
    - Line 42, `k.SetState(ctx, state)`
  - `k.Logger(ctx).Info("Inflation minted coins")` should only be executed before or after the `ctx.EventManager().EmitEvent`
  - This statement `state.InflationAssets[i] = asset` must be moved inside the scope of `if minted.IsPositive(){}`
  - Although this will never be invalidated, there should be a sanity assertion (or use `SafeSub`) for line 95
  - Rename parameter `lastAccrual` of the `calculateInflation` function to `lastAccrualTime`

`./x/inflation/accrual_test.go`

- *MINOR*
  - `TestHourlyAccrual` could include an assertion inside the for loop against a pre-calculated set of the intermediate `accum` and `minted` values, in addition to simply checking the total minted at the end. The same applies for all the other tests.
  - There could be another test for `TestMultipleCoinsAccrualRandomBlockTimes`, or if not using `MultipleCoins` at least a test involving `applyInflation` and `RandomBlockTimes`.

- *SEMANTIC*
  - Line 40 in `TestRandomBlockTimes`, `annualInterest` should be renamed to be consistent with the same variable in the previous unit test (`annualInflation`).

`./x/inflation/state_test.go`

- *MINOR*
  - Rather than simply checking that the amount is less than 20, a la `require.True(t,minted.LT(sdk.NewInt(20).MulRaw(i))),` there should be an explicit delta between the actual and expected amount, and the assertion should check for an allowable discrepancy of some small % between the two.
    - Similarly a more granular delta check would be very helpful in the subsequent test's assertion `require.True(t, initialEurAmount.Amount.LT(total.AmountOf("eur")))`

- *SEMANTIC*
  - Initializing the inflation `state := types.InflationState{...}` and `keeper.SetState(ctx, state)` should not occur in `createTestComponents()` as this is explicitly done at the beginning of `TestStartTimeInFuture`, so it should in `TestModule1`
  - There should be a comment to explain why the `lastAppliedTime` is back-dated `-2400 * time.Hour` on line 131, and similarly for starting the `blockHeight` at 55

`./x/inflation/internal/types/inflation_state.go`

- *SEMANTIC*
  - In the function `ValidateInflationState`, it would be much preferable to combine the two loops into one and just have a single boolean for `if asset.Inflation.IsNegative()` to return at the end in case more than one

error occurs.

- *INFO*
  - Tokens continuously minted into the supply by the inflation module are sent to the distribution module in order to become staking rewards of their respective denominations. The module can apply inflation with any level of time precision.
    - For example, an inflation of 1% per year on EUR 1,000,000,000, each minute should add approximately 19 euro in interest.

`./x/inflation/internal/types/inflation_state_test.go`

- *SEMANTIC*
  - In `TestValidation`, the two errors should be decoupled from each other and the appropriate error string should be tested for.
  - Line 43 is redundant because `SetInflation` is tested in `x/inflation/state_test.go`

`./x/inflation/internal/keeper/keeper.go`

- *SEMANTIC*
  - Line 60, remove TODO.
  - Line 98, only set inflation state if it actually changes within the for loop.
  - Line 74, in `SetInflation` check that `newInflation` is greater or equal to 0.

`./x/inflation/internal/types/expected_keepers.go`

- *SEMANTIC*
  - `SendCoinsFromModuleToAccount(ctx sdk.Context, senderModule string, recipientAddr sdk.AccAddress, amt sdk.Coins) sdk.Error.`
  - `SendCoinsFromModuleToModule(ctx sdk.Context, senderModule, recipientModule string, amt sdk.Coins) sdk.Error`
    - `senderModule` has no string type, although it does in the above function.

`./x/inflation/module.go`

- *MINOR*
  - We suggest registering an invariant similar to the one below (but related to the minimum block rewards):
    - // TotalSupply checks that the total supply reflects all the coins held in accounts.

`./x/inflation/alias.go`

- *SEMANTIC*
  - `QueryInflation` is unused in the whole project.

As validators will want to avoid slashing in relation to planned maintenance, they can remove themselves voluntarily and temporarily from the validator set by submitting a "entering maintenance mode" transaction. Double-signing leads to slashing subject to parameters and tombstoning of the validator.
In the slashing module, the following slashing methods' readability needs to be improved:

`./x/slashing/client/cli/query.go`

- GetCmdSigningInfo, Lines 44 to 78.
- GetCmdQueryParams, Lines 81 to 104.

`./x/slashing/client/cli/tx.go`

- GetCmdUnjail, Lines 37 to 56.

`./x/slashing/keeper.go`

- INFO
  - Line 162, jailing is now time-based, rather than block-based, `sign.Info.JailedUntil()` sets jail time by taking the time dot time argument. Down-time slashing parameters are set aggressively to ensure continued availability within short timeframes.
  - Line 237, records of missed blocks are now kept as timestamps, with the help of:
    - `getMissingBlocksForValidator()`
    - `setMissingBlocksForValidator()`

- Proceeds from slashing activities are distributed to validators in the validator set:
  - Line 273, the `handlePendingPenalties()` function gets the values for pending penalties, gets the set of validators to pay and checks if the validators are part of the penalties set, pays them accordingly, and finally updates the pending penalties set.
  - Line 347, the `slashValidator()` function calculates the slashing amount, mints the coins, then sends the coins from module to module and updates the pending penalties for execution.

`./x/slashing/client/rest/tx.go`

- *MINOR*
  - Include validation and error detection with `k.logger(ctx).Info()` for the case of inappropriate input `valAddr`.

`./x/authority/keeper/keeper.go`

- *SEMANTIC*
  - `AddMintedCoins` definitely needs a better name, to reflect that it actually pays fees to the `feeCollector` from the supply module's balance
  - Lines 17-21, it seems these keys belong in the module's `keys.go`.
  - Line 125, before panicking it's better to log `types.ErrNotAuthority` visibility/monitoring.

`./x/authority/keeper/keeper_test.go`

- *MINOR*
  - Line 237, the issuer keeper should be using `keyIssuer` (which happens to be the same as issuer `types.StoreKey`)

`./x/issuer/handler.go`

- *SEMANTIC*
  - Rather than provision for the creation of liquidity provider implicitly via `handleMsgIncreaseMintableAmount`, it is better style to provide an explicit message, handler, and keeper method for this logic.

<u>`./x/issuer/types/types.go`</u>

- *SEMANTIC*
  - If the issuer was not implemented as an account on purpose because there is no notion of a spendable balance or ability to transfer:
    - then there must at least be a utility function for keeping track of the total allowances granted by an issuer to all of his liquidity providers;
  - Unlike the liquidity provider has a spendable balance, this is what they had minted so far from their allowance.

<u>`./x/issuer/keeper/keeper.go`</u>

- *MINOR*
  - Lines 57-60 in `IncreaseMintableAmount` of `LiquidityProvider`, there is no `!res.IsOK()` therefore an error will not properly return if `lpKeeper.SetLiquidityProviderAccount` fails (although this is unlikely).
  - Line 181, `else` returns error saying Issuer already exists.
  - For `IncreaseMintable`, `DecreaseMintable`, and `RevokeLiquidityProvider` the erroneous return value of `lpKeeper.GetLiquidityProviderAccount` is nil in any case, and thus ambiguous as to whether there was no account found or if the found account was not a LiquidityProvider.
  - Line 213 in anyContained (move function to `denom_utils.go` and rename function to something better like `containsDenom`) `s[index] == st` is a redundant check.
  - Line 272, it's safe to break out of the outer for loop after the inner one has executed without returning.

- *SEMANTIC*
  - Instead of using `keyIssuerList`, use `types.`*`QueryIssuers.`*
  - A great const would be be `logger := k.logger(ctx)` instead of re-declaring references to it in every method.
  - Instead of importing the liquidity provider, its interaction with the issuer keeper should be defined in issuer's expected_keepers.go just as the InflationKeeper is.
  - Rather than invoke an `ErrDenominationAlreadyAssigned`, it's better to prune those denominations already assigned and only return an error if there are no new `denoms` at all.
    - Otherwise refrain from this implicit adding of `denoms` at all and create a separate method for it, then the error is perfect.
  - Line 172, adding `Denoms` to an existing issuer should be an explicit function rather than implicitly co-residing in AddIssuer.
    - Moreover, since the issuer keeper already has a reference to the inflation keeper, why not use the `InflationState.InflationAssets` instead of `collectDenoms()`?
  - Move Lines 85-88 to `DecreaseMintableAmount`, otherwise the `SafeSub` operation is redundant.
  - Line 191 in `RemoveIssuer`, use the type alias for `[]types.Issuer` defined in this module's types.go.

- Line 243 `if address == nil` seems to be a redundant check because by the time we enter any function that invokes this function, the validation of the address input had already been performed in base app, prior to the message even reaching the handler.

`./x/issuer/keeper/keeper_test.go`

- *MINOR*
  - Line 45, try calling `NewIssuer` with an empty `Denoms` parameter.
  - Line 294, the issuer keeper needs to be initialized with the keyIssuer `StoreKey` not the `keySupply` one.
  - Line 121, `require.NotNil(t, result)` is not the best way to determine the error is firing properly for decreasing `mintable` by too much.
    - Line 124, it's redundant to get another pointer reference to the account.
  - Line 189, in `TestDoubleLiquidityProvider` also test the failure of decrease `mintable` balance prior to increasing with issuer2.
  - There is lack of coverage for the following case in `TestIssuerModifyLiquidityProvider`.
    - The issuer should not be able to decrease the `mintable` amount of the liquidity provider for a `denom` the issuer does not control.
  - There is a lack of coverage for the following case in `TestAddAndRevokeLiquidityProvider`.
    - Decreasing `mintable` balance with a random account as the issuer should also fail against the liquidity provider.

`./x/issuer/module.go`

- *MINOR*
  - ValidateGenesis must be implemented, including the functionality that there are no duplicate `Denoms`.

- *SEMANTIC*
  - As we recommend against implicit syntax because it is bad for readability, this go idiom should be removed from the last line in the `InitGenesis` method, and changed to simply return like the standard implementation does: `return []abci.ValidatorUpdate{}`.
  - Why is there a module-wide "`ModuleCdc`" in slashing, inflation, authority, but not in LP?

`./x/issuer/genesis.go`

- *MINOR*
  - In `initGenesis`, there should be a check for isValid before adding an issuer.

`./x/issuer/types/keys.go`

- *MINOR*
  - `ExportGenesis` returns `defaultGenesisState`.
  - Should be:

    -
      ```
      gs := ExportGenesis(ctx, am.keeper)
      return ModuleCdc.MustMarshalJSON(gs)
      ```

`./x/liquidityprovider/types/account.go`

- *SEMANTIC*
  - Define a type alias for `[]types.LiquidityProviderAccount`.
  - The String() function will panic `if acc.GetPubKey == nil`.
    - For consistency with returning the error “mineable amount cannot be negative”, implement an else return error after this `if acc.GetPubKey() != nil.`
  - To comply with the semantics in which a `NewIssuer` is instantiated, we recommend to accept an address as a parameter for `NewLiquidityProviderAccount` and instantiate an account inside the function rather than beforehand.

`./x/liquidityprovider/types/account_test.go`

- *MINOR*
  - There should be a final assertion in the `TestBasic` unit making sure that the `Mintable.AmountOf` and other demons for the liquidity provider did not change.
  - There must be a panic test in `TestDecreaseMintable` for an input where there are `denoms` that the liquidity provider doesn’t have a `mintable` amount for.

`./x/liquidityprovider/types/msgs_test.go`

- *MINOR*
  - A negative amount should be invalid.

`./x/liquidityprovider/types/errors.go`

- *MINOR*
  - To match the "account does not exist" error, there should be an "account already exists" error.
    - There must also be an "allowance exhausted" error.

`./x/liquidityprovider/keeper/keeper_test.go`

- *SEMANTIC*
  - Line 60, redundant reference to account.
  - There should be a `TestMintTooMuchMultipleDenoms`.

`./x/liquidityprovider/genesis.go`

- *SEMANTIC*
  - There should be a genesis state with `[]types.LiquidityProviderAccount`, as there is with issuers, and validation implemented accordingly.

`./x/liquidityprovider/module.go`

- *SEMANTIC*
  - Why is there a module-wide "`ModuleCdc`" in slashing, inflation, authority, but not in LP?

- *MINOR*
  - There is no (but there should be) querier and therefore there is no return for `cli.GetQueryCmd(types.StoreKey, cdc)`.

`./hooks/auth/proxykeeper.go`

- *INFO*
  - Contains a proxy struct that enables listening to events from the Auth-keeper, notably changes to account balances.
  - Only the `SetAccount` method of the standard account keeper is meaningfully overridden to notify an array of subscribed listener functions

- *MINOR**
  - Authority modules do not put their types and keepers into "internal" sub-packages. This is an implicit form of access control which, combined with defining more hooks, would be more appropriate to ensure that the code inside the authority module is open for extension and closed for modification.

`./x/authority/genesis.go`

- *INFO*
  - Genesis state is preloaded with the authority key, a list of `restrictedDenoms`, and the `minGasPrices`

`./x/authority/keeper/querier.go`

- *SEMANTIC*
  - Line 35, return an error related to json Marshalling rather than a generic runtime error.

`./x/authority/module.go`

- *INFO*
  - This module controls gas prices and manages issuers. Gas prices are pre-programmed and constant, and do not correspond to the actual activity happening
    - e.g. an aggressive order in the market module can match any number of passive orders without increasing the required fee.

- *SEMANTIC*
  - As we recommend against implicit syntax because it is bad for readability, this go idiom should be removed from the last line in the `InitGenesis` method, and changed to simply return like the standard implementation does `return []abci.ValidatorUpdate{}`

`./hooks/distribution/abci.go`

- *SEMANTIC*
  - Rather than localizing
    `var ( previousProposerKey = []byte("emdistr/previousproposer") )`
    - And then passing in the entire `db.DB` as a parameter to the `BeginBlock.`
    - It's better to isolate all such constants into a dedicated file, which would enable to perform `db.Get(previousProposerKey)` prior to entering this module's `BeginBlocker` and instead pass that value in as a parameter.
  - Instead of setting the `lastProposer` the way it is done in the standard implementation, using the AppState via `k.SetPreviousProposerConsAddr(ctx, consAddr),` e-Money sets it to the current transaction batch using `batch.Set(previousProposerKey, previousProposer)`
    - This change was made to avoid triggering block creation based on that specific change in the application state.
  - Instead of using the distribution keeper's store, whose use by other modules may be granularly controlled, a more global scope is used.

- *INFO*
  - Rewards are distributed only if transactions were processed. We do not wish to generate a new block to record that a validator failed to sign the previous block, so "heart-beat" blocks that are created in 1 minute intervals. If a transaction is present in the mempool, a block will be created immediately to finalize it.
    - This is achieved by checking the amount collected by `auth.FeeCollector`.
      - This encourages driving volume to the system.

`./hooks/bank/proxykeeper.go`

- *INFO*
  - The bank module is registered in `app.go` using the `Wrap` function defined in this file, taking as parameters the bank keeper and the authority keeper.
    - Wrap returns a `ProxyKeeper`, which defines a wrapper struct that encapsulates the standard bank keeper and `RestrictedKeeper` — just a wrapper interface for the getter function that loads RestrictedDenoms, which cannot be freely traded or sent among users.
  - Per inclusion in the `RestrictedDenom`'s `Allowed` array, if either sender or recipient is present the `IsAnyAllowed` utility function will evaluate to true.
  - Of all the bank module's `BaseKeeper` methods, the only ones being overridden are `InputOutputCoins` and `SendCoins` and the only change in functionality is to throw an error if a certain movement of `RestrictedDenoms` is detected.

- `inputOutputCoins` transfers coins from any number of input accounts to any number of output accounts.
  - No restricted denoms may be in the inputs parameter to this method

`./hooks/bank/bankproxy_test.go`

- *MINOR*
  - For completeness, `TestProxySendCoins` and `TestInputOutputCoins` should also cover
    - The scenarios where the destination and sender account are the same entity
    - The amount to send exceeds the available balance of the sender
    - There is no input denomination to match an output denomination

- *SEMANTIC*
  - Should move this code to `createTestComponents` function, line 163 as an in-line declaration there
    - `bk.rk = restrictedKeeper {...}`

`./app.go`

- *INFO*
  - When we initialize the `distrKeeper` and the `bankKeeper`, the last parameter passed in is an `accountBlacklist`
    - The usual case for initializing this Cosmos module is to pass in as the last parameter a mapping of all the app's module account addresses.

- *SEMANTIC*
  - In the import section, “github.com/cosmos/cosmos-sdk/x/genaccounts"
    - As of *PR #5017,* 08/19/19, and as reflected in the latest Cosmos release, 0.38, the x/genaccounts module has been deprecated
  - At the end of the `NewApp` constructor, instead of using `tmos.Exit(err.Error())` in order to avoid importing the entire `tmos` package, a simple panic is used
    - Although it bares little functional difference, to copy the literal implementation of `tmos.Exit` is suggested for semantic consistency and because panics are actually intended for when the program, or its part, has reached an unrecoverable state.
    - `os.Exit` is used when you need to abort the program *immediately,* with no possibility of recovery or running a deferred clean-up statement

./cmd/emd/testnetCmd.go

- *MINOR*
  - The function `getAuthorityKey` starts with the code
    ```
    key, err := sdk.AccAddressFromBech32(param)
    if err == nil { return key }
    ```
    - The code does not check if the err value is not nil because it so. The code needs to first check the case of an error and handle that, and then re-check the error if it is nil and return the key.
    - Change to
      ```
      key, err := sdk.AccAddressFromBech32(param)
      if err != nil {
       panic(err) // we cant return the error value.
      } else return key
      ```

- *SEMANTIC*
  - The function `testnetCmd` lacks readability, and should be refactored to be more readable, lines 56 to 68.
    - If it contains legacy code, this should be grouped and notated, lines 93 to 103.

./capacity_test.go

- *MAJOR*
  - Summarizing 1 Failure:
    ```
    [Fail] Staking Blocks can hold many transactions  [It] Creates a lot of send transactions
      github.com/e-money/em-ledger/capacity_test.go:118
      Ran 25 of 25 Specs in 285.920 seconds
      FAIL! -- 24 Passed | 1 Failed | 0 Pending | 0 Skipped
      --- FAIL: TestSuite (285.92s)
    ```

./authority_test.go

- *MINOR*
  - There is lack of a test where a LiquidityProvider is created with a mintable denom which the invoking issuer does not control.
  - Errors should be recorded for “issuer gets revoked” and "former liquidity provider attempts to mint"