CERTIK

Audit Report

Produced by CertiK

for Reserve

Oct 9th, 2019

CertiK Audit Report For Reserve

Request Date: 2019-09-27
Revision Date: 2019-10-09
Platform Name: Ethereum

CERTIK

Contents

Disclaimer

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 Verification Services Agreement between CertiK and Reserve(the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, 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 without CertiK's prior written consent.

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, in partnership with grants from IBM and the Ethereum Foundation, has developed a proprietary Formal Verification technology to apply rigorous and complete mathematical reasoning against code. This process ensures algorithms, protocols, and business functionalities are secured and working as intended across all platforms.

CertiK differs from traditional testing approaches by employing Formal Verification to mathematically prove blockchain ecosystem and smart contracts are hacker-resistant and bug-free. CertiK uses this industry-leading technology together with standardized test suites, static analysis, and expert manual review to create a full-stack solution for our partners across the blockchain world to secure 6.2B in assets.

For more information: https://certik.org/

Executive Summary

This report has been prepared for Reserve to discover issues and vulnerabilities in the source code of their Ownable, ReserveEternalStorage, Reserve, Vault, Basket, Proposal and Manager smart contracts. A comprehensive examination has been performed, utilizing CertiK's Formal Verification Platform, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practice and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line by line manual review of the entire codebase by industry experts.

Vulnerability Classification

CertiK categorizes issues into 3 buckets based on overall risk levels:

Critical

The code implementation does not match the specification, or it could result in the loss of funds for contract owner or users.

Medium

The code implementation does not match the specification under certain conditions, or it could affect the security standard by lost of access control.

Low

The code implementation does not follow best practices, or use suboptimal design patterns, which may lead to security vulnerabilies further down the line.

Testing Summary

Type of Issues

CertiK smart label engine applied 100% formal verification coverage on the source code. Our team of engineers ao scanned the source code using our proprietary static analysis tools and code-review methodologies. The following technical issues were found:

Title	Description	Issues	SWC ID
Integer Overflow and Underflow	An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function incorrectness	Function implementation does not meet the specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker is able to write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by minors to some degree.	0	SWC-116
Insecure Compiler Version	Using an fixed outdated compiler version or floating pragma can be problematic, if there are publicly disclosed bugs and issues that affect the current compiler version used.	0	SWC-102 SWC-103
Insecure Randomness	Block attributes are insecure to generate random numbers, as they can be influenced by minors to some degree.	0	SWC-120

"tx.origin" for authorization	tx.origin should not be used for authorization. Use msg.sender instead.	0	SWC-115
Delegatecall to Untrusted Callee	Calling into untrusted contracts is very dangerous, the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Functions are public by default. A malicious user is able to make unauthorized or unintended state changes if a developer forgot to set the visibility.	0	SWC-100
Uninitialized variables	Uninitialized local storage variables can point to other unexpected storage variables in the contract.	0	SWC-109
Assertion Failure	The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.	0	SWC-110
Deprecated Solidity Features	Several functions and operators in Solidity are deprecated and should not be used as best practice.	0	SWC-111
Unused variables	Unused variables reduce code quality	0	

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

No issue found.

Review Notes

Source Code SHA-256 Checksum

See commit $9c6a64227fa5c88f87520f87d8dd4a8109412600$.

- **Basket.sol**
  `8714ce2e3d230cce9da51a16adb35864d7f36dde0d2e226d748e97f3df288c6e`
- **Manager.sol**
  `50bc1e958bddc428dddb8a07138a0b1e02864df719439410e5b93154c3654f2c`
- **Proposal.sol**
  `9d5b8729600d997d3d8f1c802f6773cff3b4d0df9f411076bf23a6305593c8bb`
- **Vault.sol**
  `09493fc98ce1c8150d226f1cd554ba073619bbef23d94a0d3c85efad9afbb170`
- **Ownable.sol**
  `711662f0791cbb98a02a594455a4588eb9cd2b25c9164bc00864e5a396714806`
- **IRSV.sol**
  `8029cc9de2c7e834fa062f08d8eb4bd52e16dc4a8daf3f7dcb1909ae2a42a60e`
- **Reserve.sol**
  `036c2da9cf01ca15ad87f05ba1b5df3bc75a7c8269819fdf9aeffc1f6165e49c`
- **ReserveEternalStorage.sol**
  `5e2ddb454f9ef003ed17033d6a45f39070d349e0d640a82d28654d108dd3999f`

Summary

CertiK was chosen by Reserve to audit the design and implementation of its soon to be released `RSV` smart contracts. To ensure comprehensive protection,the source code has been analyzed by the proprietary CertiK formal verification engine and manually reviewed by our smart contract experts and engineers. That end-to-end process ensures proof of stability as well as a hands-on, engineering-focused process to close potential loopholes and recommend design changes in accordance with the best practices in the space.

The client has demonstrated their professional and knowledgeable understanding of the project, by having:

1. A production ready repository with high-quality source code.
2. Unit tests covering the majority of its business scenarios.
3. Accessible, clean, and accurate readme documents for intentions, functionalities, and responsibilities of the smart contracts.

System Description

The `RSV` stablecoin is backed by a `Basket` of stablecoins such as USDC and Tether or other tokenized assets. It starts with a centralized stage and will gradually be turned into DAO as the final stage. The stablecoin itself is a standard ERC-20 token and is attached to the specific basket by the `Manager` contract. The backing stablecoins in the basket are stored at the address of a `Vault` contract associated with the `Manager`. Issuance of new `RSV` requires specific amounts of the stabletokens specified by the `Basket` to be transferred from the requester's account to the `Vault` account. Redemption of `RSV` will result in the amounts of stabletokens responding to the current `Basket` being transferred from the `Vault` back to the requester's account.

Proposals can be submitted to the `Manager` contract for changing the types and ratios of the stablecoins in the basket. Once a proposal is approved by the administrator, it can be further executed by the administrator. Excess amounts of the stablecoins after the proposal takes effect will be transferred back from the `Vault` account to the proposer's account, while insufficient amounts of the stablecoins will be transferred from the proposer's account to the `Vault` account individually.

The `RSV` project architecture is shown in figure 1.

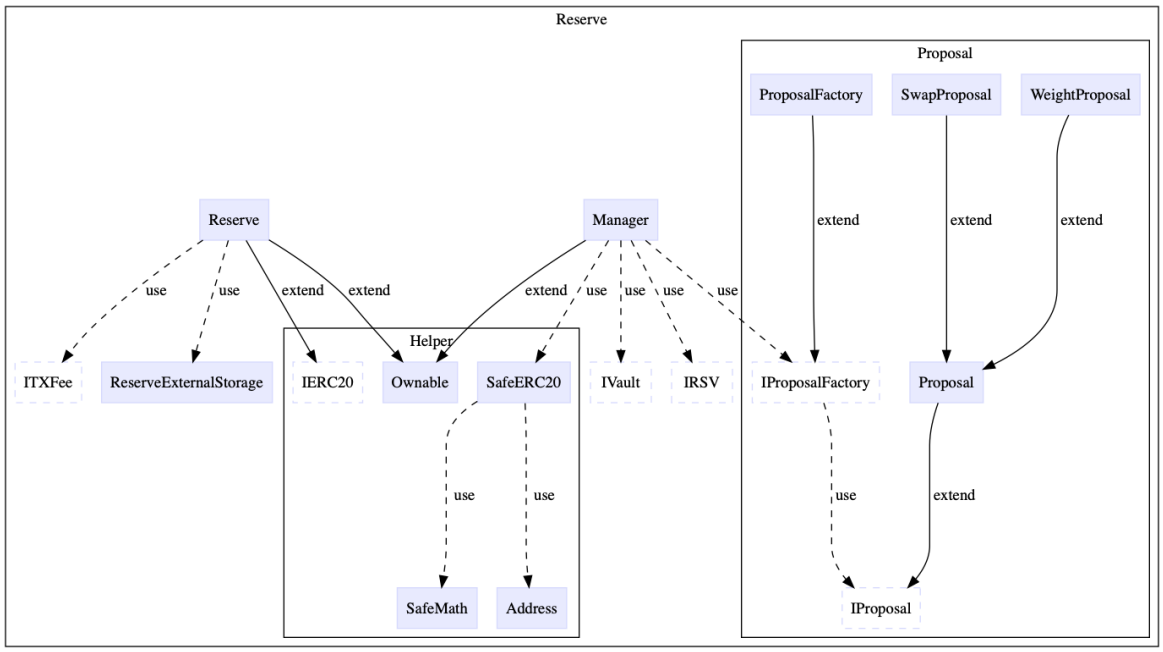

Figure 1: Reserve `RSV` Architecture

- **ReserveEternalStorage**: Storage of the `RSV` token contract, standard upgradeability pattern.
- **Reserve**: Contract for the ERC20 based `RSV` token.
- **Vault**: This is the account in which the tokens in the RSV basket will be held to be exchanged for stablecoins. The only non-trivial logic in here is logic to set the `Manager` of the `Vault`. Presumably these contracts are separate so that in the future `Manager` can be upgraded to a decentralized solution.

- **Basket**: This contract describes the different ratios of stablecoins in the `basket` that RSV represents. It is essentially READ ONLY and is only more complex than a struct because it has logic in the constructor.
- **Proposal**: Each instance of this contract represents one proposal for adjusting the "basket" that the `Manager` can either approve or deny.
- **Manager**: The Manager contract has all the core logic for the system. It handles role assignment, deposits/withdrawals, creating proposals, and executing proposals.

Weight Calculation

Unit Specification

The weight of the stablecoins in the basket is implemented with a special unit system, which requires special notification here.

As being documented in the source repository, there are three level of units used in the system. Assume the stablecoin is named `Token`:

- `Token` : 1 Token
- `qToken` : A quantum of the Token. 1 Token = 1 qToken $\times 10^{\text{decimals of the Token}}$. For example, 1ETH = 10^{18}qETH.
- `aqToken` : An atto-quantum of the Token, which is a portion of the quantum unit. 1 qToken = 10^{18} aqToken. For example, 1ETH = 10^{36}aqETH.

The following illustrations are taken from Reserve's documentation in the source code:

- `1 RSV`: 1 Reserve
- `1 qRSV`: 1 quantum of Reserve. `RSV` & `qRSV` are convertible by `.mul(10**reserve.decimals()qRSV/RSV))`.
- `1 qToken`: 1 quantum of an external Token.
- `1 aqToken`: 1 atto-quantum of an external Token. `qToken` and `aqToken` are convertible by `.mul(10**18 aqToken/qToken)`.

The weight of the stablecoins in the basket is in `aqToken/RSV` where the `Token` refers to the specific stablecoin. Suppose the decimal of the `Token` is $_{\text{Token}}$ (e.g. 6) and the amount of `Token` required by the basket is `y` (in `Token/RSV`), then the weight for the `Token` in the basket is specified as $y \cdot 10^{\text{Token}} \cdot 10^{18}$.

Conversion from `RSV` to `Token`

Suppose the weight of a stablecoin `Token` in the basket is w (in unit of `aqToken/RSV`), the decimal for `RSV` is RSV (which is 18 by default). If there is x amount of `RSV` (in unit of `qRSV`), then the corresponding amount of `Token` required by the basket is:

$$
\begin{aligned}
x \cdot w \cdot \frac{1}{10^{18} \cdot 10^{RSV}} &= \frac{x\ (\text{qRSV}) \cdot w\ (\text{aqToken/RSV})}{10^{18}\ (\text{aqToken/qToken}) \cdot 10^{RSV}\ (\text{qRSV/RSV})} \\
&= \frac{x\ (\text{qRSV})}{10^{RSV}\ (\text{qRSV/RSV})} \cdot \frac{w\ (\text{aqToken/RSV})}{10^{18}\ (\text{aqToken/qToken})} \\
&= \frac{x}{10^{RSV}}(\text{RSV}) \cdot \frac{w}{10^{18}}(\text{qToken/RSV}) \\
&= \frac{x \cdot w}{10^{18+\text{RSV}}}(\text{qToken})
\end{aligned}
$$

Collateralization Check

Suppose the current total supply of the `RSV` is `a` (in `qRSV`), the weight of a stablecoin `Token` in the basket is w (in `aqToken/RSV`), the balance of the backing `Token` in `Vault` is b (in `qToken`), and the decimal for `RSV` is $_{\text{RSV}}$ (which is 18 by default). Then the check is performed as:

$$
\begin{aligned}
a \cdot w > b \cdot (10^{18} \cdot 10^{\text{RSV}}) &\Leftrightarrow \frac{a\ (\text{qRSV}) \cdot w\ (\text{aqToken/RSV})}{10^{18}\ (\text{aqToken/qToken}) \cdot 10^{\text{RSV}}\ (\text{qRSV/RSV})} > b\ (\text{qToken}) \\
&\Leftrightarrow \frac{a}{10^{\text{RSV}}}(\text{RSV}) \cdot \frac{w}{10^{\text{RSV}}}(\text{qToken/RSV}) > b\ (\text{qToken}) \\
&\Leftrightarrow \frac{a \cdot w}{10^{\text{RSV}+18}}(\text{qToken}) > b\ (\text{qToken})
\end{aligned}
$$

Source of Truth

CertiK uses the following documentations as reference to better understand the system:

- Reserve Whitepaper [1]
- Reserve RSV Design Documentation [2]
- Reserve RSV Source Contracts [3]

Recommendations

Items in this section are low impact to the overall aspects of the smart contracts, thus will let client to decide whether to have those reflected in the final deployed version of source codes. Items are labeled CRITICAL, MAJOR, MINOR, INFO, DISCUSSION (in decreasing significance).

Second Version commit $e29c38939599fddc38fccf40b63b730a8fadf76c$

Ownable.sol

- INFO `acceptOwnership()`: Redundant `require(_nominatedOwner != address(0))` check.
  - (Reserve - Resolved) Changed.

[1] https://reserve.org/whitepaper.pdf
[2] https://github.com/reserve-protocol/rsv-v2/tree/production/design-docs
[3] https://github.com/reserve-protocol/rsv-v2/tree/production

ReserveEternalStorage.sol

- INFO `updateReserveAddress(newReserveAddress)`: Recommend checking that the `newReserveAddress` is not zero address.
  - (Reserve - Resolved) Changed.

Reserve.sol

- INFO `transferEternalStorage(newReserveAddress)`: Recommend checking that the `newReserveAddress` is not zero address.
  - (Reserve - Resolved) Changed.
- INFO `transferEternalStorage(newReserveAddress)`: Recommend emitting event to keep additional record in the original contract.
  - (Reserve - Resolved) Changed.
- INFO `changeTxFeeHelper(newReserveAddress)`: Recommend emitting event to keep additional record in the original contract.
  - (Reserve - Confirmed) No change.
- INFO `_transfer(from, to, value)`: Recommend adding `require(to != address(0), ``...")`.
  - (Reserve - Resolved) Changed.
- DISCUSSION `_transfer(from, to, value)`: Expression `(fee >= 0)` can be removed since the type is `uint256`.
  - (Reserve - Resolved) Changed.

Vault.sol

- DISCUSSION `withdrawTo(token, amount, to)`: The `require(_msgSender()== manager )` check could be wrapped as modifier as similar to `onlyOwner`.
  - (Reserve - Resolved) Changed.

Basket.sol

- MINOR `constructor(trstedPrev, _tokens, _weights), size()`: Please replace `uint` with its canonical name `uint256`.
  - (Reserve - Resolved) Changed.

Proposal.sol

- MINOR `_newBasket(trustedRSV, trustedOldBasket)`: Please replace `uint` with its canonical name `uint256`.
  - (Reserve - Resolved) Changed.

Manager.sol

- MINOR `vaultCollateralized()`: The `require()` check should be performed after function execution `_`.
  - (Reserve - Resolved) Changed.
- MINOR `isFullyCollateralized()`, `toIssue(rsvAmount)`, `toRedeem(rsvAmount)`, `issue(rsvAmount)`, `redeem(rsvAmount)`, `executeProposal(id)`: Please replace `uint` with its canonical name `uint256`.
  - (Reserve - Resolved) Changed.
- INFO `cancelProposal()`: Recommend adding the `require(proposalsLength > id, ...)` check.
  - (Reserve - Resolved) Changed.
- INFO `_weighted()`: The `amount >= 0 \&\& weight >= 0` check can be removed.
  - (Reserve - Resolved) Changed.
- INFO `_weighted()`: The `decimalsDivisor` in comment should be updated to `scaleFactor`.
  - (Reserve - Resolved) Changed.
- DISCUSSION `toIssue(rsvAmount)`, `toRedeem(rsvAmount)`: Shall these functions be kept as `internal`?
  - (Reserve - Confirmed) No Change.
- DISCUSSION `constructor()`: Consider adding additional check for `seigniorage` to ensure that the input is in correct range.
  - (Reserve - Resolved) Changed.

First Version commit 94a471353fa71adcb29c50d0e46e0191e303fc53

Ownable.sol

- DISCUSSION `_owner`: Recommend changing the visibility back to `private` to be consistent with the OpenZeppelin version and provide an external function for access.
  - (Reserve - Resolved) Changed.

ReserveEternalStorage.sol

- MINOR The `ReserveEternalStorage` contract can be inheriting from `Ownable` contract.
  - (Reserve - Resolved) Changed.
- MINOR `transferOwnership(newOwner)`, `transferEscapeHatch(newEscapeHatch)`: Consider using the pull-over-push pattern for ownership transfer.
  - (Reserve - Resolved) Changed.
- MINOR `transferOwnership(newOwner)`, `transferEscapeHatch(newEscapeHatch)`: Recommend checking the given new proposed address is not zero address.
  - (Reserve - Resolved) Changed.
- INFO Consider emitting events for monitoring contract activity.
  - (Reserve - Confirmed) No change for now.
- INFO Recommend defining the state variables at the top of the contract, according to the Solidity developer guide.
  - (Reserve - Confirmed) No change for now.
- DISCUSSION What would be the reason for removing the account frozen functionality? (Consider this contract is less lean to be upgraded)
  - (Reserve - Confirmed) Our business plans changed to not include freezing.

Reserve.sol

- DISCUSSION `transferEternalStorage()`: Recommend wrapping `paused` requirement as modifier to be consistent with `notPaused()`. Recommend providing error message for `require()` as well.
  - (Reserve - Resolved) Changed.
- DISCUSSION `changeMinter(newMinter)`, `changePauser(newPauser)`, `changeFeeRecipient(newFeeRecipient)`: The pull-over-push pattern can be used for role transfer.
  - (Reserve - Confirmed) No change, reason is adds code to the contracts that need to accept these roles
- MINOR `changeMaxSupply()`: Recommend adding a `require(maxSupply >= totalSupply, ...)` check.
  - (Reserve - Confirmed) No change, we want to be able to set `maxSupply` below current supply. When this happens, desired functionality is mint always reverts until the supply falls below that point, while burn can still happen.

- INFO `data()`: Can be renamed as `trustedData`.
  - (Reserve - Confirmed) We will consider the trusted pattern more and determine where to make this change.

Basket.sol

- DISCUSSION `constructor()`: What is the design consideration for limiting tokens change in each function call to 100? The total amount of tokens in the basket can still exceed 100. When copying from the previous bucket to the new one, `.push()` could exceed the length of 100.
  - (Reserve - Resolved) Changed.
- DISCUSSION Intuitively (or common sense), the percentage of each stable coin should be summed to 100%, but it seems there is no such check at smart contract level. By doing so it requires to maintain a list of stable coin precision constants which would introduce extra complixity, so we leave to Reserve team to consider if this is intended.
  - (Reserve - Confirmed) No change. Price information for the token is not in scope, and we also do not want to assume all tokens are stabletokens.

Proposal.sol

- INFO `accept()`: Recommend adding check for `_time` such as `require(_time > now, ...)`.
  - (Reserve - Confirmed) No change
- INFO Consider adding event logs for monitoring the contract activity or state chanced.
  - (Reserve - Confirmed) May change after more consideration.
- DISCUSSION Is the `proposer` in `Proposal` defined only for logging purpose?
  - (Reserve - Confirmed) No, proposer is necessary to understand which account to perform withdrawals from.
- DISCUSSION `constructor()`: Is there a size requirement such as `require(_tokens.length > 0, ...)`?
  - (Reserve - Resoled) Changed to contain size requirement.
- DISCUSSION `SwapProposal`: Recommend adding a few more comments or documents to emphasize that the "quantities of tokens to transfer in total" shall be closed related to the current total supply of RSV.
  - (Reserve - Resolved) Changed.

Vault.sol

- DISCUSSION The pull-over-push pattern can be used for role transfer.
  - (Reserve - Confirmed) No change.
- INFO `changeManager()` : Consider adding the error message along with the `require()`.
  - (Reserve - Resolved) Changed.
- DISCUSSION `withdrawTo()`: Recommend emitting event message.
  - (Reserve - Resolved) Changed.

Manager.sol

- INFO `address operator`: Variable visibility unspecified.
  - (Reserve - Resolved) Changed to public.
- MINOR `constructor()`: Initial operator unspecified.
  - (Reserve - Resolved) Changed initial operator to `msg.sender`.
- INFO According the best-code-practice guide, the `if` statement on line 155 should have braces for its body
  - (Reserve - Resolved) Changed.
- DISCUSSION `acceptProposal()` Consider using `SafeMath` add operation as a replacement for `now + delay` to align with other arithmetic operations.
  - (Reserve - Resolved) Changed.
- DISCUSSION `isFullyCollateralized()`: Recommend unifying the names `scaleFactor` `decimalsDivisor`(in `_weighted`), and `divisor`(in `_newBasket`).
  - (Reserve - Resolved) Changed.
- INFO `isFullyCollateralized()`: Typo in comment `// unit: qRSV`. Should be `// unit : qToken`.
  - (Reserve - Resolved) Re-check and change if correct.
- INFO `isFullyCollateralized()`: The function can be wrapped as a modifier.
  - (Reserve - Resolved) Changed. Plan is to keep a public view and wrap it with a modifier.
- DISCUSSION `_executeBasketShift()`: The parameter `Basket oldBasket` and `Basket newBasket` can be changed to `uint256 oldWeight` and `uint256 newWeight`.

  - (Reserve - Resolved) Changed.

- DISCUSSION `proposeSwap`, `proposeWeights`, `acceptProposal`, `cancelProposal`, `executeProposal` : Shall pause control be added to these actions?
  - (Reserve - Resolved) Changed. Will change pausing design to the following: `pauseIssuance` and `pauseAll`. The former only pauses issue, while the latter pauses all functions, including the 5 proposal functions.

- DISCUSSION Consider distributing the responsibility of `Manager` into different roles: 1) Paused/UnPaused; 2) Operator Role Assignment; 3) Seigniorage Role Assignment; 4) RSV Issuance/Redemption.
  - (Reserve - Confirmed) No change.

- INFO `_weighted()`: Typo in `require(..., ``Weigh negative amounts")`.
  - (Reserve - Resolved) Changed.

Best practice

Smart contract development requires a particular engineering mindset. A failure in the initial construction can be catastrophic, and fixing the project after a vulnerability is exploited can be exceedingly difficult.

To ensure the success of the project and to avoid the common pitfalls, smart contracts should adhere to best practices from the beginning. Below is a checklist of key points & vulnerability vectors that helps to indicate a high overall quality of the project.

(✓ indicates satisfaction; × indicates unsatisfaction; – indicates inapplicable)

General

Compiling

✓ Correct environment settings, e.g. compiler version, test framework

✓ No compiler warnings

Logging

× Provide error message along with `assert` & `require`

× Use events to monitor contract activities

Code Layout

✓ According to Solidity Developer Guide, Layout contract elements should following below order:

1. Pragma statements

2. Import statements
3. Interfaces
4. Libraries
5. Contracts

× Each contract, library or interface should following below order:

1. Type declarations
2. State variables
3. Events
4. Functions

× According to Solidity Developer Guide, functions should be grouped according to their visibility and ordered:

1. constructor
2. fallback function (if exists)
3. external
4. public
5. internal
6. private

Arithmetic Vulnerability

EVM specifies fixed-size data types for integers, in which means that has only a certain range of numbers it can store or represent.

Two's Complement / Integer underflow / overflow

✓ Use Math library as SafeMath for all arithmetic operations to handle integer overflow and underflow

Floating Points and Precision

– Correct handling the right precision when dealing ratios and rates

Access & Privilege Control Vulnerability

Circuit Breaker

✓ Provide pause functionality for control and emergency handling

Restriction

✓ Provide proper access control for functions

- ✓ Establish rate limiter for certain operations
- ✓ Restrict access to sensitive functions
- ✓ Restrict permission to contract destruction
- ✓ Establish speed bumps slow down some sensitive actions, any malicious actions occur, there is time to recover.

DoS Vulnerability

A type of attacks that make the contract inoperable with certain period of time or permanently.

Unexpected Revert

- ✓ Use favor pull over push pattern for handling unexpected revert

Block Gas Limit

- – Use favor pull over push pattern for handling gas spent exceeds its limit on Contract via unbounded operations
- ✓ Use favor pull over push pattern for handling gas spent exceeds its limit on the network via block stuffing

Miner Manipulation Vulnerability

BlockNumber Dependence

- – Understand the security risk level and trade-off of using `block.number` as one of core factors in the contract. Be aware that `block.number` can not be manipulated by the miner, but can lead to larger than expected time differences. With the assumptions that an Ethereum block confirmation takes 13 seconds. However, the average block time is between 13 to 15 seconds. During the difficulty bomb stage or hard/soft fork upgrade of the network, `block.number` to a time is dangerous and inaccurate as expected.

Timestamp Dependence

- ✓ Understand the security risk level and trade-off of using `block.timestamp` or alias `now` as one of core factors in the contract.
- – Correct use of 15-second rule to minimize the impact caused by timestamp variance

Transaction Ordering Or Front-Running

- – Understand the security risk level and the `gasPrice` rule in this vulnerability
- – Correct placing an upper bound on the `gasPrice` for preventing the users taking the benefit of transaction ordering

External Referencing Vulnerability

External calls may execute malicious code in that contract or any other contract that it depends upon. As such, every external call should be treated as a potential security risk

✓ Correct using the pull over push favor for external calls to reduce reduces the chance of problems with the gas limit.

Avoid state changes after external calls

✓ Correct using checks-effects-interactions pattern to minimize the state changes after external contract or call referencing.

Handle errors in external calls

✓ Correct handling errors in any external contract or call referencing by checking its return value

Race Conditions Vulnerability

A type of vulnerability caused by calling external contracts that attacker can take over the control flow, and make changes to the data that the calling function wasn't expecting. **Types of race conditions**:

Reentrancy

A state variable is changed after a contract uses `call.value()()`.

Cross-function Race Conditions

An attacker may also be able to do a similar attack using two different functions that share the same state

- Avoid using `call.value()()`, instead use `send()`, `transfer()` that consumes 2300 gas. This will prevent any external code from being executed continuously
- Finish all internal work before calling the external function for unavoidable external call.

Low-level Call Vulnerability

The low-level functions or opcodes are very useful and dangerous for allowing the Libraries implementation and modularized code. However it opens up the doors to vulnerabilities as essentially your contract is allowing anyone to do whatever they want with their state

Code Injection by delegatecall

✓ Ensure the libraries implementation is stateless and non-self-destructable

Visibility Vulnerability

Solidity functions have 4 difference visibilities to dictate how functions are allowed to be called. The visibility determines whether a function can be called externally by users, by other derived contracts, only internally or only externally.

- ✓ Specify the visibility of all functions in a contract, even if they are intentionally public

Incorrect Interface Vulnerability

A contract interface defines functions with a different type signature than the implementation, causing two different method id's to be created. As a result, when the interface is called, the fallback method will be executed.

- ✓ Ensure the defined function signatures are match with the contract interface and implementation

Bad Randomness

Pseudo random number generation is not supported by Solidity as default, which it is an unsafe operation.

- – Avoid using randomness for block variables, there may be a chance manipulated by the miners

Documentation

- ✓ Provide project README and execution guidance
- ✓ Provide inline comment for complex functions intention
- ✓ Provide instruction to initialize and execute the test files

Testing

- ✓ Provide migration scripts for continuously contracts deployment to the Ethereum network
- ✓ Provide test scripts and coverage for potential scenarios

Overall we found the smart contracts to follow good practices. With the final update of source code and delivery of the audit report, we conclude that the contract is structurally sound and not vulnerable to any classically known anti-patterns or security issues. The audit report itself is not necessarily a guarantee of correctness or trustworthiness, and we always recommend to seek multiple opinions, keep improving the codebase, and more test coverage and sandbox deployments before the mainnet release.

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 1 in File Manager.sol

```
1 pragma solidity 0.5.7;
```

! Version to compile has the following bug: 0.5.7: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor, IncorrectEventSignatureInLibraries

INSECURE_COMPILER_VERSION

Line 1 in File Vault.sol

```
1 pragma solidity 0.5.7;
```

! Version to compile has the following bug: 0.5.7: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor, IncorrectEventSignatureInLibraries

INSECURE_COMPILER_VERSION

Line 1 in File Proposal.sol

```
1 pragma solidity 0.5.7;
```

! Version to compile has the following bug: 0.5.7: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor, IncorrectEventSignatureInLibraries

TIMESTAMP_DEPENDENCY

Line 132 in File Proposal.sol

```
132     require(now > time, "wait to execute");
```

! ”now” can be influenced by minors to some degree

INSECURE_COMPILER_VERSION

Line 1 in File Basket.sol

```
1 pragma solidity 0.5.7;
```

! Version to compile has the following bug: 0.5.7: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor, IncorrectEventSignatureInLibraries

INSECURE_COMPILER_VERSION

Line 1 in File SafeMath.sol

```
1 pragma solidity 0.5.7;
```

! Version to compile has the following bug: 0.5.7: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor, IncorrectEventSignatureInLibraries

INSECURE_COMPILER_VERSION

Line 1 in File Ownable.sol

```
pragma solidity 0.5.7;
```

Version to compile has the following bug: 0.5.7: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor, IncorrectEventSignatureInLibraries

INSECURE_COMPILER_VERSION

Line 1 in File ReserveEternalStorage.sol

```
pragma solidity 0.5.7;
```

Version to compile has the following bug: 0.5.7: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor, IncorrectEventSignatureInLibraries

INSECURE_COMPILER_VERSION

Line 1 in File Reserve.sol

```
pragma solidity 0.5.7;
```

Version to compile has the following bug: 0.5.7: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor, IncorrectEventSignatureInLibraries

Formal Verification Results

How to read

Detail for Request 1

transferFrom to same address

Verification date	20, Oct 2018
Verification timespan	395.38 ms

CERTIK *label location*	Line 30-34 in File howtoread.sol

CERTIK *label*

```
30  /*@CTK FAIL "transferFrom to same address"
31      @tag assume_completion
32      @pre from == to
33      @post __post.allowed[from][msg.sender] ==
34  */
```

Raw code location	Line 35-41 in File howtoread.sol

Raw code

```
35  function transferFrom(address from, address to
        ) {
36      balances[from] = balances[from].sub(tokens
37      allowed[from][msg.sender] = allowed[from][
38      balances[to] = balances[to].add(tokens);
39      emit Transfer(from, to, tokens);
40      return true;
41  }
```

Counterexample This code violates the specification

Initial environment

```
1  Counter Example:
2  Before Execution:
3      Input = {
4          from = 0x0
5          to = 0x0
6          tokens = 0x6c
7      }
8      This = 0
53         balance: 0x0
54     }
55 }
```

Post environment

```
56
57 After Execution:
58     Input = {
59         from = 0x0
60         to = 0x0
61         tokens = 0x6c
```

Formal Verification Request 1

setIssuancePaused

09, Oct 2019
29.13 ms

Line 180-184 in File Manager.sol

```
180 /*@CTK setIssuancePaused
181   @tag assume_completion
182   @post msg.sender == operator
183   @post __post.issuancePaused == val
184  */
```

Line 185-188 in File Manager.sol

```
185 function setIssuancePaused(bool val) external onlyOperator {
186     emit IssuancePausedChanged(issuancePaused, val);
187     issuancePaused = val;
188 }
```

The code meets the specification.

Formal Verification Request 2

setEmergency

09, Oct 2019
25.23 ms

Line 191-195 in File Manager.sol

```
191 /*@CTK setEmergency
192   @tag assume_completion
193   @post msg.sender == operator
194   @post __post.emergency == val
195  */
```

Line 196-199 in File Manager.sol

```
196 function setEmergency(bool val) external onlyOperator {
197     emit EmergencyChanged(emergency, val);
198     emergency = val;
199 }
```

The code meets the specification.

Formal Verification Request 3

setVault

09, Oct 2019
23.18 ms

Line 213-217 in File Manager.sol

```
213 /*@CTK setVault
214   @tag assume_completion
215   @post msg.sender == operator
216   @post __post.proposalsLength == 0
217  */
```

Line 218-221 in File Manager.sol

```
218 function clearProposals() external onlyOperator {
219     proposalsLength = 0;
220     emit ProposalsCleared();
221 }
```

✅ The code meets the specification.

Formal Verification Request 4

setOperator

📅 09, Oct 2019
⏱ 48.44 ms

Line 224-228 in File Manager.sol

```
224 /*@CTK setOperator
225   @tag assume_completion
226   @post msg.sender == _owner
227   @post __post.operator == _operator
228  */
```

Line 229-232 in File Manager.sol

```
229 function setOperator(address _operator) external onlyOwner {
230     emit OperatorChanged(operator, _operator);
231     operator = _operator;
232 }
```

✅ The code meets the specification.

Formal Verification Request 5

setSeigniorage

📅 09, Oct 2019
⏱ 41.53 ms

Line 235-240 in File Manager.sol

```
235 /*@CTK setSeigniorage
236   @tag assume_completion
237   @post msg.sender == _owner
238   @post _seigniorage <= 1000
239   @post __post.seigniorage == _seigniorage
240  */
```

Line 241-245 in File Manager.sol

```
function setSeigniorage(uint256 _seigniorage) external onlyOwner {
    require(_seigniorage <= 1000, "max seigniorage 10%");
    emit SeigniorageChanged(seigniorage, _seigniorage);
    seigniorage = _seigniorage;
}
```

The code meets the specification.

Formal Verification Request 6

setDelay

09, Oct 2019
30.52 ms

Line 248-252 in File Manager.sol

```
/*@CTK setDelay
  @tag assume_completion
  @post msg.sender == _owner
  @post __post.delay == _delay
 */
```

Line 253-256 in File Manager.sol

```
function setDelay(uint256 _delay) external onlyOwner {
    emit DelayChanged(delay, _delay);
    delay = _delay;
}
```

The code meets the specification.

Formal Verification Request 7

Vault

09, Oct 2019
18.71 ms

Line 29-32 in File Vault.sol

```
/*@CTK Vault
  @tag assume_completion
  @post __post.manager == msg.sender
 */
```

Line 33-37 in File Vault.sol

```
constructor() public {
    // Initialize manager as _msgSender()
    manager = _msgSender();
    emit ManagerTransferred(address(0), manager);
}
```

The code meets the specification.

Formal Verification Request 8

changeManager

09, Oct 2019
51.35 ms

Line 46-51 in File Vault.sol

```
46  /*@CTK changeManager
47    @tag assume_completion
48    @post msg.sender == _owner
49    @post newManager != address(0)
50    @post __post.manager == newManager
51   */
```

Line 52-56 in File Vault.sol

```
52  function changeManager(address newManager) external onlyOwner {
53      require(newManager != address(0), "cannot be 0 address");
54      emit ManagerTransferred(manager, newManager);
55      manager = newManager;
56  }
```

The code meets the specification.

Formal Verification Request 9

Proposal

09, Oct 2019
7.15 ms

Line 83-87 in File Proposal.sol

```
83  /*@CTK Proposal
84    @tag assume_completion
85    @post __post.proposer == _proposer
86    @post __post.state == State.Created
87   */
```

Line 88-92 in File Proposal.sol

```
88  constructor(address _proposer) public {
89      proposer = _proposer;
90      state = State.Created;
91      emit ProposalCreated(proposer);
92  }
```

The code meets the specification.

Formal Verification Request 10

accept

09, Oct 2019
66.21 ms

Line 95-100 in File Proposal.sol

```
95   /*@CTK accept
96     @tag assume_completion
97     @post msg.sender == _owner
98     @post __post.time == _time
99     @post __post.state == State.Accepted
100    */
```

Line 101-106 in File Proposal.sol

```
101  function accept(uint256 _time) external onlyOwner {
102      require(state == State.Created, "proposal not created");
103      time = _time;
104      state = State.Accepted;
105      emit ProposalAccepted(proposer, _time);
106  }
```

The code meets the specification.

Formal Verification Request 11

cancel

09, Oct 2019
41.44 ms

Line 109-113 in File Proposal.sol

```
109  /*@CTK cancel
110    @tag assume_completion
111    @post msg.sender == _owner
112    @post __post.state == State.Cancelled
113    */
```

Line 114-118 in File Proposal.sol

```
114  function cancel() external onlyOwner {
115      require(state != State.Completed);
116      state = State.Cancelled;
117      emit ProposalCancelled(proposer);
118  }
```

The code meets the specification.

Formal Verification Request 12

WeightProposal

09, Oct 2019
52.75 ms

Line 156-159 in File Proposal.sol

```
156  /*@CTK WeightProposal
157    @tag assume_completion
158    @post __post.trustedBasket == _trustedBasket
159    */
```

Line 160-163 in File Proposal.sol

```
160    constructor(address _proposer, Basket _trustedBasket) Proposal(_proposer) public {
161        require(_trustedBasket.size() > 0, "proposal cannot be empty");
162        trustedBasket = _trustedBasket;
163    }
```

The code meets the specification.

Formal Verification Request 13

SwapProposal

09, Oct 2019
83.98 ms

Line 189-196 in File Proposal.sol

```
189    /*@CTK SwapProposal
190      @tag assume_completion
191      @post _tokens.length > 0
192      @post (_tokens.length == _amounts.length) && (_amounts.length == _toVault.length)
193      @post __post.tokens == _tokens
194      @post __post.amounts == _amounts
195      @post __post.toVault == _toVault
196     */
```

Line 197-209 in File Proposal.sol

```
197    constructor(address _proposer,
198                address[] memory _tokens,
199                uint256[] memory _amounts, // unit: qToken
200                bool[] memory _toVault )
201        Proposal(_proposer) public
202    {
203        require(_tokens.length > 0, "proposal cannot be empty");
204        require(_tokens.length == _amounts.length && _amounts.length == _toVault.length
           ,
205                "unequal array lengths");
206        tokens = _tokens;
207        amounts = _amounts;
208        toVault = _toVault;
209    }
```

The code meets the specification.

Formal Verification Request 14

getTokens

09, Oct 2019
5.76 ms

Line 81-84 in File Basket.sol

```
/*@CTK getTokens
  @tag assume_completion
  @post __return == tokens
 */
```

Line 85-87 in File Basket.sol

```
function getTokens() external view returns(address[] memory) {
    return tokens;
}
```

✅ The code meets the specification.

Formal Verification Request 15

size

📅 09, Oct 2019
⏱ 5.22 ms

Line 89-92 in File Basket.sol

```
/*@CTK size
  @tag assume_completion
  @post __return == tokens.length
 */
```

Line 93-95 in File Basket.sol

```
function size() external view returns(uint256) {
    return tokens.length;
}
```

✅ The code meets the specification.

Formal Verification Request 16

SafeMath add

📅 09, Oct 2019
⏱ 16.85 ms

Line 26-34 in File SafeMath.sol

```
/*@CTK "SafeMath add"
  @tag spec
  @tag is_pure
  @post (a + b < a || a + b < b) == __reverted
  @post !__reverted -> __return == a + b
  @post !__reverted -> !__has_overflow
  @post !__reverted -> !__has_assertion_failure
  @post !(__has_buf_overflow)
*/
```

Line 35-40 in File SafeMath.sol

```
function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a, "SafeMath: addition overflow");

    return c;
}
```

The code meets the specification.

Formal Verification Request 17

SafeMath sub

09, Oct 2019
15.42 ms

Line 51-59 in File SafeMath.sol

```
/*@CTK "SafeMath sub"
  @tag spec
  @tag is_pure
  @post (b > a) == __reverted
  @post !__reverted -> __return == a - b
  @post !__reverted -> !__has_overflow
  @post !__reverted -> !__has_assertion_failure
  @post !(__has_buf_overflow)
*/
```

Line 60-65 in File SafeMath.sol

```
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b <= a, "SafeMath: subtraction overflow");
    uint256 c = a - b;

    return c;
}
```

The code meets the specification.

Formal Verification Request 18

SafeMath mul zero

09, Oct 2019
19.3 ms

Line 76-81 in File SafeMath.sol

```
/*@CTK "SafeMath mul zero"
  @tag spec
  @tag is_pure
  @pre (a == 0)
  @post __return == 0
*/
```

Line 92-104 in File SafeMath.sol

```
92    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
93        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
94        // benefit is lost if 'b' is also tested.
95        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
96        if (a == 0) {
97            return 0;
98        }
99
100       uint256 c = a * b;
101       require(c / a == b, "SafeMath: multiplication overflow");
102
103       return c;
104   }
```

The code meets the specification.

Formal Verification Request 19

SafeMath mul nonzero

09, Oct 2019
292.5 ms

Line 82-91 in File SafeMath.sol

```
82    /*@CTK "SafeMath mul nonzero"
83      @tag spec
84      @tag is_pure
85      @pre (a != 0)
86      @post (a * b / a != b) == __reverted
87      @post !__reverted -> __return == a * b
88      @post !__reverted -> !__has_overflow
89      @post !__reverted -> !__has_assertion_failure
90      @post !(__has_buf_overflow)
91    */
```

Line 92-104 in File SafeMath.sol

```
92    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
93        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
94        // benefit is lost if 'b' is also tested.
95        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
96        if (a == 0) {
97            return 0;
98        }
99
100       uint256 c = a * b;
101       require(c / a == b, "SafeMath: multiplication overflow");
102
103       return c;
104   }
```

The code meets the specification.

Formal Verification Request 20

SafeMath div

09, Oct 2019
14.27 ms

Line 117-125 in File SafeMath.sol

```
117 /*@CTK "SafeMath div"
118   @tag spec
119   @tag is_pure
120   @post (b == 0) == __reverted
121   @post !__reverted -> __return == a / b
122   @post !__reverted -> !__has_overflow
123   @post !__reverted -> !__has_assertion_failure
124   @post !(__has_buf_overflow)
125 */
```

Line 126-133 in File SafeMath.sol

```
126 function div(uint256 a, uint256 b) internal pure returns (uint256) {
127     // Solidity only automatically asserts when dividing by 0
128     require(b > 0, "SafeMath: division by zero");
129     uint256 c = a / b;
130     // assert(a == b * c + a % b); // There is no case in which this doesn't hold
131
132     return c;
133 }
```

The code meets the specification.

Formal Verification Request 21

SafeMath mod

09, Oct 2019
12.71 ms

Line 146-154 in File SafeMath.sol

```
146 /*@CTK "SafeMath mod"
147   @tag spec
148   @tag is_pure
149   @post (b == 0) == __reverted
150   @post !__reverted -> __return == a % b
151   @post !__reverted -> !__has_overflow
152   @post !__reverted -> !__has_assertion_failure
153   @post !(__has_buf_overflow)
154 */
```

Line 155-158 in File SafeMath.sol

```
155 function mod(uint256 a, uint256 b) internal pure returns (uint256) {
156     require(b != 0, "SafeMath: modulo by zero");
157     return a % b;
158 }
```

The code meets the specification.

Formal Verification Request 22

Ownable

09, Oct 2019
21.67 ms

Line 25-28 in File Ownable.sol

```
25  /*@CTK Ownable
26    @tag assume_completion
27    @post __post._owner == msg.sender
28   */
```

Line 29-33 in File Ownable.sol

```
29  constructor () internal {
30      address msgSender = _msgSender();
31      _owner = msgSender;
32      emit OwnershipTransferred(address(0), msgSender);
33  }
```

The code meets the specification.

Formal Verification Request 23

nominateNewOwner

09, Oct 2019
58.27 ms

Line 66-71 in File Ownable.sol

```
66  /*@CTK nominateNewOwner
67    @tag assume_completion
68    @post msg.sender == _owner
69    @post newOwner != address(0)
70    @post __post._nominatedOwner == newOwner
71   */
```

Line 72-76 in File Ownable.sol

```
72  function nominateNewOwner(address newOwner) external onlyOwner {
73      require(newOwner != address(0), "new owner is 0 address");
74      emit NewOwnerNominated(_owner, newOwner);
75      _nominatedOwner = newOwner;
76  }
```

The code meets the specification.

Formal Verification Request 24

acceptOwnership

09, Oct 2019
19.9 ms

Line 81-85 in File Ownable.sol

```
81  /*@CTK acceptOwnership
82    @tag assume_completion
83    @post msg.sender == _nominatedOwner
84    @post __post._owner == _nominatedOwner
85   */
```

Line 86-90 in File Ownable.sol

```
86  function acceptOwnership() external {
87      require(_nominatedOwner == _msgSender(), "unauthorized");
88      emit OwnershipTransferred(_owner, _nominatedOwner);
89      _owner = _nominatedOwner;
90  }
```

The code meets the specification.

Formal Verification Request 25

renounceOwnership

09, Oct 2019
27.39 ms

Line 95-98 in File Ownable.sol

```
95  /*@CTK renounceOwnership
96    @tag assume_completion
97    @post __post._owner == address(0)
98   */
```

Line 99-102 in File Ownable.sol

```
99  function renounceOwnership() external onlyOwner {
100     emit OwnershipTransferred(_owner, address(0));
101     _owner = address(0);
102 }
```

The code meets the specification.

Formal Verification Request 26

ReserveEternalStorage

09, Oct 2019
21.17 ms

Line 32-35 in File ReserveEternalStorage.sol

```
32  /*@CTK ReserveEternalStorage
33    @tag assume_completion
34    @post __post.reserveAddress == msg.sender
35   */
```

Line 36-39 in File ReserveEternalStorage.sol

```
36  constructor() public {
37      reserveAddress = _msgSender();
38      emit ReserveAddressTransferred(address(0), reserveAddress);
39  }
```

The code meets the specification.

Formal Verification Request 27

updateReserveAddress

09, Oct 2019
54.45 ms

Line 48-53 in File ReserveEternalStorage.sol

```
48  /*@CTK updateReserveAddress
49    @tag assume_completion
50    @post newReserveAddress != address(0)
51    @post (msg.sender == reserveAddress) || (msg.sender == _owner)
52    @post __post.reserveAddress == newReserveAddress
53   */
```

Line 54-59 in File ReserveEternalStorage.sol

```
54  function updateReserveAddress(address newReserveAddress) external {
55      require(newReserveAddress != address(0), "zero address");
56      require(_msgSender() == reserveAddress || _msgSender() == owner(), "not
            authorized");
57      emit ReserveAddressTransferred(reserveAddress, newReserveAddress);
58      reserveAddress = newReserveAddress;
59  }
```

The code meets the specification.

Formal Verification Request 28

addBalance

09, Oct 2019
31.32 ms

Line 72-76 in File ReserveEternalStorage.sol

```
72  /*@CTK addBalance
73    @tag assume_completion
74    @post msg.sender == reserveAddress
75    @post __post.balance[key] == balance[key] + value
76   */
```

Line 77-79 in File ReserveEternalStorage.sol

```
77  function addBalance(address key, uint256 value) external onlyReserveAddress {
78      balance[key] = balance[key].add(value);
79  }
```

The code meets the specification.

Formal Verification Request 29

subBalance

09, Oct 2019
30.77 ms

Line 82-86 in File ReserveEternalStorage.sol

```
82  /*@CTK subBalance
83    @tag assume_completion
84    @post (msg.sender == reserveAddress)
85    @post __post.balance[key] == (balance[key] - value)
86   */
```

Line 87-89 in File ReserveEternalStorage.sol

```
87  function subBalance(address key, uint256 value) external onlyReserveAddress {
88      balance[key] = balance[key].sub(value);
89  }
```

The code meets the specification.

Formal Verification Request 30

If method completes, integer overflow would not happen.

09, Oct 2019
22.79 ms

Line 92 in File ReserveEternalStorage.sol

```
92  //@CTK NO_OVERFLOW
```

Line 100-102 in File ReserveEternalStorage.sol

```
100 function setBalance(address key, uint256 value) external onlyReserveAddress {
101     balance[key] = value;
102 }
```

The code meets the specification.

Formal Verification Request 31

Buffer overflow / array index out of bound would never happen.

09, Oct 2019
0.46 ms

Line 93 in File ReserveEternalStorage.sol

```
93  //@CTK NO_BUF_OVERFLOW
```

Line 100-102 in File ReserveEternalStorage.sol

```
100 function setBalance(address key, uint256 value) external onlyReserveAddress {
101     balance[key] = value;
102 }
```

The code meets the specification.

Formal Verification Request 32

Method will not encounter an assertion failure.

09, Oct 2019
0.46 ms

Line 94 in File ReserveEternalStorage.sol

```
94    //@CTK NO_ASF
```

Line 100-102 in File ReserveEternalStorage.sol

```
100   function setBalance(address key, uint256 value) external onlyReserveAddress {
101       balance[key] = value;
102   }
```

The code meets the specification.

Formal Verification Request 33

setBalance

09, Oct 2019
1.69 ms

Line 95-99 in File ReserveEternalStorage.sol

```
95    /*@CTK setBalance
96      @tag assume_completion
97      @post msg.sender == reserveAddress
98      @post __post.balance[key] == value
99     */
```

Line 100-102 in File ReserveEternalStorage.sol

```
100   function setBalance(address key, uint256 value) external onlyReserveAddress {
101       balance[key] = value;
102   }
```

The code meets the specification.

Formal Verification Request 34

If method completes, integer overflow would not happen.

09, Oct 2019
24.4 ms

Line 111 in File ReserveEternalStorage.sol

```
111   //@CTK NO_OVERFLOW
```

Line 119-121 in File ReserveEternalStorage.sol

```
119   function setAllowed(address from, address to, uint256 value) external
          onlyReserveAddress {
120       allowed[from][to] = value;
121   }
```

The code meets the specification.

Formal Verification Request 35

Buffer overflow / array index out of bound would never happen.

09, Oct 2019
0.47 ms

Line 112 in File ReserveEternalStorage.sol

```
112    //@CTK NO_BUF_OVERFLOW
```

Line 119-121 in File ReserveEternalStorage.sol

```
119    function setAllowed(address from, address to, uint256 value) external
           onlyReserveAddress {
120        allowed[from][to] = value;
121    }
```

The code meets the specification.

Formal Verification Request 36

Method will not encounter an assertion failure.

09, Oct 2019
0.46 ms

Line 113 in File ReserveEternalStorage.sol

```
113    //@CTK NO_ASF
```

Line 119-121 in File ReserveEternalStorage.sol

```
119    function setAllowed(address from, address to, uint256 value) external
           onlyReserveAddress {
120        allowed[from][to] = value;
121    }
```

The code meets the specification.

Formal Verification Request 37

setAllowed

09, Oct 2019
1.57 ms

Line 114-118 in File ReserveEternalStorage.sol

```
114    /*@CTK setAllowed
115      @tag assume_completion
116      @post msg.sender == reserveAddress
117      @post __post.allowed[from][to] == value
118     */
```

Line 119-121 in File ReserveEternalStorage.sol

```
119 function setAllowed(address from, address to, uint256 value) external
        onlyReserveAddress {
120     allowed[from][to] = value;
121 }
```

✅ The code meets the specification.

Formal Verification Request 38

changeMinter

📅 09, Oct 2019
⏱ 35.04 ms

Line 112-116 in File Reserve.sol

```
112 /*@CTK changeMinter
113   @tag assume_completion
114   @post (msg.sender == _owner) || (msg.sender == minter)
115   @post __post.minter == newMinter
116  */
```

Line 117-120 in File Reserve.sol

```
117 function changeMinter(address newMinter) external onlyOwnerOr(minter) {
118     minter = newMinter;
119     emit MinterChanged(newMinter);
120 }
```

✅ The code meets the specification.

Formal Verification Request 39

changePauser

📅 09, Oct 2019
⏱ 28.38 ms

Line 123-127 in File Reserve.sol

```
123 /*@CTK changePauser
124   @tag assume_completion
125   @post (msg.sender == _owner) || (msg.sender == pauser)
126   @post __post.pauser == newPauser
127  */
```

Line 128-131 in File Reserve.sol

```
128 function changePauser(address newPauser) external onlyOwnerOr(pauser) {
129     pauser = newPauser;
130     emit PauserChanged(newPauser);
131 }
```

✅ The code meets the specification.

Formal Verification Request 40

changePauser

09, Oct 2019
31.12 ms

Line 133-137 in File Reserve.sol

```
133 /*@CTK changePauser
134   @tag assume_completion
135   @post (msg.sender == _owner) || (msg.sender == feeRecipient)
136   @post __post.feeRecipient == newFeeRecipient
137  */
```

Line 138-141 in File Reserve.sol

```
138 function changeFeeRecipient(address newFeeRecipient) external onlyOwnerOr(
        feeRecipient) {
139     feeRecipient = newFeeRecipient;
140     emit FeeRecipientChanged(newFeeRecipient);
141 }
```

The code meets the specification.

Formal Verification Request 41

transferEternalStorage

09, Oct 2019
181.89 ms

Line 146-151 in File Reserve.sol

```
146 /*@CTK transferEternalStorage
147   @tag assume_completion
148   @post msg.sender == _owner
149   @post newReserveAddress != address(0)
150   @post paused == true
151  */
```

Line 152-156 in File Reserve.sol

```
152 function transferEternalStorage(address newReserveAddress) external onlyOwner
        isPaused {
153     require(newReserveAddress != address(0), "zero address");
154     emit EternalStorageTransferred(newReserveAddress);
155     trustedData.updateReserveAddress(newReserveAddress);
156 }
```

The code meets the specification.

Formal Verification Request 42

changeTxFeeHelper

09, Oct 2019
32.67 ms

Line 159-162 in File Reserve.sol

```
159 /*@CTK changeTxFeeHelper
160   @tag assume_completion
161   @post msg.sender == _owner
162  */
```

Line 163-165 in File Reserve.sol

```
163 function changeTxFeeHelper(address newTrustedTxFee) external onlyOwner {
164     trustedTxFee = ITXFee(newTrustedTxFee);
165 }
```

The code meets the specification.

Formal Verification Request 43

changeMaxSupply

09, Oct 2019
33.12 ms

Line 168-172 in File Reserve.sol

```
168 /*@CTK changeMaxSupply
169   @tag assume_completion
170   @post (msg.sender == _owner)
171   @post __post.maxSupply == newMaxSupply
172  */
```

Line 173-176 in File Reserve.sol

```
173 function changeMaxSupply(uint256 newMaxSupply) external onlyOwner {
174     maxSupply = newMaxSupply;
175     emit MaxSupplyChanged(newMaxSupply);
176 }
```

The code meets the specification.

Formal Verification Request 44

pause

09, Oct 2019
24.53 ms

Line 179-183 in File Reserve.sol

```
179 /*@CTK pause
180   @tag assume_completion
181   @post (msg.sender == pauser)
182   @post __post.paused == true
183  */
```

Line 184-187 in File Reserve.sol

```
function pause() external only(pauser) {
    paused = true;
    emit Paused(pauser);
}
```

The code meets the specification.

Formal Verification Request 45

pause

09, Oct 2019
25.45 ms

Line 190-194 in File Reserve.sol

```
/*@CTK pause
  @tag assume_completion
  @post (msg.sender == pauser)
  @post __post.paused == false
 */
```

Line 195-198 in File Reserve.sol

```
function unpause() external only(pauser) {
    paused = false;
    emit Unpaused(pauser);
}
```

The code meets the specification.

Formal Verification Request 46

mint

09, Oct 2019
231.06 ms

Line 324-331 in File Reserve.sol

```
/*@CTK mint
  @tag assume_completion
  @post paused == false
  @post msg.sender == minter
  @post account != address(0)
  @post __post.totalSupply == totalSupply + value
  @post __post.totalSupply < maxSupply
 */
```

Line 332-343 in File Reserve.sol

```
function mint(address account, uint256 value)
    external
    notPaused
    only(minter)
{
    require(account != address(0), "can't mint to address zero");
```

```

    totalSupply = totalSupply.add(value);
    require(totalSupply < maxSupply, "max supply exceeded");
    trustedData.addBalance(account, value);
    emit Transfer(address(0), account, value);
}
```

The code meets the specification.

Formal Verification Request 47

_burn

09, Oct 2019
151.77 ms

Line 385-389 in File Reserve.sol

```
/*@CTK _burn
  @tag assume_completion
  @post account != address(0)
  @post __post.totalSupply == totalSupply - value
 */
```

Line 390-396 in File Reserve.sol

```
function _burn(address account, uint256 value) internal {
    require(account != address(0), "can't burn from address zero");

    totalSupply = totalSupply.sub(value);
    trustedData.subBalance(account, value);
    emit Transfer(account, address(0), value);
}
```

The code meets the specification.

Formal Verification Request 48

_approve

09, Oct 2019
80.33 ms

Line 400-404 in File Reserve.sol

```
/*@CTK _approve
  @tag assume_completion
  @post spender != address(0)
  @post holder != address(0)
 */
```

Line 405-411 in File Reserve.sol

```
function _approve(address holder, address spender, uint256 value) internal {
    require(spender != address(0), "spender cannot be address zero");
    require(holder != address(0), "holder cannot be address zero");

```

```
        trustedData.setAllowed(holder, spender, value);
        emit Approval(holder, spender, value);
    }
```

The code meets the specification.

Source Code with CertiK Labels

File Manager.sol

```
pragma solidity 0.5.7;

import "./zeppelin/token/ERC20/SafeERC20.sol";
import "./zeppelin/token/ERC20/IERC20.sol";
import "./zeppelin/math/SafeMath.sol";
import "./rsv/IRSV.sol";
import "./ownership/Ownable.sol";
import "./Basket.sol";
import "./Proposal.sol";


interface IVault {
    function withdrawTo(address, uint256, address) external;
}

/**
 * The Manager contract is the point of contact between the Reserve ecosystem and the
 * surrounding world. It manages the Issuance and Redemption of RSV, a decentralized stablecoin
 * backed by a basket of tokens.
 *
 * The Manager also implements a Proposal system to handle administration of changes to the
 * backing of RSV. Anyone can propose a change to the backing. Once the `owner` approves the
 * proposal, then after a pre-determined delay the proposal is eligible for execution by
 * anyone. However, the funds to execute the proposal must come from the proposer.
 *
 * There are two different ways to propose changes to the backing of RSV:
 * - proposeSwap()
 * - proposeWeights()
 *
 * In both cases, tokens are exchanged with the Vault and a new RSV backing is set. You can
 * think of the first type of proposal as being useful when you don't want to rebalance the
 * Vault by exchanging absolute quantities of tokens; its downside is that you don't know
 * precisely what the resulting basket weights will be. The second type of proposal is more
 * useful when you want to fine-tune the Vault weights and accept the downside that it's
 * difficult to know what capital will be required when the proposal is executed.
 */

/* On "unit" comments:
 *
 * The units in use around weight computations are fiddly, and it's pretty annoying to get them
 * properly into the Solidity type system. So, there are many comments of the form "unit:
 * ...". Where such a comment is describing a field, method, or return parameter, the comment means
```

```
 * that the data in that place is to be interpreted to have that type. Many places also have
 * comments with more complicated expressions; that's manually working out the dimensional analysis
 * to ensure that the given expression has correct units.
 *
 * Some dimensions used in this analysis:
 * - 1 RSV: 1 Reserve
 * - 1 qRSV: 1 quantum of Reserve.
 *     (RSV & qRSV are convertible by .mul(10**reserve.decimals() qRSV/RSV))
 * - 1 qToken: 1 quantum of an external Token.
 * - 1 aqToken: 1 atto-quantum of an external Token.
 *     (qToken and aqToken are convertible by .mul(10**18 aqToken/qToken)
 * - 1 BPS: 1 Basis Point. Effectively dimensionless; convertible with .mul(10000 BPS).
 *
 * Note that we _never_ reason in units of Tokens or attoTokens.
 */
contract Manager is Ownable {
    using SafeERC20 for IERC20;
    using SafeMath for uint256;

    // ROLES

    // Manager is already Ownable, but in addition it also has an `operator`.
    address public operator;

    // DATA

    Basket public trustedBasket;
    IVault public trustedVault;
    IRSV public trustedRSV;
    IProposalFactory public trustedProposalFactory;

    // Proposals
    mapping(uint256 => IProposal) public trustedProposals;
    uint256 public proposalsLength;
    uint256 public delay = 24 hours;

    // Pausing
    bool public issuancePaused;
    bool public emergency;

    // The spread between issuance and redemption in basis points (BPS).
    uint256 public seigniorage;        // 0.1% spread -> 10 BPS. unit: BPS
    uint256 constant BPS_FACTOR = 10000; // This is what 100% looks like in BPS. unit: BPS
    uint256 constant WEIGHT_SCALE = 10**18; // unit: aqToken/qToken

    event ProposalsCleared();

    // RSV traded events
    event Issuance(address indexed user, uint256 indexed amount);
    event Redemption(address indexed user, uint256 indexed amount);

    // Pause events
    event IssuancePausedChanged(bool indexed oldVal, bool indexed newVal);
    event EmergencyChanged(bool indexed oldVal, bool indexed newVal);
```

```
    event OperatorChanged(address indexed oldAccount, address indexed newAccount);
    event SeigniorageChanged(uint256 oldVal, uint256 newVal);
    event VaultChanged(address indexed oldVaultAddr, address indexed newVaultAddr);
    event DelayChanged(uint256 oldVal, uint256 newVal);

    // Proposals
    event WeightsProposed(uint256 indexed id,
        address indexed proposer,
        address[] tokens,
        uint256[] weights);

    event SwapProposed(uint256 indexed id,
        address indexed proposer,
        address[] tokens,
        uint256[] amounts,
        bool[] toVault);

    event ProposalAccepted(uint256 indexed id, address indexed proposer);
    event ProposalCanceled(uint256 indexed id, address indexed proposer, address
        indexed canceler);
    event ProposalExecuted(uint256 indexed id,
        address indexed proposer,
        address indexed executor,
        address oldBasket,
        address newBasket);

    // ============================ Constructor ==============================

    /// Begins in `emergency` state.
    /*@CTK Manager
      @tag assume_completion
      @post _seigniorage <= 1000
      @post __post.operator == operatorAddr
      @post __post.seigniorage == seigniorage_
      @post __post.emergency == true
      @post __post.issuancePaused == false
     */
    constructor(
        address vaultAddr,
        address rsvAddr,
        address proposalFactoryAddr,
        address basketAddr,
        address operatorAddr,
        uint256 _seigniorage) public {
        require(_seigniorage <= 1000, "max seigniorage 10%");
        trustedVault = IVault(vaultAddr);
        trustedRSV = IRSV(rsvAddr);
        trustedProposalFactory = IProposalFactory(proposalFactoryAddr);
        trustedBasket = Basket(basketAddr);
        operator = operatorAddr;
        seigniorage = _seigniorage;
        emergency = true; // it's not an emergency, but we want everything to start
            paused.
    }

    // ============================ Modifiers ==============================

    /// Modifies a function to run only when issuance is not paused.
```

```
    modifier issuanceNotPaused() {
        require(!issuancePaused, "issuance is paused");
        _;
    }

    /// Modifies a function to run only when there is not some emergency that requires
         upgrades.
    modifier notEmergency() {
        require(!emergency, "contract is paused");
        _;
    }

    /// Modifies a function to run only when the caller is the operator account.
    modifier onlyOperator() {
        require(_msgSender() == operator, "operator only");
        _;
    }

    /// Modifies a function to run and complete only if the vault is collateralized.
    modifier vaultCollateralized() {
        require(isFullyCollateralized(), "undercollateralized");
        _;
        assert(isFullyCollateralized());
    }

    // ========================= Public + External ============================

    /// Set if issuance should be paused.
    /*@CTK setIssuancePaused
      @tag assume_completion
      @post msg.sender == operator
      @post __post.issuancePaused == val
     */
    function setIssuancePaused(bool val) external onlyOperator {
        emit IssuancePausedChanged(issuancePaused, val);
        issuancePaused = val;
    }

    /// Set if all contract actions should be paused.
    /*@CTK setEmergency
      @tag assume_completion
      @post msg.sender == operator
      @post __post.emergency == val
     */
    function setEmergency(bool val) external onlyOperator {
        emit EmergencyChanged(emergency, val);
        emergency = val;
    }

    /// Set the vault.
    /*@CTK setVault
      @tag assume_completion
      @post msg.sender == _owner
      @post __post.trustedVault == IVault(newVaultAddress)
     */
    function setVault(address newVaultAddress) external onlyOwner {
        emit VaultChanged(address(trustedVault), newVaultAddress);
        trustedVault = IVault(newVaultAddress);
```

```
    }

    /// Clear the list of proposals.
    /*@CTK setVault
      @tag assume_completion
      @post msg.sender == operator
      @post __post.proposalsLength == 0
     */
    function clearProposals() external onlyOperator {
        proposalsLength = 0;
        emit ProposalsCleared();
    }

    /// Set the operator.
    /*@CTK setOperator
      @tag assume_completion
      @post msg.sender == _owner
      @post __post.operator == _operator
     */
    function setOperator(address _operator) external onlyOwner {
        emit OperatorChanged(operator, _operator);
        operator = _operator;
    }

    /// Set the seigniorage, in BPS.
    /*@CTK setSeigniorage
      @tag assume_completion
      @post msg.sender == _owner
      @post _seigniorage <= 1000
      @post __post.seigniorage == _seigniorage
     */
    function setSeigniorage(uint256 _seigniorage) external onlyOwner {
        require(_seigniorage <= 1000, "max seigniorage 10%");
        emit SeigniorageChanged(seigniorage, _seigniorage);
        seigniorage = _seigniorage;
    }

    /// Set the Proposal delay in hours.
    /*@CTK setDelay
      @tag assume_completion
      @post msg.sender == _owner
      @post __post.delay == _delay
     */
    function setDelay(uint256 _delay) external onlyOwner {
        emit DelayChanged(delay, _delay);
        delay = _delay;
    }

    /// Ensure that the Vault is fully collateralized. That this is true should be an
    /// invariant of this contract: it's true before and after every txn.
    /*@CTK isFullyCollateralized
      @tag assume_completion
      @inv forall j: uint. (j >= 0 /\ j < trustedBasket.size()) -> trustedRSV.
          totalSupply() * trustedBasket.weights(trustedBasket.tokens(j)) <= IERC20(
          trustedBasket.tokens(j)).balanceOf(address(trustedVault))
     */
    function isFullyCollateralized() public view returns(bool) {
        uint256 scaleFactor = WEIGHT_SCALE.mul(uint256(10) ** trustedRSV.decimals());
```

```
        // scaleFactor unit: aqToken/qToken * qRSV/RSV

        /*@CTK loop_SafeMath
          @inv i <= trustedBasket.size()
          @inv forall j: uint. (j >= 0 /\ j < i) -> trustedRSV.totalSupply() *
              trustedBasket.weights(trustedBasket.tokens(j)) <= IERC20(trustedBasket.
              tokens(j)).balanceOf(address(trustedVault))
         */
        for (uint256 i = 0; i < trustedBasket.size(); i++) {

            address trustedToken = trustedBasket.tokens(i);
            uint256 weight = trustedBasket.weights(trustedToken); // unit: aqToken/RSV
            uint256 balance = IERC20(trustedToken).balanceOf(address(trustedVault)); //
                unit: qToken

            // Return false if this token is undercollateralized:
            if (trustedRSV.totalSupply().mul(weight) > balance.mul(scaleFactor)) {
                // checking units: [qRSV] * [aqToken/RSV] == [qToken] * [aqToken/qToken
                    * qRSV/RSV]
                return false;
            }
        }
        return true;
    }

    /// Get amounts of basket tokens required to issue an amount of RSV.
    /// The returned array will be in the same order as the current basket.tokens.
    /// return unit: qToken[]
    /*@CTK toIssue
      @tag assume_completion
      @post __return.length == trustedBasket.size()
     */
    function toIssue(uint256 rsvAmount) public view returns (uint256[] memory) {
        // rsvAmount unit: qRSV.
        uint256[] memory amounts = new uint256[](trustedBasket.size());

        uint256 feeRate = uint256(seigniorage.add(BPS_FACTOR));
        // feeRate unit: BPS
        uint256 effectiveAmount = rsvAmount.mul(feeRate).div(BPS_FACTOR);
        // effectiveAmount unit: qRSV == qRSV*BPS/BPS

        // On issuance, amounts[i] of token i will enter the vault. To maintain full
            backing,
        // we have to round _up_ each amounts[i].
        /*@CTK loop_toIssue
          @inv i <= trustedBasket.size()
          @post i == trustedBasket.size()
          @post !__should_return
         */
        for (uint256 i = 0; i < trustedBasket.size(); i++) {
            address trustedToken = trustedBasket.tokens(i);
            amounts[i] = _weighted(
                effectiveAmount,
                trustedBasket.weights(trustedToken),
                RoundingMode.UP
            );
            // unit: qToken = _weighted(qRSV, aqToken/RSV, _)
        }
```

```
        return amounts; // unit: qToken[]
    }

    /// Get amounts of basket tokens that would be sent upon redeeming an amount of RSV.
    /// The returned array will be in the same order as the current basket.tokens.
    /// return unit: qToken[]
    /*@CTK toRedeem
      @tag assume_completion
      @post __return.length == trustedBasket.size()
     */
    function toRedeem(uint256 rsvAmount) public view returns (uint256[] memory) {
        // rsvAmount unit: qRSV
        uint256[] memory amounts = new uint256[](trustedBasket.size());

        // On redemption, amounts[i] of token i will leave the vault. To maintain full backing,
        // we have to round _down_ each amounts[i].
        /*@CTK loop_toRedeem
          @inv i <= trustedBasket.size()
          @post i == trustedBasket.size()
          @post !__should_return
         */
        for (uint256 i = 0; i < trustedBasket.size(); i++) {
            address trustedToken = trustedBasket.tokens(i);
            amounts[i] = _weighted(
                rsvAmount,
                trustedBasket.weights(trustedToken),
                RoundingMode.DOWN
            );
            // unit: qToken = _weighted(qRSV, aqToken/RSV, _)
        }

        return amounts;
    }

    /// Handles issuance.
    /// rsvAmount unit: qRSV
    /*@CTK issue
      @tag assume_completion
      @post rsvAmount > 0
      @post trustedBasket.size() > 0
      @post issuancePaused == false
      @post emergency == false
      @post forall j: uint. (j >= 0 /\ j < trustedBasket.size()) -> trustedRSV.totalSupply() * trustedBasket.weights(trustedBasket.tokens(j)) <= trustedBasket.tokens(j).balanceOf(address(trustedVault))
     */
    function issue(uint256 rsvAmount) external issuanceNotPaused notEmergency vaultCollateralized {
        require(rsvAmount > 0, "cannot issue zero RSV");
        require(trustedBasket.size() > 0, "basket cannot be empty");

        // Accept collateral tokens.
        uint256[] memory amounts = toIssue(rsvAmount); // unit: qToken[]
        /*@CTK loop_issue
          @inv i <= trustedBasket.size()
```

```
        @post i == trustedBasket.size()
        @post !__should_return
       */
      for (uint256 i = 0; i < trustedBasket.size(); i++) {
         IERC20(trustedBasket.tokens(i)).safeTransferFrom(
            _msgSender(),
            address(trustedVault),
            amounts[i]
         );
         // unit check for amounts[i]: qToken.
      }

      // Compensate with RSV.
      trustedRSV.mint(_msgSender(), rsvAmount);
      // unit check for rsvAmount: qRSV.

      emit Issuance(_msgSender(), rsvAmount);
   }

   /// Handles redemption.
   /// rsvAmount unit: qRSV
   /*@CTK redeem
     @tag assume_completion
     @post rsvAmount > 0
     @post trustedBasket.size() > 0
     @post emergency == false
     @post forall j: uint. (j >= 0 /\ j < trustedBasket.size()) -> trustedRSV.
         totalSupply() * trustedBasket.weights(trustedBasket.tokens(j)) <=
         trustedBasket.tokens(j).balanceOf(address(trustedVault))
    */
   function redeem(uint256 rsvAmount) external notEmergency vaultCollateralized {
      require(rsvAmount > 0, "cannot redeem 0 RSV");
      require(trustedBasket.size() > 0, "basket cannot be empty");

      // Burn RSV tokens.
      trustedRSV.burnFrom(_msgSender(), rsvAmount);
      // unit check: rsvAmount is qRSV.

      // Compensate with collateral tokens.
      /*@CTK loop_issue
        @inv i <= trustedBasket.size()
        @post i == trustedBasket.size()
        @post !__should_return
       */
      uint256[] memory amounts = toRedeem(rsvAmount); // unit: qToken[]
      for (uint256 i = 0; i < trustedBasket.size(); i++) {
         trustedVault.withdrawTo(trustedBasket.tokens(i), amounts[i], _msgSender());
         // unit check for amounts[i]: qToken.
      }

      emit Redemption(_msgSender(), rsvAmount);
   }

   /**
    * Propose an exchange of current Vault tokens for new Vault tokens.
    *
    * These parameters are phyiscally a set of arrays because Solidity doesn't let
        you pass
```

```
 * around arrays of structs as parameters of transactions. Semantically, read these three
 * lists as a list of triples (token, amount, toVault), where:
 *
 * - token is the address of an ERC-20 token,
 * - amount is the amount of the token that the proposer says they will trade with the vault,
 * - toVault is the direction of that trade. If toVault is true, the proposer offers to send
 *   `amount` of `token` to the vault. If toVault is false, the proposer expects to receive
 *   `amount` of `token` from the vault.
 *
 * If and when this proposal is accepted and executed, then:
 *
 * 1. The Manager checks that the proposer has allowed adequate funds, for the proposed
 *    transfers from the proposer to the vault.
 * 2. The proposed set of token transfers occur between the Vault and the proposer.
 * 3. The Vault's basket weights are raised and lowered, based on these token transfers and the
 *    total supply of RSV **at the time when the proposal is executed**.
 *
 * Note that the set of token transfers will almost always be at very slightly lower volumes
 * than requested, due to the rounding error involved in (a) adjusting the weights at execution
 * time and (b) keeping the Vault fully collateralized. The contracts should never attempt to
 * trade at higher volumes than requested.
 *
 * The intended behavior of proposers is that they will make proposals that shift the Vault
 * composition towards some known target of Reserve's management while maintaining full
 * backing; the expected behavior of Reserve's management is to accept only such proposals,
 * excepting during dire emergencies.
 *
 * Note: This type of proposal does not reliably remove token addresses!
 * If you want to remove token addresses entirely, use proposeWeights.
 *
 * Returns the new proposal's ID.
 */
/*@CTK proposeSwap
  @tag assume_completion
  @post emergency == false
  @post forall j: uint. (j >= 0 /\ j < trustedBasket.size()) -> trustedRSV.totalSupply() * trustedBasket.weights(trustedBasket.tokens(j)) <= trustedBasket.tokens(j).balanceOf(address(trustedVault))
  @post (tokens.length == amounts.length) && (amounts.length == toVault.length)
  @post trustedProposals[proposalsLength]._owner == msg.sender
  @post trustedProposals[proposalsLength].proposer == msg.sender
  @post __post.proposalsLength == proposalsLength + 1
 */
function proposeSwap(
    address[] calldata tokens,
```

```
        uint256[] calldata amounts, // unit: qToken
        bool[] calldata toVault
    )
    external notEmergency vaultCollateralized returns(uint256)
    {
        require(tokens.length == amounts.length && amounts.length == toVault.length,
            "proposeSwap: unequal lengths");

        trustedProposals[proposalsLength] = trustedProposalFactory.createSwapProposal(
            _msgSender(),
            tokens,
            amounts,
            toVault
        );
        trustedProposals[proposalsLength].acceptOwnership();

        emit SwapProposed(proposalsLength, _msgSender(), tokens, amounts, toVault);
        return ++proposalsLength;
    }


    /**
     * Propose a new basket, defined by a list of tokens address, and their basket
       weights.
     *
     * Note: With this type of proposal, the allowances of tokens that will be
       required of the
     * proposer may change between proposition and execution. If the supply of RSV
       rises or falls,
     * then more or fewer tokens will be required to execute the proposal.
     *
     * Returns the new proposal's ID.
     */

    /*@CTK proposeWeights
      @tag assume_completion
      @post emergency == false
      @post forall j: uint. (j >= 0 /\ j < trustedBasket.size()) -> trustedRSV.
          totalSupply() * trustedBasket.weights(trustedBasket.tokens(j)) <=
          trustedBasket.tokens(j).balanceOf(address(trustedVault))
      @post (tokens.length == amounts.length)
      @post (tokens.length > 0)
      @post trustedProposals[proposalsLength]._owner == msg.sender
      @post trustedProposals[proposalsLength].proposer == msg.sender
      @post __post.proposalsLength == proposalsLength + 1
     */
    function proposeWeights(address[] calldata tokens, uint256[] calldata weights)
    external notEmergency vaultCollateralized returns(uint256)
    {
        require(tokens.length == weights.length, "proposeWeights: unequal lengths");
        require(tokens.length > 0, "proposeWeights: zero length");

        trustedProposals[proposalsLength] = trustedProposalFactory.createWeightProposal
            (
            _msgSender(),
            new Basket(Basket(0), tokens, weights)
        );
        trustedProposals[proposalsLength].acceptOwnership();
```

```
        emit WeightsProposed(proposalsLength, _msgSender(), tokens, weights);
        return ++proposalsLength;
    }

    /// Accepts a proposal for a new basket, beginning the required delay.
    /*@CTK acceptProposal
      @tag assume_completion
      @post msg.sender == operator
      @post emergency == false
      @post forall j: uint. (j >= 0 /\ j < trustedBasket.size()) -> trustedRSV.
          totalSupply() * trustedBasket.weights(trustedBasket.tokens(j)) <=
          trustedBasket.tokens(j).balanceOf(address(trustedVault))
      @post (proposalsLength > id)
      @post (trustedProposals[id].state == Created)
      @post (__post.trustedProposals[id].state == Accepted)
      @post (__post.trustedProposals[id].time == now + delay)
     */
    function acceptProposal(uint256 id) external onlyOperator notEmergency
        vaultCollateralized {
        require(proposalsLength > id, "proposals length <= id");
        trustedProposals[id].accept(now.add(delay));
        emit ProposalAccepted(id, trustedProposals[id].proposer());
    }

    /// Cancels a proposal. This can be done anytime before it is enacted by any of:
    /// 1. Proposer 2. Operator 3. Owner
    /*@CTK acceptProposal
      @tag assume_completion
      @post (msg.sender == _owner) || (msg.sender == operator) || (msg.sender ==
          trustedProposals[id].proposer())
      @post emergency == false
      @post forall j: uint. (j >= 0 /\ j < trustedBasket.size()) -> trustedRSV.
          totalSupply() * trustedBasket.weights(trustedBasket.tokens(j)) <=
          trustedBasket.tokens(j).balanceOf(address(trustedVault))
      @post (proposalsLength > id)
      @post (trustedProposals[id].state != Completed)
      @post (__post.trustedProposals[id].state == Cancelled)
     */
    function cancelProposal(uint256 id) external notEmergency vaultCollateralized {
        require(
            _msgSender() == trustedProposals[id].proposer() ||
            _msgSender() == owner() ||
            _msgSender() == operator,
            "cannot cancel"
        );
        require(proposalsLength > id, "proposals length <= id");
        trustedProposals[id].cancel();
        emit ProposalCanceled(id, trustedProposals[id].proposer(), _msgSender());
    }

    /// Executes a proposal by exchanging collateral tokens with the proposer.
    /*@CTK executeProposal
      @tag assume_completion
      @post(msg.sender == operator)
      @post emergency == false
      @post forall j: uint. (j >= 0 /\ j < trustedBasket.size()) -> trustedRSV.
          totalSupply() * trustedBasket.weights(trustedBasket.tokens(j)) <=
```

```
        trustedBasket.tokens(j).balanceOf(address(trustedVault))
    @post (proposalsLength > id)
    @post (trustedProposals[id].state == Accepted)
    @post (__post.trustedProposals[id].state == Completed)
   */
  function executeProposal(uint256 id) external onlyOperator notEmergency
      vaultCollateralized {
      require(proposalsLength > id, "proposals length <= id");
      address proposer = trustedProposals[id].proposer();
      Basket trustedOldBasket = trustedBasket;

      // Complete proposal and compute new basket
      trustedBasket = trustedProposals[id].complete(trustedRSV, trustedOldBasket);

      // For each token in either basket, perform transfers between proposer and
          Vault
      /*@CTK loop_executeProposal_oldBasket_tokens
        @inv i <= trustedOldBasket.size()
        @post i == trustedOldBasket.size()
        @post !__should_return
       */
      for (uint256 i = 0; i < trustedOldBasket.size(); i++) {
          address trustedToken = trustedOldBasket.tokens(i);
          _executeBasketShift(
              trustedOldBasket.weights(trustedToken),
              trustedBasket.weights(trustedToken),
              trustedToken,
              proposer
          );
      }
      /*@CTK loop_executeProposal_newBasket_tokens
        @inv i <= trustedOldBasket.size()
        @post i == trustedOldBasket.size()
        @post !__should_return
       */
      for (uint256 i = 0; i < trustedBasket.size(); i++) {
          address trustedToken = trustedBasket.tokens(i);
          if (!trustedOldBasket.has(trustedToken)) {
              _executeBasketShift(
                  trustedOldBasket.weights(trustedToken),
                  trustedBasket.weights(trustedToken),
                  trustedToken,
                  proposer
              );
          }
      }

      emit ProposalExecuted(
          id,
          proposer,
          _msgSender(),
          address(trustedOldBasket),
          address(trustedBasket)
      );
  }


  // ============================ Internal ================================
```

```
/// _executeBasketShift transfers the necessary amount of `token` between vault and `proposer`
/// to rebalance the vault's balance of token, as it goes from oldBasket to newBasket.
/// @dev To carry out a proposal, this is executed once per relevant token.
function _executeBasketShift(
    uint256 oldWeight, // unit: aqTokens/RSV
    uint256 newWeight, // unit: aqTokens/RSV
    address trustedToken,
    address proposer
) internal {
    if (newWeight > oldWeight) {
        // This token must increase in the vault, so transfer from proposer to vault.
        // (Transfer into vault: round up)
        uint256 transferAmount =_weighted(
            trustedRSV.totalSupply(),
            newWeight.sub(oldWeight),
            RoundingMode.UP
        );
        // transferAmount unit: qTokens

        if (transferAmount > 0) {
            IERC20(trustedToken).safeTransferFrom(
                proposer,
                address(trustedVault),
                transferAmount
            );
        }

    } else if (newWeight < oldWeight) {
        // This token will decrease in the vault, so transfer from vault to proposer.
        // (Transfer out of vault: round down)
        uint256 transferAmount =_weighted(
            trustedRSV.totalSupply(),
            oldWeight.sub(newWeight),
            RoundingMode.DOWN
        );
        // transferAmount unit: qTokens
        if (transferAmount > 0) {
            trustedVault.withdrawTo(trustedToken, transferAmount, proposer);
        }
    }
}

// When you perform a weighting of some amount of RSV, it will involve a division, and
// precision will be lost. When it rounds, do you want to round UP or DOWN? Be maximally
// conservative.
enum RoundingMode {UP, DOWN}

/// From a weighting of RSV (e.g., a basket weight) and an amount of RSV,
/// compute the amount of the weighted token that matches that amount of RSV.
/*@CTK _weighted
  @tag assume_completion
```

```
     @post (rnd == RoundingMode.DOWN \/ (amount * weight) % (WEIGHT_SCALE * 10**trustedRSV.decimals()) == 0) -> __return == (amount * weight) / (WEIGHT_SCALE * 10**trustedRSV.decimals())
     @post (rnd == RoundingMode.UP /\ (amount * weight) % (WEIGHT_SCALE * 10**trustedRSV.decimals()) != 0) -> __return == (amount * weight) / (WEIGHT_SCALE * 10**trustedRSV.decimals()) + 1
    */
    function _weighted(
        uint256 amount, // unit: qRSV
        uint256 weight, // unit: aqToken/RSV
        RoundingMode rnd
    ) internal view returns(uint256) // return unit: qTokens
    {
        uint256 scaleFactor = WEIGHT_SCALE.mul(uint256(10)**(trustedRSV.decimals()));
        // scaleFactor unit: aqTokens/qTokens * qRSV/RSV
        uint256 shiftedWeight = amount.mul(weight);
        // shiftedWeight unit: qRSV/RSV * aqTokens

        // If the weighting is precise, or we're rounding down, then use normal division.
        if (rnd == RoundingMode.DOWN || shiftedWeight.mod(scaleFactor) == 0) {
            return shiftedWeight.div(scaleFactor);
            // return unit: qTokens == qRSV/RSV * aqTokens * (qTokens/aqTokens * RSV/qRSV)
        }
        return shiftedWeight.div(scaleFactor).add(1); // return unit: qTokens
    }
}
```

File Vault.sol

```
pragma solidity 0.5.7;

import "./zeppelin/token/ERC20/SafeERC20.sol";
import "./zeppelin/token/ERC20/IERC20.sol";
import "./zeppelin/math/SafeMath.sol";
import "./ownership/Ownable.sol";

/**
* The Vault contract has an owner who is able to set the manager. The manager is
* able to perform withdrawals.
*/
contract Vault is Ownable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    address public manager;

    event ManagerTransferred(
        address indexed previousManager,
        address indexed newManager
    );

    event Withdrawal(
        address indexed token,
        uint256 indexed amount,
        address indexed to
    );

```

```
    /*@CTK Vault
      @tag assume_completion
      @post __post.manager == msg.sender
     */
    constructor() public {
        // Initialize manager as _msgSender()
        manager = _msgSender();
        emit ManagerTransferred(address(0), manager);
    }

    /// Modifies a function to run only when called by `manager`.
    modifier onlyManager() {
        require(_msgSender() == manager, "must be manager");
        _;
    }

    /// Changes the manager account.
    /*@CTK changeManager
      @tag assume_completion
      @post msg.sender == _owner
      @post newManager != address(0)
      @post __post.manager == newManager
     */
    function changeManager(address newManager) external onlyOwner {
        require(newManager != address(0), "cannot be 0 address");
        emit ManagerTransferred(manager, newManager);
        manager = newManager;
    }

    /// Withdraw `amount` of `token` to address `to`. Only callable by `manager`.
    /*@CTK withdrawTo
      @tag assume_completion
      @post msg.sender == manager
     */
    function withdrawTo(address token, uint256 amount, address to) external
        onlyManager {
        IERC20(token).safeTransfer(to, amount);
        emit Withdrawal(token, amount, to);
    }
}
```

File Proposal.sol

```
pragma solidity 0.5.7;

import "./zeppelin/token/ERC20/IERC20.sol";
import "./zeppelin/token/ERC20/SafeERC20.sol";
import "./rsv/IRSV.sol";
import "./ownership/Ownable.sol";
import "./Basket.sol";

/**
 * A Proposal represents a suggestion to change the backing for RSV.
 *
 * The lifecycle of a proposal:
 * 1. Creation
 * 2. Acceptance
 * 3. Completion
 *
```

```
 * A time can be set during acceptance to determine when completion is eligible. A proposal can
 * also be cancelled before it is completed. If a proposal is cancelled, it can no longer become
 * Completed.
 *
 * This contract is intended to be used in one of two possible ways. Either:
 * - A target RSV basket is proposed, and quantities to be exchanged are deduced at the time of
 *   proposal execution.
 * - A specific quantity of tokens to be exchanged is proposed, and the resultant RSV basket is
 *   determined at the time of proposal execution.
 */

interface IProposal {
    function proposer() external returns(address);
    function accept(uint256 time) external;
    function cancel() external;
    function complete(IRSV rsv, Basket oldBasket) external returns(Basket);
    function nominateNewOwner(address newOwner) external;
    function acceptOwnership() external;
}

interface IProposalFactory {
    function createSwapProposal(address,
        address[] calldata tokens,
        uint256[] calldata amounts,
        bool[] calldata toVault
    ) external returns (IProposal);

    function createWeightProposal(address proposer, Basket basket) external returns (IProposal);
}

contract ProposalFactory is IProposalFactory {
    function createSwapProposal(
        address proposer,
        address[] calldata tokens,
        uint256[] calldata amounts,
        bool[] calldata toVault
    )
        external returns (IProposal)
    {
        IProposal proposal = IProposal(new SwapProposal(proposer, tokens, amounts, toVault));
        proposal.nominateNewOwner(msg.sender);
        return proposal;
    }

    function createWeightProposal(address proposer, Basket basket) external returns (IProposal) {
        IProposal proposal = IProposal(new WeightProposal(proposer, basket));
        proposal.nominateNewOwner(msg.sender);
        return proposal;
    }
}

```

```
contract Proposal is IProposal, Ownable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    uint256 public time;
    address public proposer;

    enum State { Created, Accepted, Cancelled, Completed }
    State public state;

    event ProposalCreated(address indexed proposer);
    event ProposalAccepted(address indexed proposer, uint256 indexed time);
    event ProposalCancelled(address indexed proposer);
    event ProposalCompleted(address indexed proposer, address indexed basket);

    /*@CTK Proposal
      @tag assume_completion
      @post __post.proposer == _proposer
      @post __post.state == State.Created
     */
    constructor(address _proposer) public {
        proposer = _proposer;
        state = State.Created;
        emit ProposalCreated(proposer);
    }

    /// Moves a proposal from the Created to Accepted state.
    /*@CTK accept
      @tag assume_completion
      @post msg.sender == _owner
      @post __post.time == _time
      @post __post.state == State.Accepted
     */
    function accept(uint256 _time) external onlyOwner {
        require(state == State.Created, "proposal not created");
        time = _time;
        state = State.Accepted;
        emit ProposalAccepted(proposer, _time);
    }

    /// Cancels a proposal if it has not been completed.
    /*@CTK cancel
      @tag assume_completion
      @post msg.sender == _owner
      @post __post.state == State.Cancelled
     */
    function cancel() external onlyOwner {
        require(state != State.Completed);
        state = State.Cancelled;
        emit ProposalCancelled(proposer);
    }

    /// Moves a proposal from the Accepted to Completed state.
    /// Returns the tokens, quantitiesIn, and quantitiesOut, required to implement the
        proposal.
    /*@CTK complete
      @tag assume_completion
      @post msg.sender == _owner
```

```
     @post __post.state == State.Accepted
     @post now > time
    */
   function complete(IRSV rsv, Basket oldBasket)
      external onlyOwner returns(Basket)
   {
      require(state == State.Accepted, "proposal must be accepted");
      require(now > time, "wait to execute");
      state = State.Completed;

      Basket b = _newBasket(rsv, oldBasket);
      emit ProposalCompleted(proposer, address(b));
      return b;
   }

   /// Returns the newly-proposed basket. This varies for different types of proposals,
   /// so it's abstract here.
   function _newBasket(IRSV trustedRSV, Basket oldBasket) internal returns(Basket);
}

/**
 * A WeightProposal represents a suggestion to change the backing for RSV to a new distribution
 * of tokens. You can think of it as designating what a _single RSV_ should be backed by, but
 * deferring on the precise quantities of tokens that will be need to be exchanged until a later
 * point in time.
 *
 * When this proposal is completed, it simply returns the target basket.
 */
contract WeightProposal is Proposal {
   Basket public trustedBasket;

   /*@CTK WeightProposal
     @tag assume_completion
     @post __post.trustedBasket == _trustedBasket
    */
   constructor(address _proposer, Basket _trustedBasket) Proposal(_proposer) public {
      require(_trustedBasket.size() > 0, "proposal cannot be empty");
      trustedBasket = _trustedBasket;
   }

   /// Returns the newly-proposed basket
   function _newBasket(IRSV, Basket) internal returns(Basket) {
      return trustedBasket;
   }
}

/**
 * A SwapProposal represents a suggestion to transfer fixed amounts of tokens into and out of the
 * vault. Whereas a WeightProposal designates how much a _single RSV_ should be backed by,
 * a SwapProposal first designates what quantities of tokens to transfer in total and then
 * solves for the new resultant basket later.
```

```
 *
 * When this proposal is completed, it calculates what the weights for the new basket will be
 * and returns it. If RSV supply is 0, this kind of Proposal cannot be used.
 */

// On "unit" comments, see comment at top of Manager.sol.
contract SwapProposal is Proposal {
    address[] public tokens;
    uint256[] public amounts; // unit: qToken
    bool[] public toVault;

    uint256 constant WEIGHT_SCALE = uint256(10)**18; // unit: aqToken / qToken

    /*@CTK SwapProposal
      @tag assume_completion
      @post _tokens.length > 0
      @post (_tokens.length == _amounts.length) && (_amounts.length == _toVault.length)
      @post __post.tokens == _tokens
      @post __post.amounts == _amounts
      @post __post.toVault == _toVault
     */
    constructor(address _proposer,
                address[] memory _tokens,
                uint256[] memory _amounts, // unit: qToken
                bool[] memory _toVault )
        Proposal(_proposer) public
    {
        require(_tokens.length > 0, "proposal cannot be empty");
        require(_tokens.length == _amounts.length && _amounts.length == _toVault.length,
                "unequal array lengths");
        tokens = _tokens;
        amounts = _amounts;
        toVault = _toVault;
    }

    /// Return the newly-proposed basket, based on the current vault and the old basket.
    /*@CTK SwapProposal_newBasket
      @tag assume_completion
      @post __pst.tokens.length <= 100
      @post __pst.weights.length <= 100
     */
    function _newBasket(IRSV trustedRSV, Basket trustedOldBasket) internal returns(Basket) {

        uint256[] memory weights = new uint256[](tokens.length);
        // unit: aqToken/RSV

        uint256 scaleFactor = WEIGHT_SCALE.mul(uint256(10)**(trustedRSV.decimals()));
        // unit: aqToken/qToken * qRSV/RSV

        uint256 rsvSupply = trustedRSV.totalSupply();
        // unit: qRSV

        /*@CTK loop_SwapProposal_newBasket
          @inv i <= tokens.length
```

```
        @post i == tokens.length
        @post !__should_return
       */
      for (uint256 i = 0; i < tokens.length; i++) {
         uint256 oldWeight = trustedOldBasket.weights(tokens[i]);
         // unit: aqToken/RSV

         if (toVault[i]) {
            // We require that the execution of a SwapProposal takes in no more than the funds
            // offered in its proposal -- that's part of the premise. It turns out that,
            // because we're rounding down _here_ and rounding up in
            // Manager._executeBasketShift(), it's possible for the naive implementation of
            // this mechanism to overspend the proposer's tokens by 1 qToken. We avoid that,
            // here, by making the effective proposal one less. Yeah, it's pretty fiddly.

            weights[i] = oldWeight.add( (amounts[i].sub(1)).mul(scaleFactor).div(rsvSupply) );
            //unit: aqToken/RSV == aqToken/RSV == [qToken] * [aqToken/qToken*qRSV/RSV] / [qRSV]
         } else {
            weights[i] = oldWeight.sub( amounts[i].mul(scaleFactor).div(rsvSupply) );
            //unit: aqToken/RSV
         }
      }

      return new Basket(trustedOldBasket, tokens, weights);
      // unit check for weights: aqToken/RSV
   }
}
```

File Basket.sol

```
pragma solidity 0.5.7;


/**
 * This Basket contract is essentially just a data structure; it represents the tokens and weights
 * in some Reserve-backing basket, either proposed or accepted.
 *
 * @dev Each `weights` value is an integer, with unit aqToken/RSV. (That is, atto-quantum-Tokens
 * per RSV). If you prefer, you can think about this as if the weights value is itself an
 * 18-decimal fixed-point value with unit qToken/RSV. (It would be prettier if these were just
 * straightforwardly qTokens/RSV, but that introduces unacceptable rounding error in some of our
 * basket computations.)
 *
 * @dev For example, let's say we have the token USDX in the vault, and it's represented to 6
 * decimal places, and the RSV basket should include 3/10ths of a USDX for each RSV.
```

```
    Then the
 * corresponding basket weight will be represented as 3*(10**23), because:
 *
 * @dev 3*(10**23) aqToken/RSV == 0.3 Token/RSV * (10**6 qToken/Token) * (10**18 aqToken/qToken)
 *
 * @dev For further notes on units, see the header comment for Manager.sol.
*/

contract Basket {
    address[] public tokens;
    mapping(address => uint256) public weights; // unit: aqToken/RSV
    mapping(address => bool) public has;
    // INVARIANT: {addr | addr in tokens} == {addr | has[addr] == true}

    // SECURITY PROPERTY: The value of prev is always a Basket, and cannot be set by any user.

    // WARNING: A basket can be of size 0. It is the Manager's responsibility
    //                 to ensure Issuance does not happen against an empty basket.

    /// Construct a new basket from an old Basket `prev`, and a list of tokens and weights with
    /// which to update `prev`. If `prev == address(0)`, act like it's an empty basket.
    /*@CTK Basket
      @tag assume_completion
      @post tokens.length <= 10
     */
    constructor(Basket trustedPrev, address[] memory _tokens, uint256[] memory _weights) public {
        require(_tokens.length == _weights.length, "Basket: unequal array lengths");

        // Initialize data from input arrays
        tokens = new address[](_tokens.length);
        /*@CTK loop_Basket
          @inv i <= _tokens.length
          @inv forall j: uint. (j >= 0 /\ j < i) ->
          @inv forall j: uint. (j >= 0 /\ j < i) -> weights[j] == _weights[j]
          @inv forall j: uint. (j >= 0 /\ j < i) -> tokens[j] == _tokens[j]
          @inv forall j: uint. (j >= 0 /\ j < i) -> has[_tokens[j]] == true
          @post i == _tokens.length
          @post !__should_return
         */
        for (uint256 i = 0; i < _tokens.length; i++) {
            require(!has[_tokens[i]], "duplicate token entries");
            weights[_tokens[i]] = _weights[i];
            has[_tokens[i]] = true;
            tokens[i] = _tokens[i];
        }

        // If there's a previous basket, copy those of its contents not already set.
        if (trustedPrev != Basket(0)) {
            /*@CTK loop_Basket_trustedPrev
              @inv i <= trustedPrev.size()
              @inv forall j: uint. (j >= 0 /\ j < i /\ !has[trustedPrev.tokens(j)]) -> has[trustedPrev.tokens(j)] == true
              @inv forall j: uint. (j >= 0 /\ j < i /\ !has[trustedPrev.tokens(j)]) ->
```

```
                weights[trustedPrev.tokens(j)] == trustedPrev.weights(trustedPrev.
                tokens(j))
            @post !__should_return
           */
        for (uint256 i = 0; i < trustedPrev.size(); i++) {
            address tok = trustedPrev.tokens(i);
            if (!has[tok]) {
                weights[tok] = trustedPrev.weights(tok);
                has[tok] = true;
                tokens.push(tok);
            }
        }
    }
    require(tokens.length <= 10, "Basket: bad length");
  }

  /*@CTK getTokens
    @tag assume_completion
    @post __return == tokens
   */
  function getTokens() external view returns(address[] memory) {
    return tokens;
  }

  /*@CTK size
    @tag assume_completion
    @post __return == tokens.length
   */
  function size() external view returns(uint256) {
    return tokens.length;
  }
}
```

File zeppelin/math/SafeMath.sol

```
pragma solidity 0.5.7;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
  /**
   * @dev Returns the addition of two unsigned integers, reverting on
   * overflow.
   *
   * Counterpart to Solidity's `+` operator.
   *
   * Requirements:
   * - Addition cannot overflow.
```

```
 */
/*@CTK "SafeMath add"
  @tag spec
  @tag is_pure
  @post (a + b < a || a + b < b) == __reverted
  @post !__reverted -> __return == a + b
  @post !__reverted -> !__has_overflow
  @post !__reverted -> !__has_assertion_failure
  @post !(__has_buf_overflow)
*/
function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a, "SafeMath: addition overflow");

    return c;
}

/**
 * @dev Returns the subtraction of two unsigned integers, reverting on
 * overflow (when the result is negative).
 *
 * Counterpart to Solidity's `-` operator.
 *
 * Requirements:
 * - Subtraction cannot overflow.
 */
/*@CTK "SafeMath sub"
  @tag spec
  @tag is_pure
  @post (b > a) == __reverted
  @post !__reverted -> __return == a - b
  @post !__reverted -> !__has_overflow
  @post !__reverted -> !__has_assertion_failure
  @post !(__has_buf_overflow)
*/
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b <= a, "SafeMath: subtraction overflow");
    uint256 c = a - b;

    return c;
}

/**
 * @dev Returns the multiplication of two unsigned integers, reverting on
 * overflow.
 *
 * Counterpart to Solidity's `*` operator.
 *
 * Requirements:
 * - Multiplication cannot overflow.
 */
/*@CTK "SafeMath mul zero"
  @tag spec
  @tag is_pure
  @pre (a == 0)
  @post __return == 0
*/
/*@CTK "SafeMath mul nonzero"
```

```
     @tag spec
     @tag is_pure
     @pre (a != 0)
     @post (a * b / a != b) == __reverted
     @post !__reverted -> __return == a * b
     @post !__reverted -> !__has_overflow
     @post !__reverted -> !__has_assertion_failure
     @post !(__has_buf_overflow)
   */
   function mul(uint256 a, uint256 b) internal pure returns (uint256) {
       // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
       // benefit is lost if 'b' is also tested.
       // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
       if (a == 0) {
           return 0;
       }

       uint256 c = a * b;
       require(c / a == b, "SafeMath: multiplication overflow");

       return c;
   }

   /**
    * @dev Returns the integer division of two unsigned integers. Reverts on
    * division by zero. The result is rounded towards zero.
    *
    * Counterpart to Solidity's `/` operator. Note: this function uses a
    * `revert` opcode (which leaves remaining gas untouched) while Solidity
    * uses an invalid opcode to revert (consuming all remaining gas).
    *
    * Requirements:
    * - The divisor cannot be zero.
    */
   /*@CTK "SafeMath div"
     @tag spec
     @tag is_pure
     @post (b == 0) == __reverted
     @post !__reverted -> __return == a / b
     @post !__reverted -> !__has_overflow
     @post !__reverted -> !__has_assertion_failure
     @post !(__has_buf_overflow)
   */
   function div(uint256 a, uint256 b) internal pure returns (uint256) {
       // Solidity only automatically asserts when dividing by 0
       require(b > 0, "SafeMath: division by zero");
       uint256 c = a / b;
       // assert(a == b * c + a % b); // There is no case in which this doesn't hold

       return c;
   }

   /**
    * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
        modulo),
    * Reverts when dividing by zero.
    *
    * Counterpart to Solidity's `%` operator. This function uses a `revert`
```

```
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    /*@CTK "SafeMath mod"
      @tag spec
      @tag is_pure
      @post (b == 0) == __reverted
      @post !__reverted -> __return == a % b
      @post !__reverted -> !__has_overflow
      @post !__reverted -> !__has_assertion_failure
      @post !(__has_buf_overflow)
    */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0, "SafeMath: modulo by zero");
        return a % b;
    }
}
```

File ownership/Ownable.sol

```
pragma solidity 0.5.7;

import "../zeppelin/GSN/Context.sol";
/**
 * @dev Contract module which provides a basic access control mechanism, where there is an account
 * (owner) that can be granted exclusive access to specific functions.
 *
 * This module is used through inheritance by using the modifier `onlyOwner`.
 *
 * To change ownership, use a 2-part nominate-accept pattern.
 *
 * This contract is loosely based off of https://git.io/JenNF but additionally requires new owners
 * to accept ownership before the transition occurs.
 */
contract Ownable is Context {
    address private _owner;
    address private _nominatedOwner;

    event NewOwnerNominated(address indexed previousOwner, address indexed nominee);
    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    /*@CTK Ownable
      @tag assume_completion
      @post __post._owner == msg.sender
     */
    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }
```

```

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }

    /**
     * @dev Returns the address of the current nominated owner.
     */
    function nominatedOwner() external view returns (address) {
        return _nominatedOwner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        _onlyOwner();
        _;
    }

    function _onlyOwner() internal view {
        require(_msgSender() == _owner, "caller is not owner");
    }

    /**
     * @dev Nominates a new owner `newOwner`.
     * Requires a follow-up `acceptOwnership`.
     * Can only be called by the current owner.
     */
    /*@CTK nominateNewOwner
      @tag assume_completion
      @post msg.sender == _owner
      @post newOwner != address(0)
      @post __post._nominatedOwner == newOwner
     */
    function nominateNewOwner(address newOwner) external onlyOwner {
        require(newOwner != address(0), "new owner is 0 address");
        emit NewOwnerNominated(_owner, newOwner);
        _nominatedOwner = newOwner;
    }

    /**
     * @dev Accepts ownership of the contract.
     */
    /*@CTK acceptOwnership
      @tag assume_completion
      @post msg.sender == _nominatedOwner
      @post __post._owner == _nominatedOwner
     */
    function acceptOwnership() external {
        require(_nominatedOwner == _msgSender(), "unauthorized");
        emit OwnershipTransferred(_owner, _nominatedOwner);
        _owner = _nominatedOwner;
    }

```

```
    /** Set `_owner` to the 0 address.
     * Only do this to deliberately lock in the current permissions.
     */
    /*@CTK renounceOwnership
      @tag assume_completion
      @post __post._owner == address(0)
     */
    function renounceOwnership() external onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }
}
```

File rsv/ReserveEternalStorage.sol

```
pragma solidity 0.5.7;

import "../ownership/Ownable.sol";
import "../zeppelin/math/SafeMath.sol";

/**
 * @title Eternal Storage for the Reserve Token
 *
 * @dev Eternal Storage facilitates future upgrades.
 *
 * If Reserve chooses to release an upgraded contract for the Reserve in the future, Reserve will
 * have the option of reusing the deployed version of this data contract to simplify migration.
 *
 * The use of this contract does not imply that Reserve will choose to do a future upgrade, nor
 * that any future upgrades will necessarily re-use this storage. It merely provides option value.
 */
contract ReserveEternalStorage is Ownable {

    using SafeMath for uint256;


    // ===== auth =====

    address public reserveAddress;

    event ReserveAddressTransferred(
        address indexed oldReserveAddress,
        address indexed newReserveAddress
    );

    /// On construction, set auth fields.
    /*@CTK ReserveEternalStorage
      @tag assume_completion
      @post __post.reserveAddress == msg.sender
     */
    constructor() public {
        reserveAddress = _msgSender();
        emit ReserveAddressTransferred(address(0), reserveAddress);
    }

```

```
    /// Only run modified function if sent by `reserveAddress`.
    modifier onlyReserveAddress() {
        require(_msgSender() == reserveAddress, "onlyReserveAddress");
        _;
    }

    /// Set `reserveAddress`.
    /*@CTK updateReserveAddress
      @tag assume_completion
      @post newReserveAddress != address(0)
      @post (msg.sender == reserveAddress) || (msg.sender == _owner)
      @post __post.reserveAddress == newReserveAddress
     */
    function updateReserveAddress(address newReserveAddress) external {
        require(newReserveAddress != address(0), "zero address");
        require(_msgSender() == reserveAddress || _msgSender() == owner(), "not
            authorized");
        emit ReserveAddressTransferred(reserveAddress, newReserveAddress);
        reserveAddress = newReserveAddress;
    }



    // ===== balance =====

    mapping(address => uint256) public balance;

    /// Add `value` to `balance[key]`, unless this causes integer overflow.
    ///
    /// @dev This is a slight divergence from the strict Eternal Storage pattern, but
        it reduces
    /// the gas for the by-far most common token usage, it's a *very simple*
        divergence, and
    /// `setBalance` is available anyway.
    /*@CTK addBalance
      @tag assume_completion
      @post msg.sender == reserveAddress
      @post __post.balance[key] == balance[key] + value
     */
    function addBalance(address key, uint256 value) external onlyReserveAddress {
        balance[key] = balance[key].add(value);
    }

    /// Subtract `value` from `balance[key]`, unless this causes integer underflow.
    /*@CTK subBalance
      @tag assume_completion
      @post (msg.sender == reserveAddress)
      @post __post.balance[key] == (balance[key] - value)
     */
    function subBalance(address key, uint256 value) external onlyReserveAddress {
        balance[key] = balance[key].sub(value);
    }

    /// Set `balance[key]` to `value`.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK setBalance
```

```
     @tag assume_completion
     @post msg.sender == reserveAddress
     @post __post.balance[key] == value
    */
   function setBalance(address key, uint256 value) external onlyReserveAddress {
      balance[key] = value;
   }



   // ===== allowed =====

   mapping(address => mapping(address => uint256)) public allowed;

   /// Set `to`'s allowance of `from`'s tokens to `value`.
   //@CTK NO_OVERFLOW
   //@CTK NO_BUF_OVERFLOW
   //@CTK NO_ASF
   /*@CTK setAllowed
     @tag assume_completion
     @post msg.sender == reserveAddress
     @post __post.allowed[from][to] == value
    */
   function setAllowed(address from, address to, uint256 value) external
       onlyReserveAddress {
      allowed[from][to] = value;
   }
}
```

File rsv/Reserve.sol

```
pragma solidity 0.5.7;

import "../zeppelin/token/ERC20/IERC20.sol";
import "../zeppelin/math/SafeMath.sol";
import "../ownership/Ownable.sol";
import "./ReserveEternalStorage.sol";

/**
 * @title An interface representing a contract that calculates transaction fees
 */
 interface ITXFee {
    function calculateFee(address from, address to, uint256 amount) external returns
        (uint256);
 }

/**
 * @title The Reserve Token
 * @dev An ERC-20 token with minting, burning, pausing, and user freezing.
 * Based on OpenZeppelin's [implementation](https://github.com/OpenZeppelin/
     openzeppelin-solidity/blob/41aa39afbc13f0585634061701c883fe512a5469/contracts/
     token/ERC20/ERC20.sol).
 *
 * Non-constant-sized data is held in ReserveEternalStorage, to facilitate potential
     future upgrades.
 */
contract Reserve is IERC20, Ownable {
   using SafeMath for uint256;

```

```
    // ==== State ====


    // Non-constant-sized data
    ReserveEternalStorage internal trustedData;

    // TX Fee helper contract
    ITXFee public trustedTxFee;

    // Basic token data
    uint256 public totalSupply;
    uint256 public maxSupply;

    // Paused data
    bool public paused;

    // Auth roles
    address public minter;
    address public pauser;
    address public feeRecipient;


    // ==== Events, Constants, and Constructor ====


    // Auth role change events
    event MinterChanged(address indexed newMinter);
    event PauserChanged(address indexed newPauser);
    event FeeRecipientChanged(address indexed newFeeRecipient);
    event MaxSupplyChanged(uint256 indexed newMaxSupply);
    event EternalStorageTransferred(address indexed newReserveAddress);
    event TxFeeHelperChanged(address indexed newTxFeeHelper);

    // Pause events
    event Paused(address indexed account);
    event Unpaused(address indexed account);

    // Basic information as constants
    string public constant name = "Reserve";
    string public constant symbol = "RSV";
    uint8 public constant decimals = 18;

    /// Initialize critical fields.
    /*@CTK Reserve
      @tag assume_completion
      @post __post.pauser == msg.sender
      @post __post.feeRecipient == msg.sender
      @post __post.maxSupply == 2**256 - 1
      @post __post.paused = true
      @post __post.trustedTxFee = ITXFee(address(0))
     */
    constructor() public {
        pauser = msg.sender;
        feeRecipient = msg.sender;
        // minter defaults to the zero address.

        maxSupply = 2 ** 256 - 1;
```

```
        paused = true;

        trustedTxFee = ITXFee(address(0));
        trustedData = new ReserveEternalStorage();
        trustedData.nominateNewOwner(msg.sender);
    }

    /// Accessor for eternal storage contract address.
    function getEternalStorageAddress() external view returns(address) {
        return address(trustedData);
    }


    // ==== Admin functions ====


    /// Modifies a function to only run if sent by `role`.
    modifier only(address role) {
        require(msg.sender == role, "unauthorized: not role holder");
        _;
    }

    /// Modifies a function to only run if sent by `role` or the contract's `owner`.
    modifier onlyOwnerOr(address role) {
        require(msg.sender == owner() || msg.sender == role, "unauthorized: not owner
            or role");
        _;
    }

    /// Change who holds the `minter` role.
    /*@CTK changeMinter
      @tag assume_completion
      @post (msg.sender == _owner) || (msg.sender == minter)
      @post __post.minter == newMinter
     */
    function changeMinter(address newMinter) external onlyOwnerOr(minter) {
        minter = newMinter;
        emit MinterChanged(newMinter);
    }

    /// Change who holds the `pauser` role.
    /*@CTK changePauser
      @tag assume_completion
      @post (msg.sender == _owner) || (msg.sender == pauser)
      @post __post.pauser == newPauser
     */
    function changePauser(address newPauser) external onlyOwnerOr(pauser) {
        pauser = newPauser;
        emit PauserChanged(newPauser);
    }

    /*@CTK changePauser
      @tag assume_completion
      @post (msg.sender == _owner) || (msg.sender == feeRecipient)
      @post __post.feeRecipient == newFeeRecipient
     */
    function changeFeeRecipient(address newFeeRecipient) external onlyOwnerOr(
        feeRecipient) {
```

```
        feeRecipient = newFeeRecipient;
        emit FeeRecipientChanged(newFeeRecipient);
    }

    /// Make a different address the EternalStorage contract's reserveAddress.
    /// This will break this contract, so only do it if you're
    /// abandoning this contract, e.g., for an upgrade.
    /*@CTK transferEternalStorage
      @tag assume_completion
      @post msg.sender == _owner
      @post newReserveAddress != address(0)
      @post paused == true
     */
    function transferEternalStorage(address newReserveAddress) external onlyOwner
        isPaused {
        require(newReserveAddress != address(0), "zero address");
        emit EternalStorageTransferred(newReserveAddress);
        trustedData.updateReserveAddress(newReserveAddress);
    }

    /// Change the contract that helps with transaction fee calculation.
    /*@CTK changeTxFeeHelper
      @tag assume_completion
      @post msg.sender == _owner
     */
    function changeTxFeeHelper(address newTrustedTxFee) external onlyOwner {
        trustedTxFee = ITXFee(newTrustedTxFee);
    }

    /// Change the maximum supply allowed.
    /*@CTK changeMaxSupply
      @tag assume_completion
      @post (msg.sender == _owner)
      @post __post.maxSupply == newMaxSupply
     */
    function changeMaxSupply(uint256 newMaxSupply) external onlyOwner {
        maxSupply = newMaxSupply;
        emit MaxSupplyChanged(newMaxSupply);
    }

    /// Pause the contract.
    /*@CTK pause
      @tag assume_completion
      @post (msg.sender == pauser)
      @post __post.paused == true
     */
    function pause() external only(pauser) {
        paused = true;
        emit Paused(pauser);
    }

    /// Unpause the contract.
    /*@CTK pause
      @tag assume_completion
      @post (msg.sender == pauser)
      @post __post.paused == false
     */
    function unpause() external only(pauser) {
```

```
        paused = false;
        emit Unpaused(pauser);
    }

    /// Modifies a function to run only when the contract is paused.
    modifier isPaused() {
        require(paused, "contract is not paused");
        _;
    }

    /// Modifies a function to run only when the contract is not paused.
    modifier notPaused() {
        require(!paused, "contract is paused");
        _;
    }


    // ==== Token transfers, allowances, minting, and burning ====


    /// @return how many attoRSV are held by `holder`.
    function balanceOf(address holder) external view returns (uint256) {
        return trustedData.balance(holder);
    }

    /// @return how many attoRSV `holder` has allowed `spender` to control.
    function allowance(address holder, address spender) external view returns (uint256
        ) {
        return trustedData.allowed(holder, spender);
    }

    /// Transfer `value` attoRSV from `msg.sender` to `to`.
    /*@CTK transfer
      @tag assume_completion
      @post paused == false
     */
    function transfer(address to, uint256 value)
        external
        notPaused
        returns (bool)
    {
        _transfer(msg.sender, to, value);
        return true;
    }

    /**
     * Approve `spender` to spend `value` attotokens on behalf of `msg.sender`.
     *
     * Beware that changing a nonzero allowance with this method brings the risk that
     * someone may use both the old and the new allowance by unfortunate transaction
       ordering. One
     * way to mitigate this risk is to first reduce the spender's allowance
     * to 0, and then set the desired value afterwards, per
     * [this ERC-20 issue](https://github.com/ethereum/EIPs/issues/20#issuecomment
       -263524729).
     *
     * A simpler workaround is to use `increaseAllowance` or `decreaseAllowance`,
       below.
```

```
     *
     * @param spender address The address which will spend the funds.
     * @param value uint256 How many attotokens to allow `spender` to spend.
     */
    /*@CTK approve
      @tag assume_completion
      @post paused == false
     */
    function approve(address spender, uint256 value)
        external
        notPaused
        returns (bool)
    {
        _approve(msg.sender, spender, value);
        return true;
    }

    /// Transfer approved tokens from one address to another.
    /// @param from address The address to send tokens from.
    /// @param to address The address to send tokens to.
    /// @param value uint256 The number of attotokens to send.
    /*@CTK approve
      @tag assume_completion
      @post paused == false
     */
    function transferFrom(address from, address to, uint256 value)
        external
        notPaused
        returns (bool)
    {
        _transfer(from, to, value);
        _approve(from, msg.sender, trustedData.allowed(from, msg.sender).sub(value));
        return true;
    }

    /// Increase `spender`'s allowance of the sender's tokens.
    /// @dev From MonolithDAO Token.sol
    /// @param spender The address which will spend the funds.
    /// @param addedValue How many attotokens to increase the allowance by.
    /*@CTK increaseAllowance
      @tag assume_completion
      @post paused == false
     */
    function increaseAllowance(address spender, uint256 addedValue)
        external
        notPaused
        returns (bool)
    {
        _approve(msg.sender, spender, trustedData.allowed(msg.sender, spender).add(
            addedValue));
        return true;
    }

    /// Decrease `spender`'s allowance of the sender's tokens.
    /// @dev From MonolithDAO Token.sol
    /// @param spender The address which will spend the funds.
    /// @param subtractedValue How many attotokens to decrease the allowance by.
    /*@CTK decreaseAllowance
```

```
     @tag assume_completion
     @post paused == false
    */
   function decreaseAllowance(address spender, uint256 subtractedValue)
       external
       notPaused
       returns (bool)
   {
       _approve(
           msg.sender,
           spender,
           trustedData.allowed(msg.sender, spender).sub(subtractedValue)
       );
       return true;
   }

   /// Mint `value` new attotokens to `account`.
   /*@CTK mint
     @tag assume_completion
     @post paused == false
     @post msg.sender == minter
     @post account != address(0)
     @post __post.totalSupply == totalSupply + value
     @post __post.totalSupply < maxSupply
    */
   function mint(address account, uint256 value)
       external
       notPaused
       only(minter)
   {
       require(account != address(0), "can't mint to address zero");

       totalSupply = totalSupply.add(value);
       require(totalSupply < maxSupply, "max supply exceeded");
       trustedData.addBalance(account, value);
       emit Transfer(address(0), account, value);
   }

   /// Burn `value` attotokens from `account`, if sender has that much allowance from
        `account`.
   /*@CTK burnFrom
     @tag assume_completion
     @post paused == false
     @post msg.sender == minter
    */
   function burnFrom(address account, uint256 value)
       external
       notPaused
       only(minter)
   {
       _burn(account, value);
       _approve(account, msg.sender, trustedData.allowed(account, msg.sender).sub(
           value));
   }

   /// @dev Transfer of `value` attotokens from `from` to `to`.
   /// Internal; doesn't check permissions.
   /*@CTK _transfer
```

```
     @tag assume_completion
     @post to != address(0)
    */
   function _transfer(address from, address to, uint256 value) internal {
      require(to != address(0), "can't transfer to address zero");
      trustedData.subBalance(from, value);
      uint256 fee = 0;

      if (address(trustedTxFee) != address(0)) {
         fee = trustedTxFee.calculateFee(from, to, value);
         require(fee <= value, "transaction fee out of bounds");

         trustedData.addBalance(feeRecipient, fee);
         emit Transfer(from, feeRecipient, fee);
      }

      trustedData.addBalance(to, value.sub(fee));
      emit Transfer(from, to, value.sub(fee));
   }

   /// @dev Burn `value` attotokens from `account`.
   /// Internal; doesn't check permissions.
   /*@CTK _burn
     @tag assume_completion
     @post account != address(0)
     @post __post.totalSupply == totalSupply - value
    */
   function _burn(address account, uint256 value) internal {
      require(account != address(0), "can't burn from address zero");

      totalSupply = totalSupply.sub(value);
      trustedData.subBalance(account, value);
      emit Transfer(account, address(0), value);
   }

   /// @dev Set `spender`'s allowance on `holder`'s tokens to `value` attotokens.
   /// Internal; doesn't check permissions.
   /*@CTK _approve
     @tag assume_completion
     @post spender != address(0)
     @post holder != address(0)
    */
   function _approve(address holder, address spender, uint256 value) internal {
      require(spender != address(0), "spender cannot be address zero");
      require(holder != address(0), "holder cannot be address zero");

      trustedData.setAllowed(holder, spender, value);
      emit Approval(holder, spender, value);
   }
}
```