CertiK Audit Report For Band Protocol

Request Date: 2019-06-18
Revision Date: 2019-08-07
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 Band Protocol(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/

Exective Summary

This report has been prepared as product of the Smart Contract Audit request by Band Protocol. This audit was conducted to discover issues and vulnerabilities in the source code of Band Protocol's Smart Contracts. Utilizing CertiK's Formal Verification Platform, Static Analysis and Manual Review, a comprehensive examination has been performed. The auditing process pays special attention to the following considerations.

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessment of the codebase for 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

For every issues found, CertiK categorizes them into 3 buckets based on its risk level:

Critical

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

Medium

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

Low

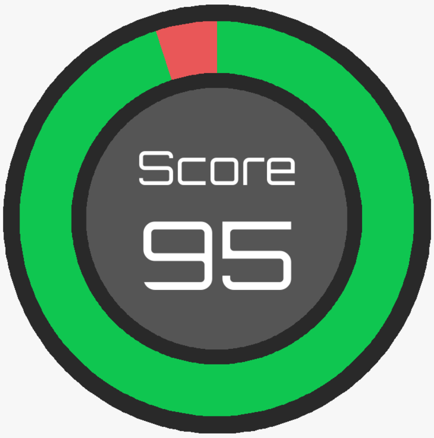
The code implementation is not a best practice, or use a suboptimal design pattern, which may lead to security vulnerability, but no concern found yet.

Testing Summary

PASS

CERTIK *believes this smart contract passes security qualifications to be listed on digital asset exchanges.*

Aug 07, 2019



Type of Issues

CertiK smart label engine applied 100% coveraged formal verification labels on the source code, and scanned the code using our proprietary static analysis and formal verification engine to detect the follow type of issues.

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	1	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.	1	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

Commit $63f7fd3d1b49b6949e73e57fe4083de13fd443e4$[1]

- **BandRegistry.sol**
  `a58c10f55898db35f2ba0bb58e087b0eb641503c993c3fb401cb3743b284ce59`

- **BandToken.sol**
  `37af21c307045dfc8294f2d75c94f9759c17ca5f3471f5b65a6e03b9fc7a68ca`

- **CommunityToken.sol**
  `3817984b759ca5ab65522194d2fc287f7c705af7cac63416ea360070704fbcbc`

- **Migrations.sol**
  `be83a8399a278fbf5a19e859ba87dfbbeba589bd2c520146290c2b0aeb913f11`

- **Parameters.sol**
  `97a1de77e12fbb0f76a342fac0baa0ba503c631a9c58670516bae02b7dfa1ecc`

- **AggTCD.sol**
  `4b697fc13dc3b15da83bb334189252aed831eca70958de60360efc7b749dbaf1`

- **MultiSigTCD.sol**
  `837ddd135843b5c58654fd36aa9da9a0efb451d52cd54cb3708ae44d66b8c9fa`

- **OffchainAggTCD.sol**
  `fb70bc1aceca9e11efb74d4c20cd9bc7d3b2b90b9fdd30686c6007795ab2a96b`

- **QueryInterface.sol**
  `f107ee7e630d1fa3a1b53383aac6d189322b95bcad3c8b27057465b8a4ec108f`

- **QueryTCR.sol**
  `58e79bf1ad891892d33c39c39f3889ed2c6b901b636efa89cf5572be81f3ebfc`

- **TCDBase.sol**
  `e62004e4cc439505170ca05e9291d42254024ac51334b9d3ee6f3c6b66548597`

- **TCRBase.sol**
  `f2ce5d1f585a01d22eed4334e2d82ce42b9119f7a500c9559281882154dc43ac`

- **WhiteListInterface.sol**
  `074d4c0b3a44b843320bbb1a1d7ed4bf25a78d59f4ed0426d767e04c2aa835d1`

- **WhiteListTCR.sol**
  `31e7b7bf9ecff4307ae1da480e476a343755fa8b8762a1497d537dd2ebc0c642`

- **BandExchangeInterface.sol**
  `b3a0a09043192243faf5b73bc762eff975cdcb3902a9d5051278e29d305bd951`

- **BondingCurve.sol**
  `d0fe45bdce98cdfe972d3fc969c5780232f8e8886729e977b83e6b9da3e45dfc`

[1] Band Protocol Smart Contracts: `https://github.com/bandprotocol/contracts`

- **ERC20Acceptor.sol**
  `e42984fbd0d0e44a7660db784728a89fbb28584ab1d4be2a3292c6e13785b9d5`
- **ERC20Base.sol**
  `0ef237ef620962ef389f389cc1b4ecf4dedced0634cce345ef881524c90b0b09`
- **ERC20Interface.sol**
  `186709d6c8502daadfc62453d2fd2e43165007e554ff50d8774692434efcfaee`
- **LockableToken.sol**
  `cb477faa8c59c3520994c602015097b5e162cc63716d2471379b86310f6e06a1`
- **SnapshotToken.sol**
  `f6f590d3ad84dbd317578d57ee209e2303d74b442d551d67e48d59397e88088f`
- **VestingWallet.sol**
  `8d137a75e07aa663bf4a4a44c326a55af415b850ef5dfea76ca84b168c2e95b7`
- **Aggregator.sol**
  `57ca85fd4fca4ae7b49da06b653879aad88496a1164807b7d9bd1e1591603ccd`
- **Equation.sol**
  `4fdb9cf89fc8361b53a0c108bb75361f788d8551870122773b23246bf67fe762`
- **Expression.sol**
  `c11bb27aa89731c323b2c7cc3e3458446f5d2561da801d12ed66c3196531177e`
- **Fractional.sol**
  `346d44caf4f91389b8aed20e50980629bb26a9bca37666bd84a6a28d1659159e`
- **AggTCDFactory.sol**
  `56ac4f0e13357b5c0844d677c80d3842c7751fc22e066ca14e2339dfcab1e1a5`
- **BondingCurveFactory.sol**
  `06377c0e493fcb8d1b0043e99a24bf81c681fbd68c8a7a0f3339fcea6260cc07`
- **CommunityFactory.sol**
  `d82af2a8fbc211d74223c77584817fe9eac5cfc8ea6369632f707a3647f0899a`
- **CommunityTokenFactory.sol**
  `d540f994b9612b449f6615cfabc097bde33738b6f22cfb6f7b545127c345a7bf`
- **MultiSigTCDFactory.sol**
  `d539ad3a18d5b4b4a834598652a7c9c753920ced23c94b6ecce2d6f12069141c`
- **OffchainAggTCDFactory.sol**
  `4f2d84dede0bb4e40991e336fbeb0c498137e9430d4284fedc64b88e214b1890`
- **ParametersFactory.sol**
  `3bdcf09d2da9a0502ff1345a870e1c34ff2d0cf268497b79330dde9b1e448cec`
- **TCRFactory.sol**
  `ca584e4116c2771731c4685394f79e019802d9e5ba40df48beeabdc16f043aff`

- **BandMockExchange.sol**
  `75858c5ffe11d20040260a03ef6294d1efb6d469e6dac168c0f51097d22c7753`
- **BondingCurveMock.sol**
  `31e3f17c76dd410ca3a7fa24d87fe02f2ff1336c65f8385690926e1635276c61`
- **EquationMock.sol**
  `ddec50dc9e211a89e0421fda60b1084a00948f0e71e4142ed8fcf95fa31e6a41`
- **MockDataSource.sol**
  `8cff62634e99d8dd7c120822920b9a7a86abec53b6d6e8a0b9d6970c14faeccc`
- **MultiSigWalletFactory.sol**
  `647136ae34bcecc47c81d7f176ffb40b17eea321312b1abb5e844370cfba59c3`
- **QueryTCDMock.sol**
  `d62a98e6667cb13eab2081211e052c81404ade7a70d9140975e13e850bb64375`
- **TCDListMock.sol**
  `635aa619a0c7c48305ee8c9ed64939b6a9a733bce0c574fa39098d09c69b8276`

Summary

CertiK was chosen by Band Protocol team to audit the design and implementation of its soon to be released 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.

Our client Band has demonstrated their professional and knowledgeable understanding of the project Band Protocol, 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.

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 contracts are 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.

Documentation

CertiK used the following sources of truth about how Band Protocol should work:

1. Band Protocol Website
2. Test Scenarios
3. Band Protocol Github Code Base.

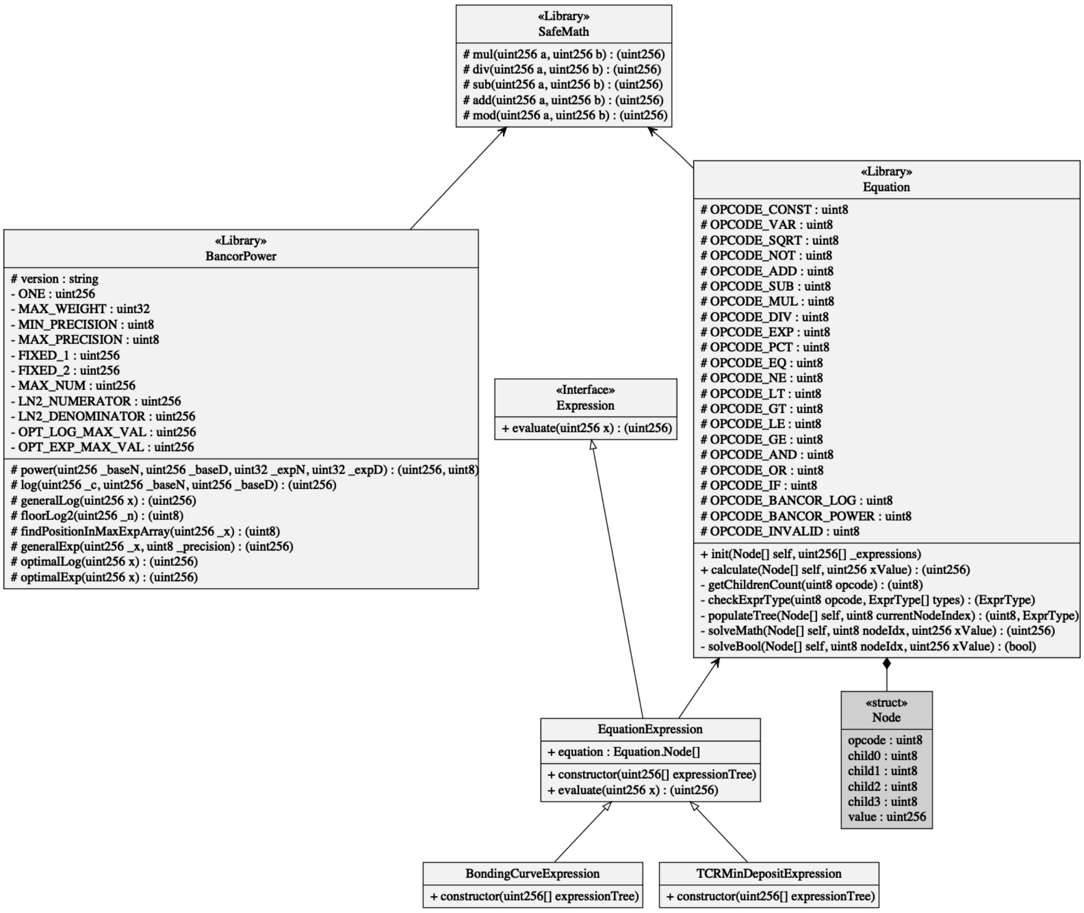

«Library»
SafeMath
# mul(uint256 a, uint256 b) : (uint256)
# div(uint256 a, uint256 b) : (uint256)
# sub(uint256 a, uint256 b) : (uint256)
# add(uint256 a, uint256 b) : (uint256)
# mod(uint256 a, uint256 b) : (uint256)
«Library»
Equation
# OPCODE_CONST : uint8
# OPCODE_VAR : uint8
# OPCODE_SQRT : uint8
# OPCODE_NOT : uint8
# OPCODE_ADD : uint8
# OPCODE_SUB : uint8
# OPCODE_MUL : uint8
# OPCODE_DIV : uint8
# OPCODE_EXP : uint8
# OPCODE_PCT : uint8
# OPCODE_EQ : uint8
# OPCODE_NE : uint8
# OPCODE_LT : uint8
# OPCODE_GT : uint8
# OPCODE_LE : uint8
# OPCODE_GE : uint8
# OPCODE_AND : uint8
# OPCODE_OR : uint8
# OPCODE_IF : uint8
# OPCODE_BANCOR_LOG : uint8
# OPCODE_BANCOR_POWER : uint8
# OPCODE_INVALID : uint8
+ init(Node[] self, uint256[] _expressions)
+ calculate(Node[] self, uint256 xValue) : (uint256)
- getChildrenCount(uint8 opcode) : (uint8)
- checkExprType(uint8 opcode, ExprType[] types) : (ExprType)
- populateTree(Node[] self, uint8 currentNodeIndex) : (uint8, ExprType)
- solveMath(Node[] self, uint8 nodeIdx, uint256 xValue) : (uint256)
- solveBool(Node[] self, uint8 nodeIdx, uint256 xValue) : (bool)
«Library»
BancorPower
# version : string
- ONE : uint256
- MAX_WEIGHT : uint32
- MIN_PRECISION : uint8
- MAX_PRECISION : uint8
- FIXED_1 : uint256
- FIXED_2 : uint256
- MAX_NUM : uint256
- LN2_NUMERATOR : uint256
- LN2_DENOMINATOR : uint256
- OPT_LOG_MAX_VAL : uint256
- OPT_EXP_MAX_VAL : uint256
# power(uint256 _baseN, uint256 _baseD, uint32 _expN, uint32 _expD) : (uint256, uint8)
# log(uint256 _c, uint256 _baseN, uint256 _baseD) : (uint256)
# generalLog(uint256 x) : (uint256)
# floorLog2(uint256 _n) : (uint8)
# findPositionInMaxExpArray(uint256 _x) : (uint8)
# generalExp(uint256 _x, uint8 _precision) : (uint256)
# optimalLog(uint256 x) : (uint256)
# optimalExp(uint256 x) : (uint256)
«Interface»
Expression
+ evaluate(uint256 x) : (uint256)
EquationExpression
+ equation : Equation.Node[]
+ constructor(uint256[] expressionTree)
+ evaluate(uint256 x) : (uint256)
«struct»
Node
opcode : uint8
child0 : uint8
child1 : uint8
child2 : uint8
child3 : uint8
value : uint256
BondingCurveExpression
+ constructor(uint256[] expressionTree)
TCRMinDepositExpression
+ constructor(uint256[] expressionTree)

Figure 1: Expression

4. Developer Guide.

All listed sources act as a specification. If we discovered inconsistencies within the actual code behavior, we consulted with the Band Protocol team for further discussion and confirmation.

Components Overview

Supporting Library: `Equation` & `Expression`

Library `Equation` defines an expression tree(reference) consisting of operator opcodes and operand values from an input prefix-ordered array. Each opcode/operator operates on different number of operands. A single unknown variable is supported by the equation tree which can be evaluated upon an input variable value. The library performs math and boolean operations with support of the `SafeMath` and `BancorPower` libraries(reference).

Interface `Expression` exposes method `evaluate` on top of `Equation` for actual usage. The `Equation` and supported contracts power the calculation of flexible bonding curve expressions or deposit expressions.

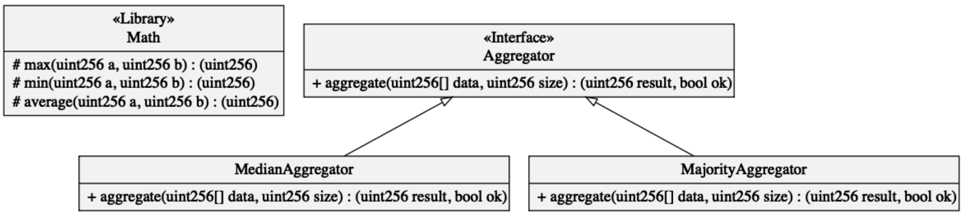

«Library»
Math
# max(uint256 a, uint256 b) : (uint256)
# min(uint256 a, uint256 b) : (uint256)
# average(uint256 a, uint256 b) : (uint256)
«Interface»
Aggregator
+ aggregate(uint256[] data, uint256 size) : (uint256 result, bool ok)
MedianAggregator
+ aggregate(uint256[] data, uint256 size) : (uint256 result, bool ok)
MajorityAggregator
+ aggregate(uint256[] data, uint256 size) : (uint256 result, bool ok)

Figure 2: Aggregator

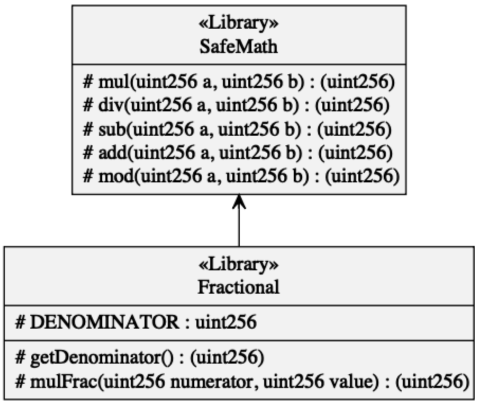

«Library»
SafeMath
# mul(uint256 a, uint256 b) : (uint256)
# div(uint256 a, uint256 b) : (uint256)
# sub(uint256 a, uint256 b) : (uint256)
# add(uint256 a, uint256 b) : (uint256)
# mod(uint256 a, uint256 b) : (uint256)
«Library»
Fractional
# DENOMINATOR : uint256
# getDenominator() : (uint256)
# mulFrac(uint256 numerator, uint256 value) : (uint256)

Figure 3: Fractional

Supporting Contract: `Aggregator`

`Aggregator` provides an interface to aggregate an array of unsigned integer into single result. The `MedianAggregator` contract implements the aggregation by selecting/combining the unweighted median value(s), and the `MajorityAggregator` contract implements the aggregation by picking the value that prevails(over 50%) the entire array, or returning 0 if no such "majority" value exists.

Supporting Math Library: `Fractional`

The `Fractional` library powers fraction multiplication functionality by exposing a `mulFrac` function. The function assumes the first argument to be the numerator x of a fraction $\frac{x}{10^{18}}$, and the second argument to be a normal number.

Supporting Token Contracts: `ERC20Base`, `SnapshotToken` & `LockableToken`

The `ERC20Base` token is a standard `ERC20` token with an `MinterRole` for `mint/burn` permission control.

On top of `ERC20`, the `EIP677 transferAndCall` is implemented in support of safe transfer to contract.

The `SnapshotToken` contract is an `ERC20Base` token with historical balances logged for each account address. The internal `_transfer/_mint/_burn` operations are overriden to trigger logging of the balances of the participants at the moment. The "snapshot" of the global state (in terms of `totalSupply`) and the user state (in terms of user balance) are packed to single `uint256` field for gas saving(reference).

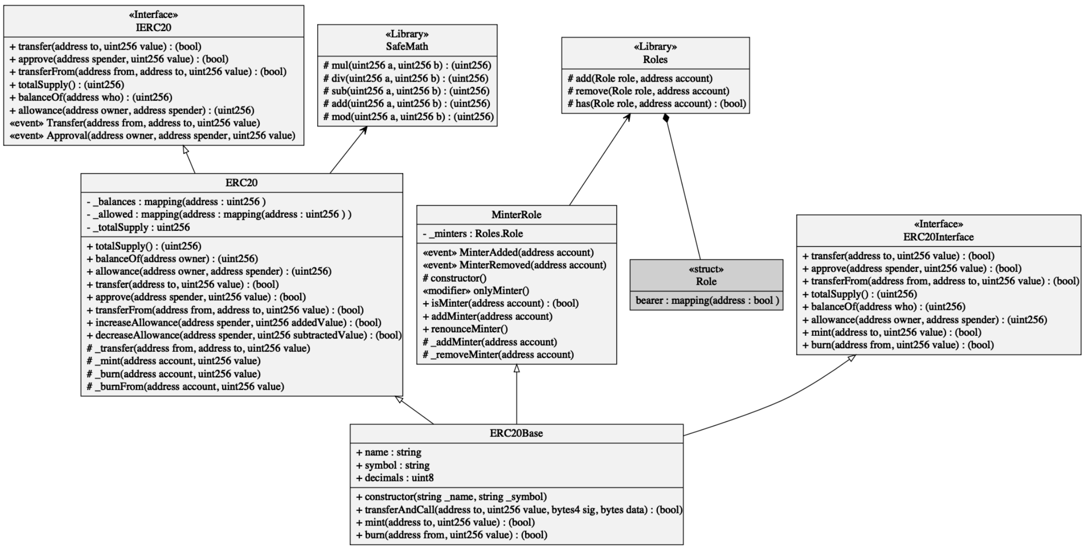

«Interface»
IERC20
+ transfer(address to, uint256 value) : (bool)
+ approve(address spender, uint256 value) : (bool)
+ transferFrom(address from, address to, uint256 value) : (bool)
+ totalSupply() : (uint256)
+ balanceOf(address who) : (uint256)
+ allowance(address owner, address spender) : (uint256)
«event» Transfer(address from, address to, uint256 value)
«event» Approval(address owner, address spender, uint256 value)
«Library»
SafeMath
# mul(uint256 a, uint256 b) : (uint256)
# div(uint256 a, uint256 b) : (uint256)
# sub(uint256 a, uint256 b) : (uint256)
# add(uint256 a, uint256 b) : (uint256)
# mod(uint256 a, uint256 b) : (uint256)
«Library»
Roles
# add(Role role, address account)
# remove(Role role, address account)
# has(Role role, address account) : (bool)
ERC20
- _balances : mapping(address : uint256)
- _allowed : mapping(address : mapping(address : uint256))
- _totalSupply : uint256
+ totalSupply() : (uint256)
+ balanceOf(address owner) : (uint256)
+ allowance(address owner, address spender) : (uint256)
+ transfer(address to, uint256 value) : (bool)
+ approve(address spender, uint256 value) : (bool)
+ transferFrom(address from, address to, uint256 value) : (bool)
+ increaseAllowance(address spender, uint256 addedValue) : (bool)
+ decreaseAllowance(address spender, uint256 subtractedValue) : (bool)
# _transfer(address from, address to, uint256 value)
# _mint(address account, uint256 value)
# _burn(address account, uint256 value)
# _burnFrom(address account, uint256 value)
MinterRole
- _minters : Roles.Role
«event» MinterAdded(address account)
«event» MinterRemoved(address account)
# constructor()
«modifier» onlyMinter()
+ isMinter(address account) : (bool)
+ addMinter(address account)
+ renounceMinter()
# _addMinter(address account)
# _removeMinter(address account)
«struct»
Role
bearer : mapping(address : bool)
«Interface»
ERC20Interface
+ transfer(address to, uint256 value) : (bool)
+ approve(address spender, uint256 value) : (bool)
+ transferFrom(address from, address to, uint256 value) : (bool)
+ totalSupply() : (uint256)
+ balanceOf(address who) : (uint256)
+ allowance(address owner, address spender) : (uint256)
+ mint(address to, uint256 value) : (bool)
+ burn(address from, uint256 value) : (bool)
ERC20Base
+ name : string
+ symbol : string
+ decimals : uint8
+ constructor(string _name, string _symbol)
+ transferAndCall(address to, uint256 value, bytes4 sig, bytes data) : (bool)
+ mint(address to, uint256 value) : (bool)
+ burn(address from, uint256 value) : (bool)

Figure 4: ERC20Base

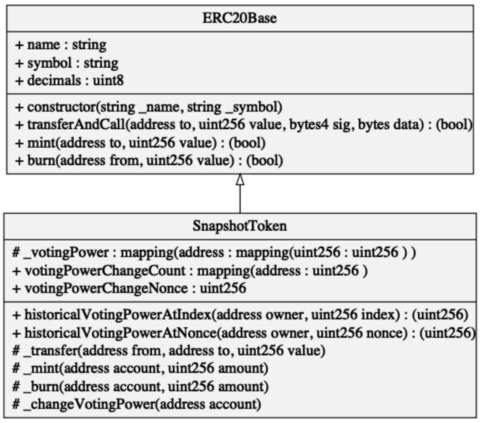

ERC20Base
+ name : string
+ symbol : string
+ decimals : uint8
+ constructor(string _name, string _symbol)
+ transferAndCall(address to, uint256 value, bytes4 sig, bytes data) : (bool)
+ mint(address to, uint256 value) : (bool)
+ burn(address from, uint256 value) : (bool)
SnapshotToken
# _votingPower : mapping(address : mapping(uint256 : uint256))
+ votingPowerChangeCount : mapping(address : uint256)
+ votingPowerChangeNonce : uint256
+ historicalVotingPowerAtIndex(address owner, uint256 index) : (uint256)
+ historicalVotingPowerAtNonce(address owner, uint256 nonce) : (uint256)
# _transfer(address from, address to, uint256 value)
# _mint(address account, uint256 amount)
# _burn(address account, uint256 amount)
# _changeVotingPower(address account)

Figure 5: SnapshotToken

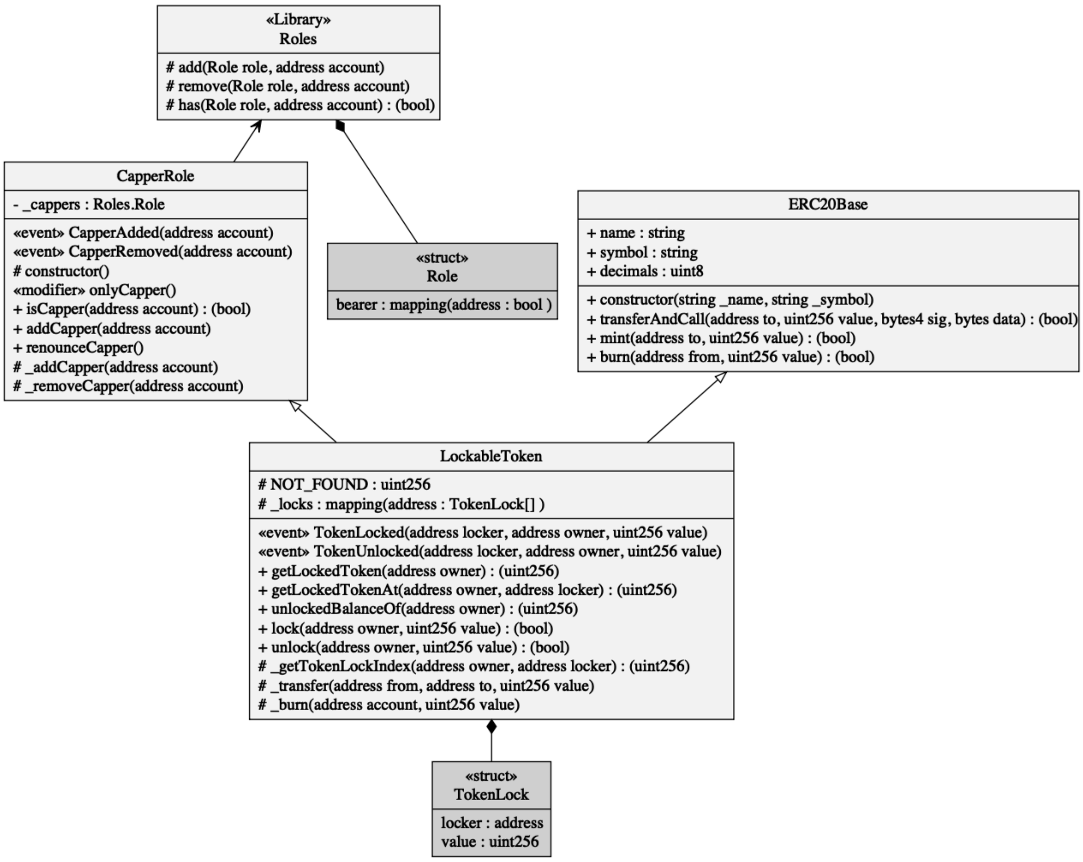

«Library»
Roles
# add(Role role, address account)
# remove(Role role, address account)
# has(Role role, address account) : (bool)
CapperRole
- _cappers : Roles.Role
«event» CapperAdded(address account)
«event» CapperRemoved(address account)
# constructor()
«modifier» onlyCapper()
+ isCapper(address account) : (bool)
+ addCapper(address account)
+ renounceCapper()
# _addCapper(address account)
# _removeCapper(address account)
«struct»
Role
bearer : mapping(address : bool)
ERC20Base
+ name : string
+ symbol : string
+ decimals : uint8
+ constructor(string _name, string _symbol)
+ transferAndCall(address to, uint256 value, bytes4 sig, bytes data) : (bool)
+ mint(address to, uint256 value) : (bool)
+ burn(address from, uint256 value) : (bool)
LockableToken
# NOT_FOUND : uint256
# _locks : mapping(address : TokenLock[])
«event» TokenLocked(address locker, address owner, uint256 value)
«event» TokenUnlocked(address locker, address owner, uint256 value)
+ getLockedToken(address owner) : (uint256)
+ getLockedTokenAt(address owner, address locker) : (uint256)
+ unlockedBalanceOf(address owner) : (uint256)
+ lock(address owner, uint256 value) : (bool)
+ unlock(address owner, uint256 value) : (bool)
# _getTokenLockIndex(address owner, address locker) : (uint256)
# _transfer(address from, address to, uint256 value)
# _burn(address account, uint256 value)
«struct»
TokenLock
locker : address
value : uint256

Figure 6: LockableToken

The `LockableToken` contract is an `ERC20Base` token with `Capper` role added. Each capper can add or remove a lock to a given account. The lock with the maximal value for an account is treated as the main lock for the account and the available balance of the account is calculated by deducting the maximal lock value from the current balance.

Supporting Contract: `Parameters`

The `Parameters` contract provides dynamic storage for a parameter dictionary in which the parameters are used to control the behavior of the band system.

Each `Parameters` set is bound to a `SnapshotToken` whose `votingPowerChangeNonce` is used to identify the voting power of each address at the time of proposal creation.

A voting mechanism is built in to support change of parameter values. The `struct Proposal` and related functions such as `propose`, `vote`, `resolve` are used for the processing of a proposal.

The proposing and resolving process:

1. User create a `Proposal` for the change of one or multiple parameters through `propose`.

2. User can vote for a `Proposal` through `vote` provided that the proposal hasn't expired and that the user has not voted. Its historical balance at/before the creation moment of the proposal is counted as its voting power towards to proposal. User can vote "yes"/"no" for the proposal. Then an early resolution is attempted:

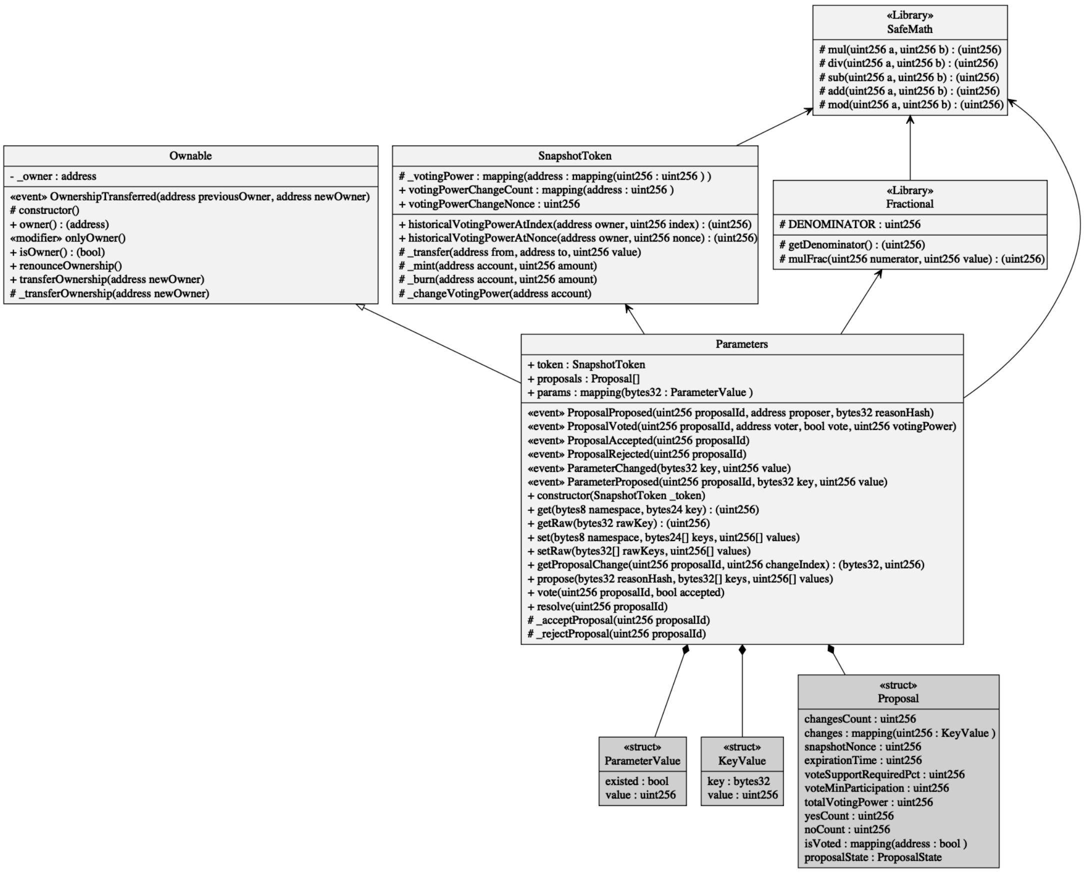

«Library»
SafeMath
# mul(uint256 a, uint256 b) : (uint256)
# div(uint256 a, uint256 b) : (uint256)
# sub(uint256 a, uint256 b) : (uint256)
# add(uint256 a, uint256 b) : (uint256)
# mod(uint256 a, uint256 b) : (uint256)
Ownable
- _owner : address
«event» OwnershipTransferred(address previousOwner, address newOwner)
# constructor()
+ owner() : (address)
«modifier» onlyOwner()
+ isOwner() : (bool)
+ renounceOwnership()
+ transferOwnership(address newOwner)
# _transferOwnership(address newOwner)
SnapshotToken
# _votingPower : mapping(address : mapping(uint256 : uint256))
+ votingPowerChangeCount : mapping(address : uint256)
+ votingPowerChangeNonce : uint256
+ historicalVotingPowerAtIndex(address owner, uint256 index) : (uint256)
+ historicalVotingPowerAtNonce(address owner, uint256 nonce) : (uint256)
# _transfer(address from, address to, uint256 value)
# _mint(address account, uint256 amount)
# _burn(address account, uint256 amount)
# _changeVotingPower(address account)
«Library»
Fractional
# DENOMINATOR : uint256
# getDenominator() : (uint256)
# mulFrac(uint256 numerator, uint256 value) : (uint256)
Parameters
+ token : SnapshotToken
+ proposals : Proposal[]
+ params : mapping(bytes32 : ParameterValue)
«event» ProposalProposed(uint256 proposalId, address proposer, bytes32 reasonHash)
«event» ProposalVoted(uint256 proposalId, address voter, bool vote, uint256 votingPower)
«event» ProposalAccepted(uint256 proposalId)
«event» ProposalRejected(uint256 proposalId)
«event» ParameterChanged(bytes32 key, uint256 value)
«event» ParameterProposed(uint256 proposalId, bytes32 key, uint256 value)
+ constructor(SnapshotToken _token)
+ get(bytes8 namespace, bytes24 key) : (uint256)
+ getRaw(bytes32 rawKey) : (uint256)
+ set(bytes8 namespace, bytes24[] keys, uint256[] values)
+ setRaw(bytes32[] rawKeys, uint256[] values)
+ getProposalChange(uint256 proposalId, uint256 changeIndex) : (bytes32, uint256)
+ propose(bytes32 reasonHash, bytes32[] keys, uint256[] values)
+ vote(uint256 proposalId, bool accepted)
+ resolve(uint256 proposalId)
# _acceptProposal(uint256 proposalId)
# _rejectProposal(uint256 proposalId)
«struct»
ParameterValue
existed : bool
value : uint256
«struct»
KeyValue
key : bytes32
value : uint256
«struct»
Proposal
changesCount : uint256
changes : mapping(uint256 : KeyValue)
snapshotNonce : uint256
expirationTime : uint256
voteSupportRequiredPct : uint256
voteMinParticipation : uint256
totalVotingPower : uint256
yesCount : uint256
noCount : uint256
isVoted : mapping(address : bool)
proposalState : ProposalState

Figure 7: Parameters

- If the accumulated voting power voting for the proposal exceeds $(\frac{\text{voteSupportRequiredPct}}{10^{18}})$ of the total supply at the creation time of the proposal, the proposal is accepted and changes are applied.
- Else if the accumulated voting power voting against the proposal exceeds $(1 - \frac{\text{voteSupportRequiredPct}}{10^{18}})$ of the total supply at the creation time of the proposal, then the proposal is rejected and parameter is changed.
- The `voteSupportRequiredPct` is a parameter in the `Parameters` set as well.

3. If the proposal isn't resolved before it has expired, then after the expiration date user can call `resolve` on the proposal.

    - If the total voted power exceeds the `voteMinParticipation`, and the percentage of the voting power voting for the proposal exceeds $(\frac{\text{voteSupportRequiredPct}}{10^{18}})$ of the total voted power, then the proposal is accepted and changes are aplied.
    - Otherwise the proposal is rejected.

The `owner` of the `Parameters` set is also capable of setting parameters through the `set` and `setRaw` functions directly.

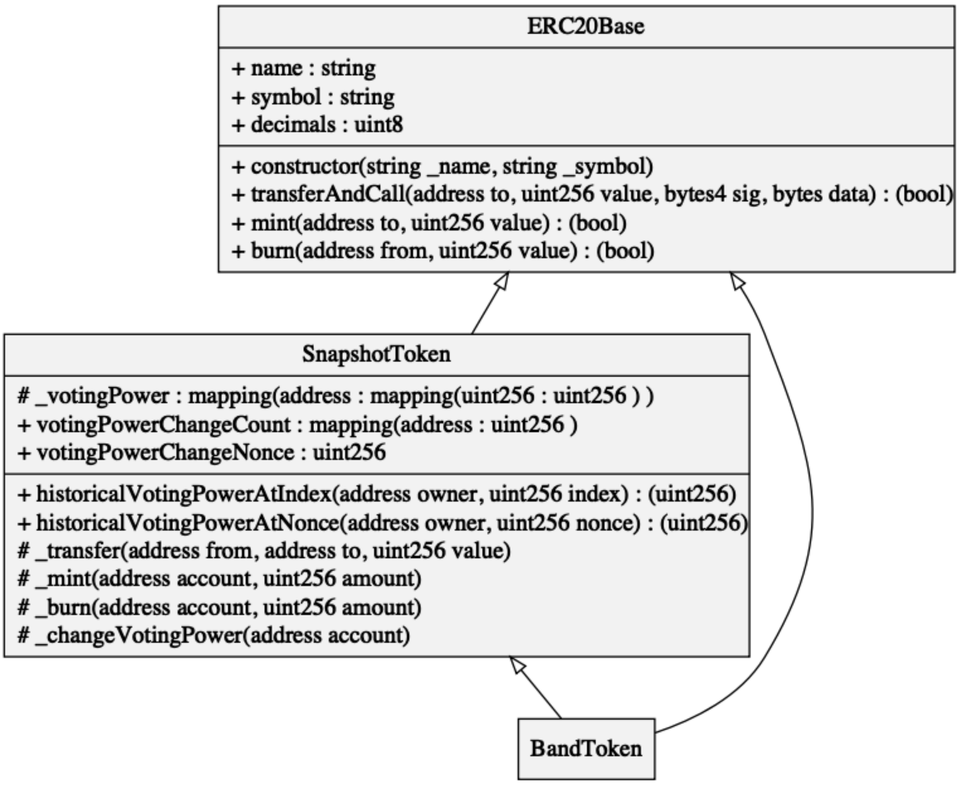

ERC20Base
+ name : string
+ symbol : string
+ decimals : uint8
+ constructor(string _name, string _symbol)
+ transferAndCall(address to, uint256 value, bytes4 sig, bytes data) : (bool)
+ mint(address to, uint256 value) : (bool)
+ burn(address from, uint256 value) : (bool)
SnapshotToken
# _votingPower : mapping(address : mapping(uint256 : uint256))
+ votingPowerChangeCount : mapping(address : uint256)
+ votingPowerChangeNonce : uint256
+ historicalVotingPowerAtIndex(address owner, uint256 index) : (uint256)
+ historicalVotingPowerAtNonce(address owner, uint256 nonce) : (uint256)
# _transfer(address from, address to, uint256 value)
# _mint(address account, uint256 amount)
# _burn(address account, uint256 amount)
# _changeVotingPower(address account)
BandToken

Figure 8: BandToken

Main Contract: `BandToken`, `CommunityToken`

BandToken describes the native ERC-20 token of Band Protocol called `BAND`, which is the universal platform token interchangeable to different `CommunityToken` in each band community. It is a `SnapshotToken` which logs the historical balance of each account upon balance update.

The inheritance from the `SnapshotToken` indicates the presence of the `MinterRole` for `mint`/`burn` control of the `BandToken`.

`CommunityToken` is a ERC-20 token for each band community. It is interchangeable to the universal `BandToken`.

The inheritance from the `SnapshotToken` and `LockableToken` indicates the presence of the `MinterRole` for `mint`/`burn` control and the `CapperRole` for `lock`/`unlock` control of the `CommunityToken`.

Supporting Contract: `ERC20Acceptor`

The `ERC20Acceptor` extends an ERC-20 token by adding a modifier `requireToken`, which ensures an `amount` of tokens to be transferred to the current ERC-20 contract from `token` before further operations.

Main Contract: `BondingCurve`

The `BondingCurve` contract serves as the exchange for two tokens: one `collateralToken` (`ERC20Interface`) and one `bondedToken`(`ERC20Interface`). The `collateralToken` is mainly the

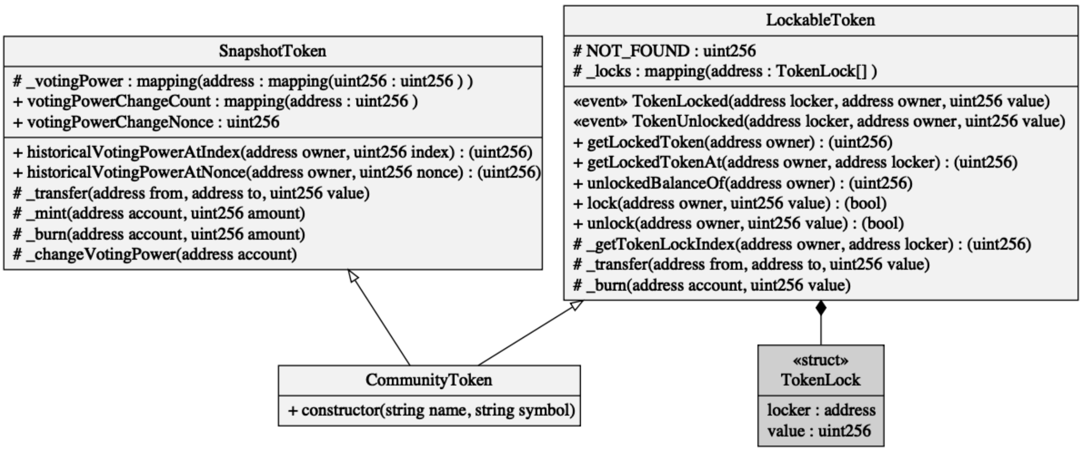

SnapshotToken
# _votingPower : mapping(address : mapping(uint256 : uint256))
+ votingPowerChangeCount : mapping(address : uint256)
+ votingPowerChangeNonce : uint256
+ historicalVotingPowerAtIndex(address owner, uint256 index) : (uint256)
+ historicalVotingPowerAtNonce(address owner, uint256 nonce) : (uint256)
# _transfer(address from, address to, uint256 value)
# _mint(address account, uint256 amount)
# _burn(address account, uint256 amount)
# _changeVotingPower(address account)
LockableToken
# NOT_FOUND : uint256
# _locks : mapping(address : TokenLock[])
«event» TokenLocked(address locker, address owner, uint256 value)
«event» TokenUnlocked(address locker, address owner, uint256 value)
+ getLockedToken(address owner) : (uint256)
+ getLockedTokenAt(address owner, address locker) : (uint256)
+ unlockedBalanceOf(address owner) : (uint256)
+ lock(address owner, uint256 value) : (bool)
+ unlock(address owner, uint256 value) : (bool)
# _getTokenLockIndex(address owner, address locker) : (uint256)
# _transfer(address from, address to, uint256 value)
# _burn(address account, uint256 value)
CommunityToken
+ constructor(string name, string symbol)
«struct»
TokenLock
locker : address
value : uint256

Figure 9: CommunityToken

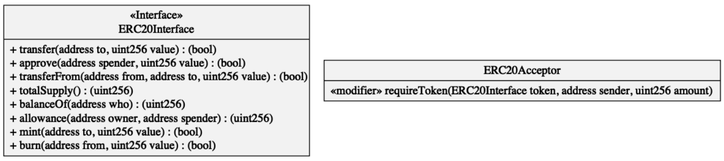

«Interface»
ERC20Interface
+ transfer(address to, uint256 value) : (bool)
+ approve(address spender, uint256 value) : (bool)
+ transferFrom(address from, address to, uint256 value) : (bool)
+ totalSupply() : (uint256)
+ balanceOf(address who) : (uint256)
+ allowance(address owner, address spender) : (uint256)
+ mint(address to, uint256 value) : (bool)
+ burn(address from, uint256 value) : (bool)
ERC20Acceptor
«modifier» requireToken(ERC20Interface token, address sender, uint256 amount)

Figure 10: ERC20Acceptor

`BandToken` and the `bondedToken` is usually `CommunityToken`.

A `BondingCurve` exchange is usually needed for each band community to exchange between the universal `BAND` initially released as an ERC-20 token on Ethereum, and the community-specific `CommunityToken`. `BAND` is used as collateral to issue community tokens (i.e. community tokens are bonded by `BAND`).

Anyone can buy community token by sending BAND to the data governance group's bonding curve smart contract. Conversely, community token can be sold to the bonding curve to receive back BAND.

An input bonding curve in the form of `Equation` tree (stored in the `Paramters` set of the `BondingCurve` exchange) is used to calculate the buying and selling price of the `bondedToken` based on its supply. A liquidity spread (also stored in the `Parameters` set of the `BondingCurve` exchange) is imposed upon the calculated price to better maintain the total supply of the `CommunityToken`(reference).

The `Parameters` set of the `BondingCurve` is usually initialized with the bonded `CommunityToken` of the `BondingCurve` as its based `SnapshotToken`, which suggests that the change of parameters of a community is determined by the `CommunityToken` holders within the community.

Main Contract: `TCRBase` & Derivatives

`TCRBase` (Token-Curated-Registry, reference) defines an on-chain entry list data structure. Application candidate stake dataset/community tokens (larger than `min_deposit`, a parameter stored in the `Paramters` set of the `TCR`) in order to enlist an entry in the TCR.

The dataset/community token is the same as the `SnapshotToken` of the `Parameters` set in the TCR.

The current minimal deposit requirement of an entry conforms to a depreciative stake model(reference), calculated dynamically with regard to the current time through a piecewise decreasing `Expression` stored in the `deposit_decay_function` parameter of the `TCR`.

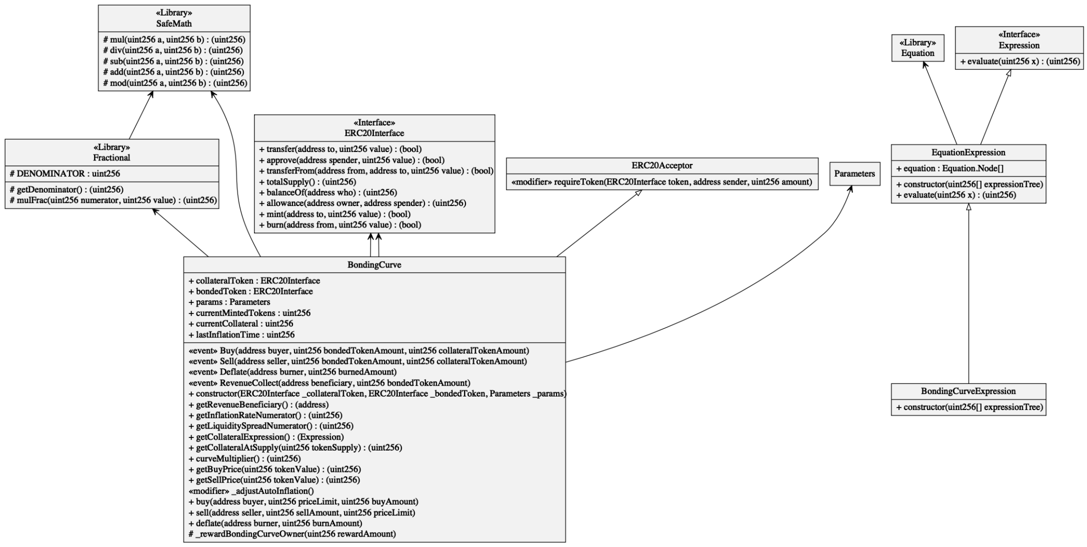

«Library»
SafeMath
# mul(uint256 a, uint256 b) : (uint256)
# div(uint256 a, uint256 b) : (uint256)
# sub(uint256 a, uint256 b) : (uint256)
# add(uint256 a, uint256 b) : (uint256)
# mod(uint256 a, uint256 b) : (uint256)
«Library»
Fractional
# DENOMINATOR : uint256
# getDenominator() : (uint256)
# mulFrac(uint256 numerator, uint256 value) : (uint256)
«Interface»
ERC20Interface
+ transfer(address to, uint256 value) : (bool)
+ approve(address spender, uint256 value) : (bool)
+ transferFrom(address from, address to, uint256 value) : (bool)
+ totalSupply() : (uint256)
+ balanceOf(address who) : (uint256)
+ allowance(address owner, address spender) : (uint256)
+ mint(address to, uint256 value) : (bool)
+ burn(address from, uint256 value) : (bool)
ERC20Acceptor
«modifier» requireToken(ERC20Interface token, address sender, uint256 amount)
Parameters
«Library»
Equation
«Interface»
Expression
+ evaluate(uint256 x) : (uint256)
EquationExpression
+ equation : Equation.Node[]
+ constructor(uint256[] expressionTree)
+ evaluate(uint256 x) : (uint256)
BondingCurve
+ collateralToken : ERC20Interface
+ bondedToken : ERC20Interface
+ params : Parameters
+ currentMintedTokens : uint256
+ currentCollateral : uint256
+ lastInflationTime : uint256
«event» Buy(address buyer, uint256 bondedTokenAmount, uint256 collateralTokenAmount)
«event» Sell(address seller, uint256 bondedTokenAmount, uint256 collateralTokenAmount)
«event» Deflate(address burner, uint256 burnedAmount)
«event» RevenueCollect(address beneficiary, uint256 bondedTokenAmount)
+ constructor(ERC20Interface _collateralToken, ERC20Interface _bondedToken, Parameters _params)
+ getRevenueBeneficiary() : (address)
+ getInflationRateNumerator() : (uint256)
+ getLiquiditySpreadNumerator() : (uint256)
+ getCollateralExpression() : (Expression)
+ getCollateralAtSupply(uint256 tokenSupply) : (uint256)
+ curveMultiplier() : (uint256)
+ getBuyPrice(uint256 tokenValue) : (uint256)
+ getSellPrice(uint256 tokenValue) : (uint256)
«modifier» _adjustAutoInflation()
+ buy(address buyer, uint256 priceLimit, uint256 buyAmount)
+ sell(address seller, uint256 sellAmount, uint256 priceLimit)
+ deflate(address burner, uint256 burnAmount)
# _rewardBondingCurveOwner(uint256 rewardAmount)
BondingCurveExpression
+ constructor(uint256[] expressionTree)

Figure 11: BondingCurve

Token holders are capable of challenging an existing entry, as well as voting for/against a challenged entry. The challenge depends on the snapshot voting power stored in its `SnapshotToken` token.

The challenging and resolving process:

1. User challenges to kick out an entry by staking an amount larger than the current minimal deposit requirement of the entry. The challenge stake goes to the `rewardPool` of the challenge. The staking of the challenged entry is temporarily deducted.

2. Token holder can vote for keeping an entry before the challenge has expired. Its historical balance at the time of the creation of the challenge is used as the its voting power. The voted value of each voter is encrypted using a salt value to prevent early exposure.

3. When the challenge voting time has expired, there is another time period for voter to reveal their vote by providing the voted value and the salt for verification. The voted value (`keep` or `remove`) is updated in the challenge entry.

4. Anyone may try to resolve a challenge after the vote committing time period has expired.

    - If the total commited voting power is 0 or less than the `min_participation_pct` parameter, the challenge is defined as `Inconlusive`. The deducted deposit of the entry as well as the staking for the challenge will be sent back to the original owner.

    - Else if the commited voting power for removing the entry has exceeded the `voteRemoveRequiredPct` parameter of the total commited voting power, the challenge succeeds and the entry is to be removed. The original challenge stake is sent back to the challenger, together with `dispensation_percentage` and the challenger's voting power percentage (among all the voting power voting `removed`) of the entry's deposit. The reset is to be split by other voters voting `removed` according to their historical voting power.

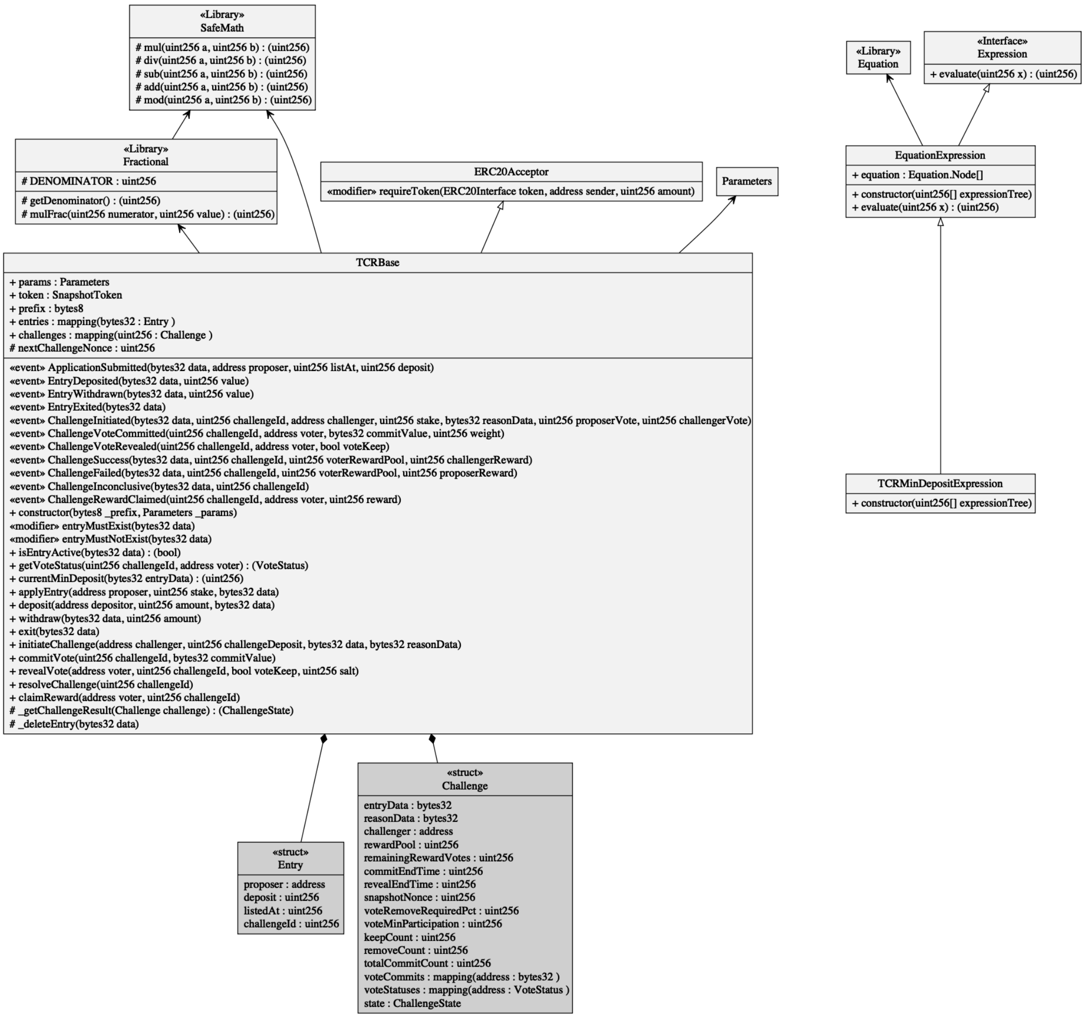

«Library»
SafeMath
# mul(uint256 a, uint256 b) : (uint256)
# div(uint256 a, uint256 b) : (uint256)
# sub(uint256 a, uint256 b) : (uint256)
# add(uint256 a, uint256 b) : (uint256)
# mod(uint256 a, uint256 b) : (uint256)
«Library»
Fractional
# DENOMINATOR : uint256
# getDenominator() : (uint256)
# mulFrac(uint256 numerator, uint256 value) : (uint256)
ERC20Acceptor
«modifier» requireToken(ERC20Interface token, address sender, uint256 amount)
Parameters
TCRBase
+ params : Parameters
+ token : SnapshotToken
+ prefix : bytes8
+ entries : mapping(bytes32 : Entry)
+ challenges : mapping(uint256 : Challenge)
# nextChallengeNonce : uint256
«event» ApplicationSubmitted(bytes32 data, address proposer, uint256 listAt, uint256 deposit)
«event» EntryDeposited(bytes32 data, uint256 value)
«event» EntryWithdrawn(bytes32 data, uint256 value)
«event» EntryExited(bytes32 data)
«event» ChallengeInitiated(bytes32 data, uint256 challengeId, address challenger, uint256 stake, bytes32 reasonData, uint256 proposerVote, uint256 challengerVote)
«event» ChallengeVoteCommitted(uint256 challengeId, address voter, bytes32 commitValue, uint256 weight)
«event» ChallengeVoteRevealed(uint256 challengeId, address voter, bool voteKeep)
«event» ChallengeSuccess(bytes32 data, uint256 challengeId, uint256 voterRewardPool, uint256 challengerReward)
«event» ChallengeFailed(bytes32 data, uint256 challengeId, uint256 voterRewardPool, uint256 proposerReward)
«event» ChallengeInconclusive(bytes32 data, uint256 challengeId)
«event» ChallengeRewardClaimed(uint256 challengeId, address voter, uint256 reward)
+ constructor(bytes8 _prefix, Parameters _params)
«modifier» entryMustExist(bytes32 data)
«modifier» entryMustNotExist(bytes32 data)
+ isEntryActive(bytes32 data) : (bool)
+ getVoteStatus(uint256 challengeId, address voter) : (VoteStatus)
+ currentMinDeposit(bytes32 entryData) : (uint256)
+ applyEntry(address proposer, uint256 stake, bytes32 data)
+ deposit(address depositor, uint256 amount, bytes32 data)
+ withdraw(bytes32 data, uint256 amount)
+ exit(bytes32 data)
+ initiateChallenge(address challenger, uint256 challengeDeposit, bytes32 data, bytes32 reasonData)
+ commitVote(uint256 challengeId, bytes32 commitValue)
+ revealVote(address voter, uint256 challengeId, bool voteKeep, uint256 salt)
+ resolveChallenge(uint256 challengeId)
+ claimReward(address voter, uint256 challengeId)
# _getChallengeResult(Challenge challenge) : (ChallengeState)
# _deleteEntry(bytes32 data)
«struct»
Entry
proposer : address
deposit : uint256
listedAt : uint256
challengeId : uint256
«struct»
Challenge
entryData : bytes32
reasonData : bytes32
challenger : address
rewardPool : uint256
remainingRewardVotes : uint256
commitEndTime : uint256
revealEndTime : uint256
snapshotNonce : uint256
voteRemoveRequiredPct : uint256
voteMinParticipation : uint256
keepCount : uint256
removeCount : uint256
totalCommitCount : uint256
voteCommits : mapping(address : bytes32)
voteStatuses : mapping(address : VoteStatus)
state : ChallengeState
«Library»
Equation
«Interface»
Expression
+ evaluate(uint256 x) : (uint256)
EquationExpression
+ equation : Equation.Node[]
+ constructor(uint256[] expressionTree)
+ evaluate(uint256 x) : (uint256)
TCRMinDepositExpression
+ constructor(uint256[] expressionTree)

Figure 12: TCRBase

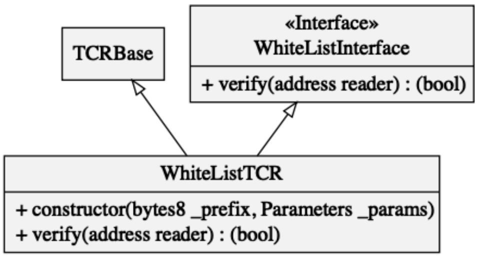

TCRBase
«Interface»
WhiteListInterface
+ verify(address reader) : (bool)
WhiteListTCR
+ constructor(bytes8 _prefix, Parameters _params)
+ verify(address reader) : (bool)

Figure 13: WhitelistTCR

- Otherwise the challenge failed and the entry is to be kept. The entry deposit is sent back to the entry, together with `dispensation_percentage` and the entry proposer's voting power percentage (among all the voting power voting `kept`) of the challenge staking. The reset is to be split by other voters voting `kept` according to their historical voting power.

5. Voter wining a challenge may claim its share of the reward pool according to the percentage of its voting power among all the winning voting power.

Entry owner may choose to delete a self-proposed entry any time that the entry isn't being challenged. The deposit of the entry is transferred back immediately.

Entry owner may deposit more tokens to the entry. Entry owner may also withdraw tokens from the entry. If the entry is not challenged, the remaining amount must exceeds the current minimal deposit requirement determined by the depreciative stake model in order for the withdraw to succeed; else it can withdraw any amount of the remaining deposit of the entry, as the initiation of the challenge has already deducted a required amount from the challenged entry's deposit.

`TCRBase` Derivative: `WhitelistTCR`(`WhiteListInterface`)

`WhitelistTCR` is using TCR as a whitelist for authenticating a given address `reader`. It relies on the design that `TCRBase` requires an amount of community tokens to be staked in the TCR contract in order to be enlisted as an entry.

`TCRBase` Derivative: `QueryTCR` (`QueryInterface`)

The `QueryInterface` specifies three query methods. It uses the whitelist contract contained in the `BandRegistry` (the band system addresses bundler, see below) for implementation of data query.

`QueryTCR` is a `TCR` that supports query of its the TCR entry at zero cost. It may serve as persistent data storage for a band community.

Main Contract: `BandExchangeInterface` & `BandRegistry`

The `BandExchangeInterface` interface specifies the most fundamental functionality of a band exchange as the method name suggested.

The `BandRegistry` serves as the basic infrastructure of the band ecosystem, which stores the three main contracts within the band ecosystem: the BAND token, a decentralized exchange for ETH and BAND, and a whitelist for verifying non-malicious data consumers/community participants (e.g. `WhitelistTCR`).

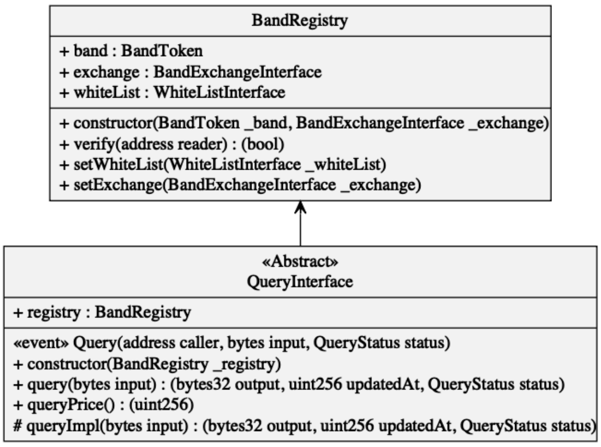

BandRegistry
+ band : BandToken
+ exchange : BandExchangeInterface
+ whiteList : WhiteListInterface
+ constructor(BandToken _band, BandExchangeInterface _exchange)
+ verify(address reader) : (bool)
+ setWhiteList(WhiteListInterface _whiteList)
+ setExchange(BandExchangeInterface _exchange)
«Abstract»
QueryInterface
+ registry : BandRegistry
«event» Query(address caller, bytes input, QueryStatus status)
+ constructor(BandRegistry _registry)
+ query(bytes input) : (bytes32 output, uint256 updatedAt, QueryStatus status)
+ queryPrice() : (uint256)
# queryImpl(bytes input) : (bytes32 output, uint256 updatedAt, QueryStatus status)

Figure 14: QueryInterface

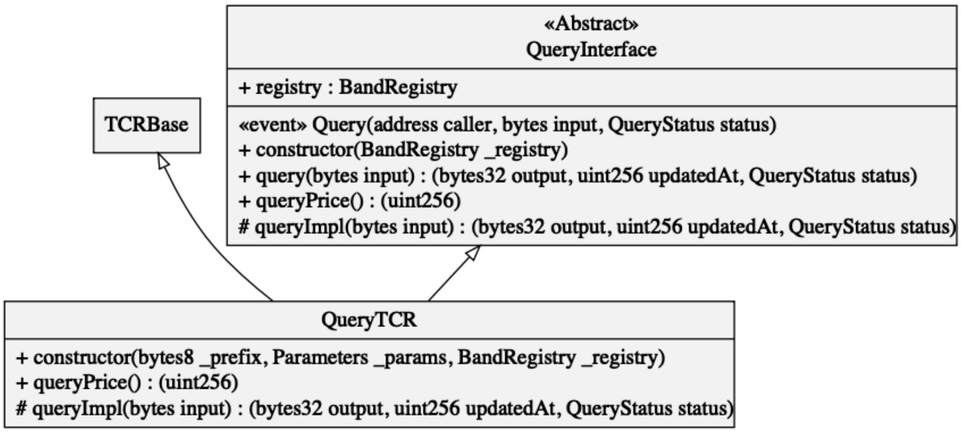

«Abstract»
QueryInterface
+ registry : BandRegistry
«event» Query(address caller, bytes input, QueryStatus status)
+ constructor(BandRegistry _registry)
+ query(bytes input) : (bytes32 output, uint256 updatedAt, QueryStatus status)
+ queryPrice() : (uint256)
# queryImpl(bytes input) : (bytes32 output, uint256 updatedAt, QueryStatus status)
TCRBase
QueryTCR
+ constructor(bytes8 _prefix, Parameters _params, BandRegistry _registry)
+ queryPrice() : (uint256)
# queryImpl(bytes input) : (bytes32 output, uint256 updatedAt, QueryStatus status)

Figure 15: QueryTCR

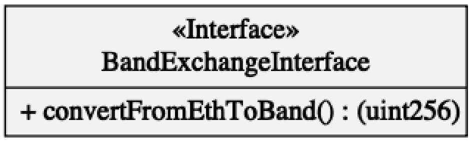


Figure 16: BandExchangeInterface

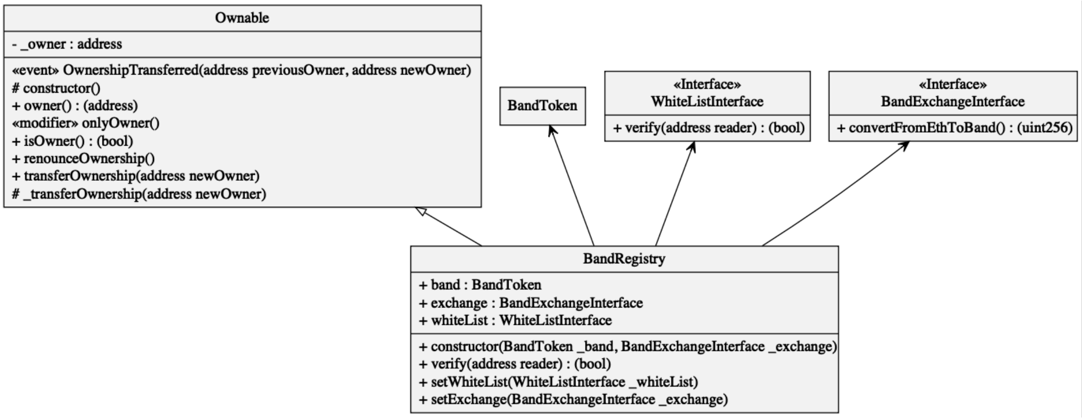

Ownable
- _owner : address
«event» OwnershipTransferred(address previousOwner, address newOwner)
# constructor()
+ owner() : (address)
«modifier» onlyOwner()
+ isOwner() : (bool)
+ renounceOwnership()
+ transferOwnership(address newOwner)
# _transferOwnership(address newOwner)
BandToken
«Interface»
WhiteListInterface
+ verify(address reader) : (bool)
«Interface»
BandExchangeInterface
+ convertFromEthToBand() : (uint256)
BandRegistry
+ band : BandToken
+ exchange : BandExchangeInterface
+ whiteList : WhiteListInterface
+ constructor(BandToken _band, BandExchangeInterface _exchange)
+ verify(address reader) : (bool)
+ setWhiteList(WhiteListInterface _whiteList)
+ setExchange(BandExchangeInterface _exchange)

Figure 17: BandRegistry

Main Contract: TCDBase & Derivatives

TCDBase is an abstract class for Band Protocol's Token-Curated Data Sources. Anyone can register a new data source by staking an amount of tokens larger than the min_provider_stake specified in Parameters set.

A BondingCurve exchange is contained in the TCD to exchange BAND with the community token. A BandRegistry addresses registry is also contained in the TCD, which provides an exchange for BAND and ETH, and an access to the collateral BAND token. The main token for the TCD is the bonded community token specified in the BondingCurve. The BondingCurve exchange is granted nearly unlimited amount of allowances to trade the BAND token with the community token.

Token holders can stake or unstake community tokens for a proposed data source. Two sorted linked list (implemented using the mapping structure) are used to organize the data sources.

- The activeList keeps a sorted list of data sources in increasing staking order. It contains the highest staking data sources which are chosen to provide data for the community. It has a maximum size limit specified by the max_provider_count parameter.

- The reserveList keeps a sorted list of data sources in decreasing staking order. It stores all the non-highest-staked data sources which might potentially become active data source upon changes of staking.

For unstaking of a data source, if the withdrawer is the data source provider, the unstaked amount is saved to a withdrawReceipts which can only be withdrawn after a certain amount of time. Otherwise the share of staking of the withdrawer is unlocked and can be transferred traded instantly. Every change to the staking condition of the data sources such as register, stake, and unstak will trigger reordering of the activeList and reserveList.

The data querying fees are collected in ETH and converted to BAND using the Band ExchangeInterface function convertFromEthToBand. A portion of the fees can be distributed evenly among the active data sources as community tokens after further converting from BAND to the community tokens.

TCDBase Derivative: AggTCD

AggTCD implements the QueryInterface upon the TCDBase. The query price is determined by the query_price parameter. For queryImpl the data are collected from the active data sources count in TCDBase by calling get(bytes) to each data source contract. Only when $\geq \frac{2}{3}$ of the total active data sources have returned values will the result values be proceeded and aggregated (using an Aggregator) and successfully returned to the user. Otherwise a NOT_AVAILABLE result will be returned.

TCDBase Derivative: MultiSigTCD

MultiSigTCD also implements the QueryInterface upon the TCDBase. The results from the data sources are gathered off-chain and reported back to the MultiSigTCD using the report function. The ECDSA signatures of the active data source providers are provided together with the aggregated data for validation of the identity.

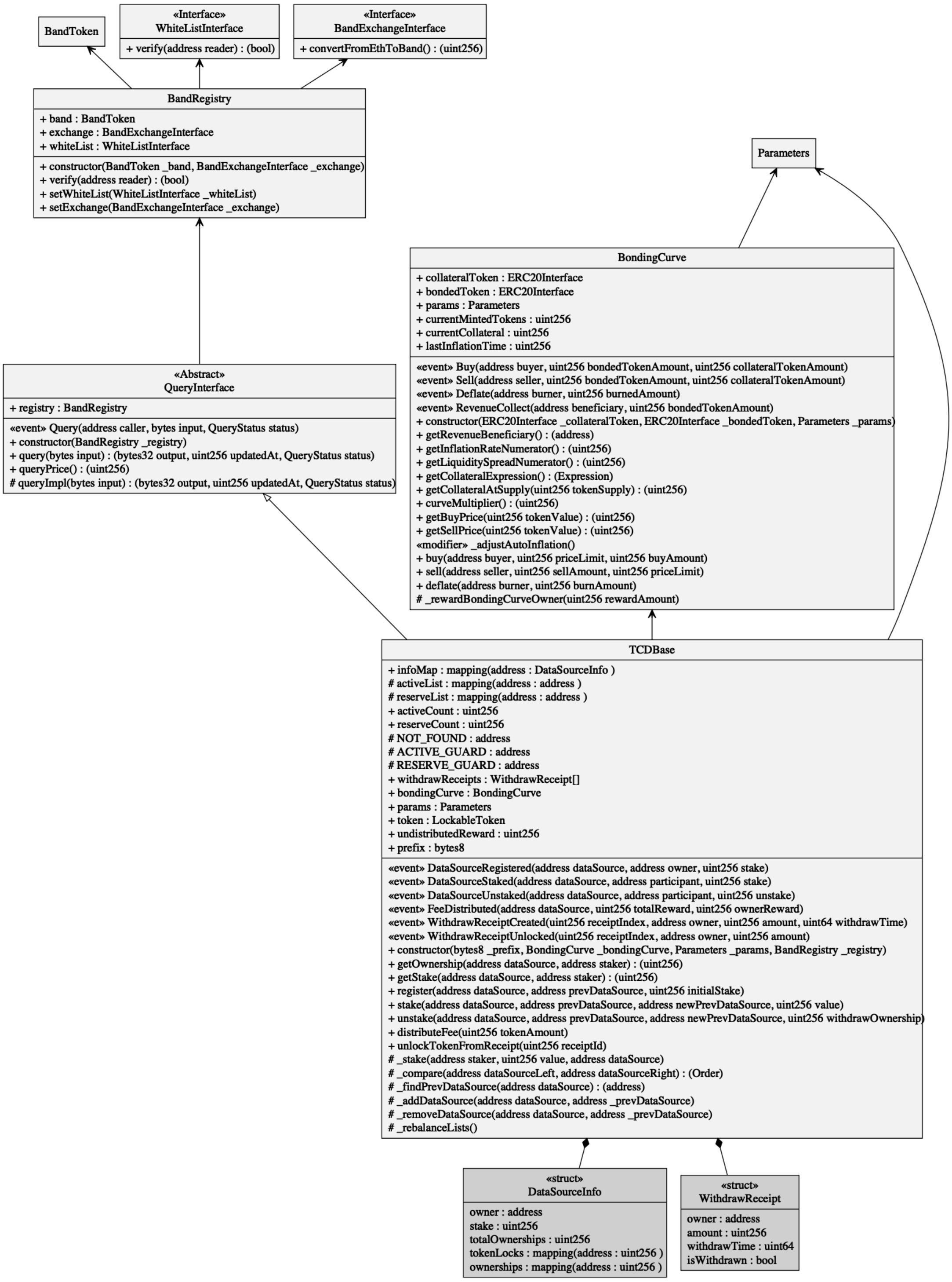

BandToken
«Interface»
WhiteListInterface
+ verify(address reader) : (bool)
«Interface»
BandExchangeInterface
+ convertFromEthToBand() : (uint256)
BandRegistry
+ band : BandToken
+ exchange : BandExchangeInterface
+ whiteList : WhiteListInterface
+ constructor(BandToken _band, BandExchangeInterface _exchange)
+ verify(address reader) : (bool)
+ setWhiteList(WhiteListInterface _whiteList)
+ setExchange(BandExchangeInterface _exchange)
Parameters
BondingCurve
+ collateralToken : ERC20Interface
+ bondedToken : ERC20Interface
+ params : Parameters
+ currentMintedTokens : uint256
+ currentCollateral : uint256
+ lastInflationTime : uint256
«event» Buy(address buyer, uint256 bondedTokenAmount, uint256 collateralTokenAmount)
«event» Sell(address seller, uint256 bondedTokenAmount, uint256 collateralTokenAmount)
«event» Deflate(address burner, uint256 burnedAmount)
«event» RevenueCollect(address beneficiary, uint256 bondedTokenAmount)
+ constructor(ERC20Interface _collateralToken, ERC20Interface _bondedToken, Parameters _params)
+ getRevenueBeneficiary() : (address)
+ getInflationRateNumerator() : (uint256)
+ getLiquiditySpreadNumerator() : (uint256)
+ getCollateralExpression() : (Expression)
+ getCollateralAtSupply(uint256 tokenSupply) : (uint256)
+ curveMultiplier() : (uint256)
+ getBuyPrice(uint256 tokenValue) : (uint256)
+ getSellPrice(uint256 tokenValue) : (uint256)
«modifier» _adjustAutoInflation()
+ buy(address buyer, uint256 priceLimit, uint256 buyAmount)
+ sell(address seller, uint256 sellAmount, uint256 priceLimit)
+ deflate(address burner, uint256 burnAmount)
# _rewardBondingCurveOwner(uint256 rewardAmount)
«Abstract»
QueryInterface
+ registry : BandRegistry
«event» Query(address caller, bytes input, QueryStatus status)
+ constructor(BandRegistry _registry)
+ query(bytes input) : (bytes32 output, uint256 updatedAt, QueryStatus status)
+ queryPrice() : (uint256)
# queryImpl(bytes input) : (bytes32 output, uint256 updatedAt, QueryStatus status)
TCDBase
+ infoMap : mapping(address : DataSourceInfo)
# activeList : mapping(address : address)
# reserveList : mapping(address : address)
+ activeCount : uint256
+ reserveCount : uint256
# NOT_FOUND : address
# ACTIVE_GUARD : address
# RESERVE_GUARD : address
+ withdrawReceipts : WithdrawReceipt[]
+ bondingCurve : BondingCurve
+ params : Parameters
+ token : LockableToken
+ undistributedReward : uint256
+ prefix : bytes8
«event» DataSourceRegistered(address dataSource, address owner, uint256 stake)
«event» DataSourceStaked(address dataSource, address participant, uint256 stake)
«event» DataSourceUnstaked(address dataSource, address participant, uint256 unstake)
«event» FeeDistributed(address dataSource, uint256 totalReward, uint256 ownerReward)
«event» WithdrawReceiptCreated(uint256 receiptIndex, address owner, uint256 amount, uint64 withdrawTime)
«event» WithdrawReceiptUnlocked(uint256 receiptIndex, address owner, uint256 amount)
+ constructor(bytes8 _prefix, BondingCurve _bondingCurve, Parameters _params, BandRegistry _registry)
+ getOwnership(address dataSource, address staker) : (uint256)
+ getStake(address dataSource, address staker) : (uint256)
+ register(address dataSource, address prevDataSource, uint256 initialStake)
+ stake(address dataSource, address prevDataSource, address newPrevDataSource, uint256 value)
+ unstake(address dataSource, address prevDataSource, address newPrevDataSource, uint256 withdrawOwnership)
+ distributeFee(uint256 tokenAmount)
+ unlockTokenFromReceipt(uint256 receiptId)
# _stake(address staker, uint256 value, address dataSource)
# _compare(address dataSourceLeft, address dataSourceRight) : (Order)
# _findPrevDataSource(address dataSource) : (address)
# _addDataSource(address dataSource, address _prevDataSource)
# _removeDataSource(address dataSource, address _prevDataSource)
# _rebalanceLists()
«struct»
DataSourceInfo
owner : address
stake : uint256
totalOwnerships : uint256
tokenLocks : mapping(address : uint256)
ownerships : mapping(address : uint256)
«struct»
WithdrawReceipt
owner : address
amount : uint256
withdrawTime : uint64
isWithdrawn : bool

Figure 18: TCDBase

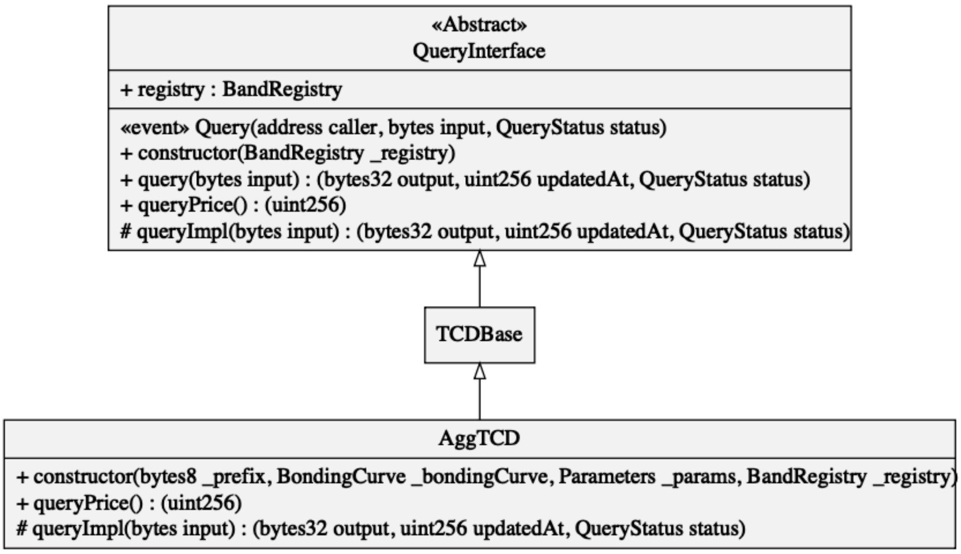

«Abstract»
QueryInterface
+ registry : BandRegistry
«event» Query(address caller, bytes input, QueryStatus status)
+ constructor(BandRegistry _registry)
+ query(bytes input) : (bytes32 output, uint256 updatedAt, QueryStatus status)
+ queryPrice() : (uint256)
# queryImpl(bytes input) : (bytes32 output, uint256 updatedAt, QueryStatus status)
TCDBase
AggTCD
+ constructor(bytes8 _prefix, BondingCurve _bondingCurve, Parameters _params, BandRegistry _registry)
+ queryPrice() : (uint256)
# queryImpl(bytes input) : (bytes32 output, uint256 updatedAt, QueryStatus status)

Figure 19: AggTCD

Only when the reported results $\geq \frac{2}{3}$ of the total active data sources count and that the reported signatures are all valid from the active data source providers will the reported results be aggregated and saved for user query.

`TCDBase` Derivative: `OffchainAggTCD`

`OffchainAggTCD` is similar to the `MultiSigTCD` where data providers provide their signatures for identity verification when reporting data, whereas the difference lies in the data providers are responsible of aggregating the results off-chain before reporting back the contract to be saved for user query.

Main Contract: `VestingWallet`

The `VestingWallet` contract specifies a linear token vesting process with cliff.

Recommendations

Items in this section are low impact on the overall aspects of the smart contracts, which will let the client decide whether to have those reflected in the final deployed version of source codes. The entries are labeled CRITICAL IMPORTANT INFO DISCUSSION (in a decreasing significance level manner).

TCRBase.sol

- INFO Recommend deactivating the entry when it is being challenged in `isEntryActive()`.
- INFO Recommend saving the deducted entry deposit in `initiateChallenge()` to provide written proof of consistent total supply.

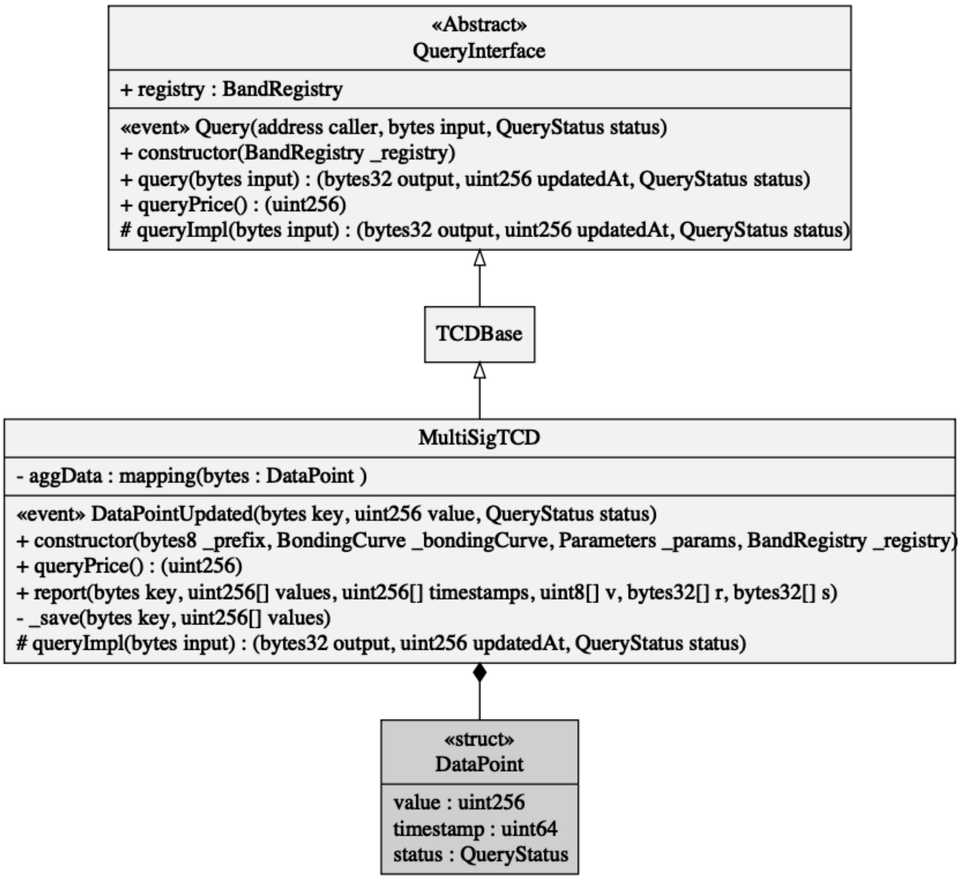

«Abstract»
QueryInterface
+ registry : BandRegistry
«event» Query(address caller, bytes input, QueryStatus status)
+ constructor(BandRegistry _registry)
+ query(bytes input) : (bytes32 output, uint256 updatedAt, QueryStatus status)
+ queryPrice() : (uint256)
# queryImpl(bytes input) : (bytes32 output, uint256 updatedAt, QueryStatus status)
TCDBase
MultiSigTCD
- aggData : mapping(bytes : DataPoint)
«event» DataPointUpdated(bytes key, uint256 value, QueryStatus status)
+ constructor(bytes8 _prefix, BondingCurve _bondingCurve, Parameters _params, BandRegistry _registry)
+ queryPrice() : (uint256)
+ report(bytes key, uint256[] values, uint256[] timestamps, uint8[] v, bytes32[] r, bytes32[] s)
- _save(bytes key, uint256[] values)
# queryImpl(bytes input) : (bytes32 output, uint256 updatedAt, QueryStatus status)
«struct»
DataPoint
value : uint256
timestamp : uint64
status : QueryStatus

Figure 20: MultiSigTCD

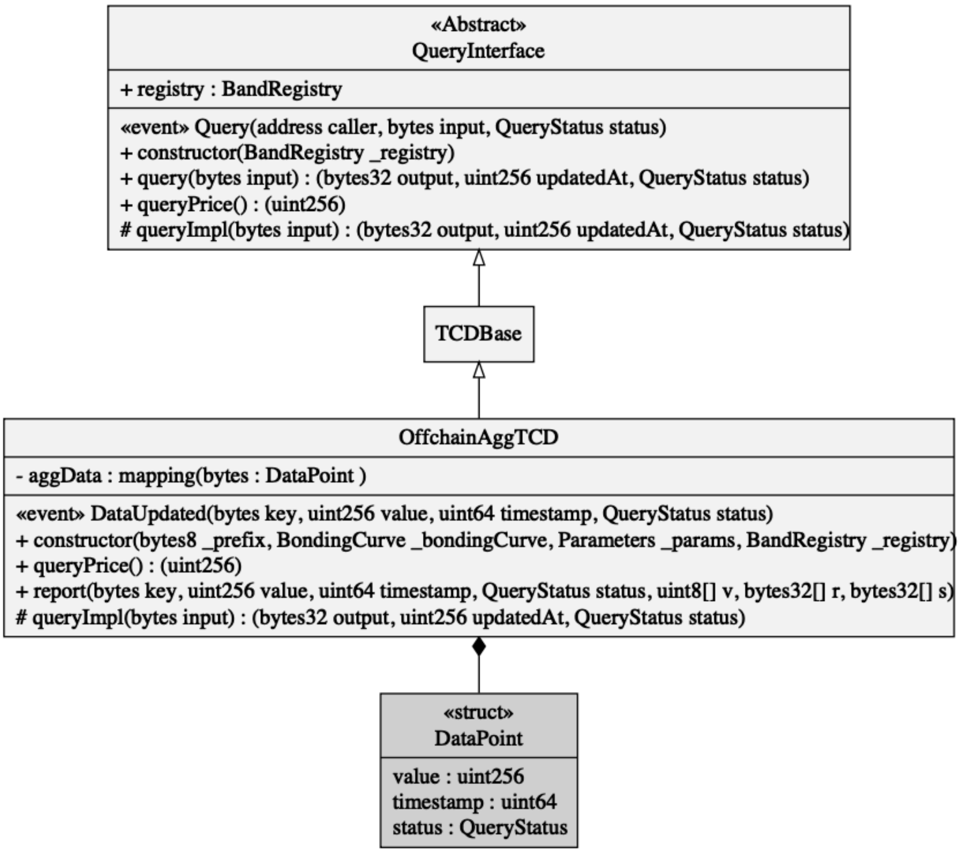

«Abstract»
QueryInterface
+ registry : BandRegistry
«event» Query(address caller, bytes input, QueryStatus status)
+ constructor(BandRegistry _registry)
+ query(bytes input) : (bytes32 output, uint256 updatedAt, QueryStatus status)
+ queryPrice() : (uint256)
# queryImpl(bytes input) : (bytes32 output, uint256 updatedAt, QueryStatus status)
TCDBase
OffchainAggTCD
- aggData : mapping(bytes : DataPoint)
«event» DataUpdated(bytes key, uint256 value, uint64 timestamp, QueryStatus status)
+ constructor(bytes8 _prefix, BondingCurve _bondingCurve, Parameters _params, BandRegistry _registry)
+ queryPrice() : (uint256)
+ report(bytes key, uint256 value, uint64 timestamp, QueryStatus status, uint8[] v, bytes32[] r, bytes32[] s)
# queryImpl(bytes input) : (bytes32 output, uint256 updatedAt, QueryStatus status)
«struct»
DataPoint
value : uint256
timestamp : uint64
status : QueryStatus

Figure 21: OffchainAggTCD

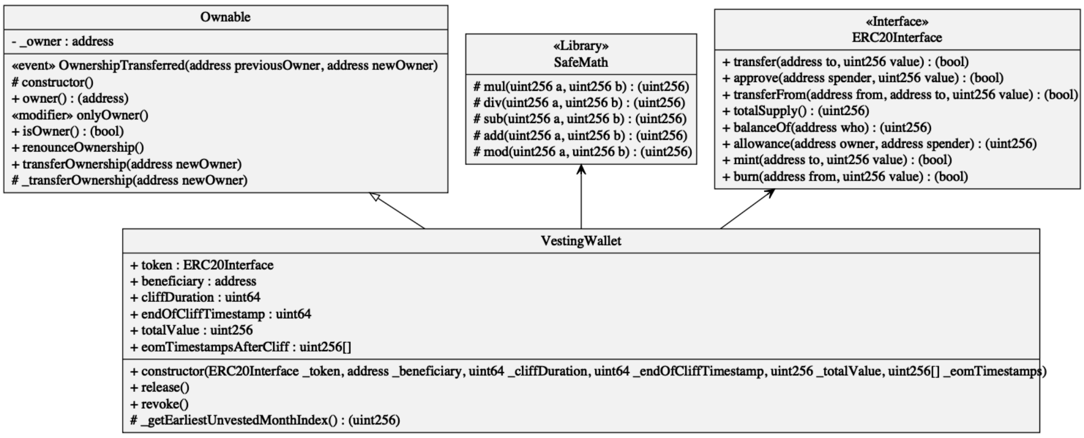

Ownable
- _owner : address
«event» OwnershipTransferred(address previousOwner, address newOwner)
# constructor()
+ owner() : (address)
«modifier» onlyOwner()
+ isOwner() : (bool)
+ renounceOwnership()
+ transferOwnership(address newOwner)
# _transferOwnership(address newOwner)
«Library»
SafeMath
# mul(uint256 a, uint256 b) : (uint256)
# div(uint256 a, uint256 b) : (uint256)
# sub(uint256 a, uint256 b) : (uint256)
# add(uint256 a, uint256 b) : (uint256)
# mod(uint256 a, uint256 b) : (uint256)
«Interface»
ERC20Interface
+ transfer(address to, uint256 value) : (bool)
+ approve(address spender, uint256 value) : (bool)
+ transferFrom(address from, address to, uint256 value) : (bool)
+ totalSupply() : (uint256)
+ balanceOf(address who) : (uint256)
+ allowance(address owner, address spender) : (uint256)
+ mint(address to, uint256 value) : (bool)
+ burn(address from, uint256 value) : (bool)
VestingWallet
+ token : ERC20Interface
+ beneficiary : address
+ cliffDuration : uint64
+ endOfCliffTimestamp : uint64
+ totalValue : uint256
+ eomTimestampsAfterCliff : uint256[]
+ constructor(ERC20Interface _token, address _beneficiary, uint64 _cliffDuration, uint64 _endOfCliffTimestamp, uint256 _totalValue, uint256[] _eomTimestamps)
+ release()
+ revoke()
# _getEarliestUnvestedMonthIndex() : (uint256)

Figure 22: VestingWallet

- INFO Recommend changing the check `require(now >= challenge.commitEndTime)` to `require(now >= challenge.revealEndTime)` in `_getChallengeResult()`. Otherwise a challenged entry proposer may try calling `resolveChallenge()` as soon as `challenge.commitEndTime` has arrived to prevent the entry being removed, if it knows the challenge will likely to succeed (e.g. the entry is a spam).

TCDBase.sol

- INFO Recommend adding a timestamp field in DataSourceInfo to store the update time of a data source for use of tie-breaking during the comparison.

VestingWallet.sol

- INFO Recommend checking the order and gaps of the vesting timestamps in `constructor()`.

MedianAggregator.sol, MedianAggregator.sol

- INFO `aggregate()`: Recommend using `SafeMath` to be more consistent with the overflow/underflow handling.

- INFO `aggregate()`: Recommend checking `require( data.length == size, array length must be the same as size)` to avoid `ArrayIndexOutOfRange` error.

Other

- INFO Recommend supplementing error messages to `require()` in the following files: `Parameters.sol`, `AggTCD.sol`, `MultiSigTCD.sol`, `OffchainAggTCD.sol`, `QueryInterface.sol`, `QueryTCR.sol`, `TCDBase.sol`, `TCRBase.sol`, `WhiteListTCR.sol`, `BondingCurve.sol`, `ERC20Acceptor.sol`, `ERC20Base.sol`, `LockableToken.sol`, `SnapshotToken.sol`, `VestingWallet.sol`, `Equation.sol`, `Aggregator.sol`.

- DISCUSSION Recommend renaming local variable `token` in contract `ERC20Acceptor` resolve state variable shadowing.

Discussions

1. Can a data provider get multiple query jobs at the same time or the data provider is only allowed to perform one job at a time?

    - (Band Protocol) Yes. Data providers are free to join multiple Token-Curated DataSources (TCDs) as long as token holders agree/stake for them and they can provide good data up to the community's standard.

2. Will the query be able to support scheduling as Oraclize? (i.e. get the result from the given query in every 60 seconds, or at an absolute time)

- (Band Protocol) Not at the moment. As mentioned above, we take the approach of making data readily available on-chain. There's no concept of callbacks at the moment.

3. Is there an incentive for data providers to stake more tokens than the `min_deposit`?

   - (Band Protocol) Yes. There are two benefits:
     - First, staking more than `min_deposit` increases the chance of the data provider to become active (top `max_provider_count` based on the number of staked tokens).
     - Second, revenue collected from data queries is split into two parts (based on `owner_revenue_pct` parameter). The first part is paid directly to the data provider. The second part is paid to stakers, including the data provider herself, proportional to the number of tokens they stake. Hence, staking more entitles them to earn more from revenue flowing through the system.

4. Is Band the initiator/organizer of all communities? Or that communities can be created by users?

   - (Band Protocol) Band serves as the organizer for the initial communities creation at least in the first year. Nevertheless communities creation should be passed to the hand of users later.

5. Is there a penalty mechanism for data provider who doesn't provide data on time frequently?

   - (Band Protocol) Yes, for those data provider, it will lose the chance of being the main provider.

6. If a user is registered as an entry but constantly revert the query after getting the data in order to get rid of the fee, how can it be removed out of the PCR entries considering users only care about themselves?

   - (Band Protocol) The goal of the contract is to provide data for smart contracts to support on-chain transactions. BandToken holders should be responsible for the health of the community so it is token holders' responsibilities to kick out malicious users. Band Protocol team is also working extensively on Whitelist improvement for preventing the middleman attack.

7. We found that the function `TransferAndCall()` not in use internally within the contracts. Can you provide the reasoning?

   - (Band Protocol) The use of this function is aligned with the ERC667 proposal.

8. What is the usage of `VestingWallet` contract in the current Band protocol ecosystem?

   - (Band Protocol) An Independent module. It is similar to the standard vesting token contract that are used for Band Protocol investors.

9. Are contracts supposed to be upgradable?

- (Band Protocol) If a new version of the contracts such as is released, a new community shall be created and the assets of the original community shall be transferred to the new contract address.

Best Practice

Solidity Protocol

✓ Use stable solidity version

✓ Handle possible errors properly when making external calls

× Provide error message along with require()

✓ Use modifiers properly

✓ Use events to monitor contract activities

✓ Refer and use libraries properly

✓ No compiler warnings

Privilege Control

✓ Provide pause functionality for control and emergency handling

✓ Restrict access to sensitive functions

Documentation

✓ Provide project readme and execution guidance

✓ Provide inline comment for function intention

✓ Provide instruction to initialize and execute the test files

Testing

✓ Provide migration scripts

✓ Provide test scripts and coverage for potential scenarios

With the final update of source code and delivery of the audit report, CertiK is able to conclude that the Band Protocol contracts are not vulnerable to any classically known anti-patterns or security issues.

While this CertiK review is a strong and positive indication, the audit report itself is not necessarily a guarantee of correctness or trustworthiness. CertiK always recommends seeking multiple opinions, test coverage, sandbox deployments before any mainnet release.

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 1 in File Parameters.sol

```
1 pragma solidity 0.5.9;
```

ⓘ Only these compiler versions are safe to compile your code: 0.5.9

TIMESTAMP_DEPENDENCY

Line 237 in File Parameters.sol

```
237     require(now < proposal.expirationTime);
```

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

TIMESTAMP_DEPENDENCY

Line 277 in File Parameters.sol

```
277     require(now >= proposal.expirationTime);
```

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

INSECURE_COMPILER_VERSION

Line 1 in File BandRegistry.sol

```
1 pragma solidity 0.5.9;
```

ⓘ Only these compiler versions are safe to compile your code: 0.5.9

INSECURE_COMPILER_VERSION

Line 1 in File ERC20Base.sol

```
1 pragma solidity 0.5.9;
```

ⓘ Only these compiler versions are safe to compile your code: 0.5.9

INSECURE_COMPILER_VERSION

Line 1 in File SnapshotToken.sol

```
1 pragma solidity 0.5.9;
```

ⓘ Only these compiler versions are safe to compile your code: 0.5.9

INSECURE_COMPILER_VERSION

Line 1 in File LockableToken.sol

```
1 pragma solidity 0.5.9;
```

ⓘ Only these compiler versions are safe to compile your code: 0.5.9

INSECURE_COMPILER_VERSION

Line 1 in File VestingWallet.sol

```
1 pragma solidity 0.5.9;
```

ℹ Only these compiler versions are safe to compile your code: 0.5.9

TIMESTAMP_DEPENDENCY

Line 67 in File VestingWallet.sol

```
67     require(now > endOfCliffTimestamp);
```

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

TIMESTAMP_DEPENDENCY

Line 99 in File VestingWallet.sol

```
99       if (now < eomTimestampsAfterCliff[i]) return i;
```

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

INSECURE_COMPILER_VERSION

Line 1 in File BondingCurve.sol

```
1 pragma solidity 0.5.9;
```

ℹ Only these compiler versions are safe to compile your code: 0.5.9

TIMESTAMP_DEPENDENCY

Line 116 in File BondingCurve.sol

```
116     if (lastInflationTime < now) {
```

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

TIMESTAMP_DEPENDENCY

Line 124 in File BondingCurve.sol

```
124     lastInflationTime = now;
```

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

INSECURE_COMPILER_VERSION

Line 1 in File Fractional.sol

```
1 pragma solidity 0.5.9;
```

ℹ Only these compiler versions are safe to compile your code: 0.5.9

INSECURE_COMPILER_VERSION

Line 1 in File Aggregator.sol

```
1 pragma solidity 0.5.9;
```

ℹ Only these compiler versions are safe to compile your code: 0.5.9

INSECURE_COMPILER_VERSION

Line 1 in File MultiSigTCD.sol

```
1   pragma solidity 0.5.9;
```

ℹ Only these compiler versions are safe to compile your code: 0.5.9

INSECURE_COMPILER_VERSION

Line 1 in File TCRBase.sol

```
1   pragma solidity 0.5.9;
```

ℹ Only these compiler versions are safe to compile your code: 0.5.9

TIMESTAMP_DEPENDENCY

Line 105 in File TCRBase.sol

```
105     return listedAt != 0 && now >= listedAt;
```

❗ "now" can be influenced by minors to some degree

TIMESTAMP_DEPENDENCY

Line 129 in File TCRBase.sol

```
129     if (now < entry.listedAt) {
```

❗ "now" can be influenced by minors to some degree

TIMESTAMP_DEPENDENCY

Line 310 in File TCRBase.sol

```
310     require(challenge.state == ChallengeState.Open && now < challenge.commitEndTime);
```

❗ "now" can be influenced by minors to some degree

TIMESTAMP_DEPENDENCY

Line 334 in File TCRBase.sol

```
334     require(now >= challenge.commitEndTime && now < challenge.revealEndTime);
```

❗ "now" can be influenced by minors to some degree

TIMESTAMP_DEPENDENCY

Line 334 in File TCRBase.sol

```
334     require(now >= challenge.commitEndTime && now < challenge.revealEndTime);
```

❗ "now" can be influenced by minors to some degree

TIMESTAMP_DEPENDENCY

Line 495 in File TCRBase.sol

```
495     require(now >= challenge.commitEndTime);
```

❗ "now" can be influenced by minors to some degree

INSECURE_COMPILER_VERSION

Line 1 in File AggTCD.sol

```
1   pragma solidity 0.5.9;
```

ℹ Only these compiler versions are safe to compile your code: 0.5.9

INSECURE_COMPILER_VERSION

Line 1 in File TCDBase.sol

```
1   pragma solidity 0.5.9;
```

ℹ Only these compiler versions are safe to compile your code: 0.5.9

TIMESTAMP_DEPENDENCY

Line 263 in File TCDBase.sol

```
263       require(!receipt.isWithdrawn && now >= receipt.withdrawTime);
```

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

INSECURE_COMPILER_VERSION

Line 1 in File OffchainAggTCD.sol

```
1   pragma solidity 0.5.9;
```

ℹ Only these compiler versions are safe to compile your code: 0.5.9

TIMESTAMP_DEPENDENCY

Line 76 in File OffchainAggTCD.sol

```
76        require(timestamp > aggData[key].timestamp && uint256(timestamp) <= now);
```

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

Source Code with CertiK Labels

File Parameters.sol

```
pragma solidity 0.5.9;

import "openzeppelin-solidity/contracts/ownership/Ownable.sol";
import "openzeppelin-solidity/contracts/math/SafeMath.sol";
import "./token/SnapshotToken.sol";
import "./utils/Fractional.sol";


/// "Parameters" contract controls how other smart contracts behave through a key-value mapping, which other contracts
/// will query using ‘get‘ or ‘getRaw‘ functions. Every dataset community has one governance parameters contract.
/// Additionally, there is one parameter contract that is controlled by BandToken for protocol-wide parameters.
/// Conducting parameter changes can be done through the following process.
///   1. Anyone can propose for a change by sending a ‘propose‘ transaction, which will assign an ID to the proposal.
///   2. While the proposal is open, token holders can vote for approval or rejection through ‘vote‘ function.
///   3. After the voting period ends, if the proposal receives enough participation and support, it will get accepted.
///      ‘resolve‘ function must to be called to trigger the decision process.
///   4. Additionally, to facilitate unanimous parameter changes, a proposal is automatically resolved prior to its
///      expiration if more than the required percentage of ALL tokens approve the proposal.
/// Parameters contract uses the following parameters for its internal logic. These parameters can be change via the
/// same proposal process.
///   ‘params:expiration_time‘: Number of seconds that a proposal stays open after getting proposed.
///   ‘params:min_participation_pct‘: % of tokens required to participate in order for a proposal to be considered.
///   ‘params:support_required_pct‘: % of participating tokens required to approve a proposal.
/// Parameters contract is "Ownable" initially to allow its owner to overrule the parameters during the initial
/// deployment as a measure against possible smart contract vulnerabilities. Owner can be set to 0x0 address afterwards.
contract Parameters is Ownable {
  using SafeMath for uint256;
  using Fractional for uint256;

  event ProposalProposed(uint256 indexed proposalId, address indexed proposer, bytes32 reasonHash);
  event ProposalVoted(uint256 indexed proposalId, address indexed voter, bool vote, uint256 votingPower);
  event ProposalAccepted(uint256 indexed proposalId);
  event ProposalRejected(uint256 indexed proposalId);
  event ParameterChanged(bytes32 indexed key, uint256 value);
  event ParameterProposed(uint256 indexed proposalId, bytes32 indexed key, uint256 value);

  struct ParameterValue { bool existed; uint256 value; }
  struct KeyValue { bytes32 key; uint256 value; }
  enum ProposalState { INVALID, OPEN, ACCEPTED, REJECTED }

  struct Proposal {
    uint256 changesCount;               /// The number of parameter changes
    mapping (uint256 => KeyValue) changes; /// The list of parameter changes in proposal
    uint256 snapshotNonce;              /// The votingPowerNonce to count voting power
    uint256 expirationTime;             /// The time at which this proposal resolves
    uint256 voteSupportRequiredPct;     /// Threshold % for determining proposal acceptance
    uint256 voteMinParticipation;       /// The minimum # of votes required
    uint256 totalVotingPower;           /// The total voting power at this snapshotNonce
    uint256 yesCount;                   /// The current total number of YES votes
    uint256 noCount;                    /// The current total number of NO votes
    mapping (address => bool) isVoted;  /// Mapping for check who already voted
    ProposalState proposalState;        /// Current state of this proposal.
  }

  SnapshotToken public token;
  Proposal[] public proposals;
  mapping (bytes32 => ParameterValue) public params;

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK Parameters
    @tag assume_completion
    @post __post.token == _token
   */
  constructor(SnapshotToken _token) public {
    token = _token;
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK get
    @tag assume_completion
    @post params[rawKey].existed
    @post __return == params[bytes32(namespace) | (bytes32(key) >> (8 * __post.namespaceSize))].value
   */
  function get(bytes8 namespace, bytes24 key) public view returns (uint256) {
    uint8 namespaceSize = 0;
    /*@CTK loop_get
      @post (namespaceSize == 8) || (namespace[namespaceSize] == byte(0))
      @post !__should_return
     */
    while (namespaceSize < 8 && namespace[namespaceSize] != byte(0)) ++namespaceSize;
    return getRaw(bytes32(namespace) | (bytes32(key) >> (8 * namespaceSize)));
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK getRaw
```

```
    @tag assume_completion
    @post params[rawKey].existed
    @post __return == params[rawKey].value
   */
  function getRaw(bytes32 rawKey) public view returns (uint256) {
    ParameterValue storage param = params[rawKey];
    require(param.existed);
    return param.value;
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK set
    @tag assume_completion
    @post keys.length == values.length
   */
  function set(bytes8 namespace, bytes24[] memory keys, uint256[] memory values) public onlyOwner {
    require(keys.length == values.length);
    bytes32[] memory rawKeys = new bytes32[](keys.length);
    uint8 namespaceSize = 0;
    /*@CTK loop_set_namespaceSize
      @post (namespaceSize == 8) || (namespace[namespaceSize] == byte(0))
      @post !__should_return
     */
    while (namespaceSize < 8 && namespace[namespaceSize] != byte(0)) ++namespaceSize;
    /*@CTK loop_set
      @inv i <= keys.length
      @post i == keys.length
      @inv rawKeys[i] != 0
      @post !__should_return
     */
    for (uint256 i = 0; i < keys.length; i++) {
      rawKeys[i] = bytes32(namespace) | bytes32(keys[i]) >> (8 * namespaceSize);
    }
    setRaw(rawKeys, values);
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK setRaw
    @tag assume_completion
    @post rawKeys.length == values.length
    @post forall j: uint256. (j >= 0 /\ j < rawKeys.length) -> (params[rawKeys[i]].existed == true) && (params[rawKeys[i]].value == values[i])
   */
  function setRaw(bytes32[] memory rawKeys, uint256[] memory values) public onlyOwner {
    require(rawKeys.length == values.length);
    /*@CTK loop_setRaw
      @inv i <= rawKeys.length
      @post i == rawKeys.length
      @post !__should_return
     */
    for (uint256 i = 0; i < rawKeys.length; i++) {
      params[rawKeys[i]].existed = true;
      params[rawKeys[i]].value = values[i];
      emit ParameterChanged(rawKeys[i], values[i]);
    }
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK getProposalChange
    @tag assume_completion
    @post __return == (proposals[proposalId].changes[changeIndex].key, proposals[proposalId].changes[changeIndex].value)
   */
  function getProposalChange(uint256 proposalId, uint256 changeIndex) public view returns (bytes32, uint256) {
    KeyValue memory keyValue = proposals[proposalId].changes[changeIndex];
    return (keyValue.key, keyValue.value);
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK propose
    @tag assume_completion
    @post keys.length == values.length
    @post forall j: uint256. (j >= 0 /\ j < keys.length) -> __post.proposals[proposals.length].changes[j] == KeyValue({key: keys[j], value: values[j]})
   */
  function propose(bytes32 reasonHash, bytes32[] calldata keys, uint256[] calldata values) external {
    require(keys.length == values.length);
    uint256 proposalId = proposals.length;
    Proposal newProposal = Proposal();
    newProposal.changesCount = keys.length;
    newProposal.snapshotNonce = token.votingPowerChangeNonce();
    newProposal.expirationTime = now.add(getRaw("params:expiration_time"));
    newProposal.voteSupportRequiredPct = getRaw("params:support_required_pct");
    newProposal.voteMinParticipation = getRaw("params:min_participation_pct").mulFrac(token.totalSupply());
    newProposal.totalVotingPower = token.totalSupply();
    newProposal.yesCount = 0;
    newProposal.noCount = 0;
    newProposal.proposalState = ProposalState.OPEN;
    proposals.push(newProposal);
    emit ProposalProposed(proposalId, msg.sender, reasonHash);
    /*@CTK loop_propose
      @inv index <= keys.length
      @post index == keys.length
      @post !__should_return
     */
    for (uint256 index = 0; index < keys.length; ++index) {
      bytes32 key = keys[index];
      uint256 value = values[index];
      emit ParameterProposed(proposalId, key, value);
      proposals[proposalId].changes[index] = KeyValue(key, value);
    }
  }

```

```
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK vote_accept
  @tag assume_completion
  @pre accepted
  @post (proposals[proposalId].proposalState == ProposalState.OPEN)
  @post (now < proposals[proposalId].expirationTime)
  @post (!proposals[proposalId].isVoted[msg.sender])
  @post __post.proposals[proposalId].yesCount >= proposals[proposalId].yesCount
  @post __post.proposals[proposalId].noCount == proposals[proposalId].noCount
  @post __post.proposals[proposalId].isVoted[msg.sender] == true
  @post (proposals[proposalId].yesCount >= __post.minVoteToAccept) -> (__post.proposals[proposalId] == ProposalState.ACCEPTED)
  @post (proposals[proposalId].yesCount < __post.minVoteToAccept) && (proposals[proposalId].noCount > __post.minVoteToReject) -> (__post.proposals[proposalId] == ProposalState.REJECTED)
 */
/*@CTK vote_reject
  @tag assume_completion
  @pre !accepted
  @post (proposals[proposalId].proposalState == ProposalState.OPEN)
  @post (now < proposals[proposalId].expirationTime)
  @post (!proposals[proposalId].isVoted[msg.sender])
  @post __post.proposals[proposalId].yesCount == proposals[proposalId].yesCount
  @post __post.proposals[proposalId].noCount >= proposals[proposalId].noCount
  @post __post.proposals[proposalId].isVoted[msg.sender] == true
  @post (proposals[proposalId].yesCount >= __post.minVoteToAccept) -> (__post.proposals[proposalId] == ProposalState.ACCEPTED)
  @post (proposals[proposalId].yesCount < __post.minVoteToAccept) && (proposals[proposalId].noCount > __post.minVoteToReject) -> (__post.proposals[proposalId] == ProposalState.REJECTED)
 */
function vote(uint256 proposalId, bool accepted) public {
  Proposal storage proposal = proposals[proposalId];
  require(proposal.proposalState == ProposalState.OPEN);
  require(now < proposal.expirationTime);
  require(!proposal.isVoted[msg.sender]);
  uint256 votingPower = token.historicalVotingPowerAtNonce(msg.sender, proposal.snapshotNonce);
  require(votingPower > 0);
  if (accepted) {
    proposal.yesCount = proposal.yesCount.add(votingPower);
  } else {
    proposal.noCount = proposal.noCount.add(votingPower);
  }
  proposal.isVoted[msg.sender] = true;
  emit ProposalVoted(proposalId, msg.sender, accepted, votingPower);
  uint256 minVoteToAccept = proposal.voteSupportRequiredPct.mulFrac(proposal.totalVotingPower);
  uint256 minVoteToReject = proposal.totalVotingPower.sub(minVoteToAccept);
  if (proposal.yesCount >= minVoteToAccept) {
    _acceptProposal(proposalId);
  } else if (proposal.noCount > minVoteToReject) {
    _rejectProposal(proposalId);
  }
}
```

```
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK resolve_early_accept
  @tag assume_completion
  @pre proposals[proposalId].proposalState == ProposalState.OPEN
  @pre now >= proposals[proposalId].expirationTime
  @pre ((proposals[proposalId].yesCount + proposals[proposalId].noCount) >= proposals[proposalId].voteMinParticipation) && (proposals[proposalId].yesCount * 1000000000000000000 >= proposals[proposalId].voteSupportRequiredPct * (proposals[proposalId].yesCount + proposals[proposalId].noCount))
  @post __post.proposals[proposalId].proposalState == ProposalState.ACCEPTED
 */
/*@CTK resolve_early_reject
  @tag assume_completion
  @pre proposals[proposalId].proposalState == ProposalState.OPEN
  @pre now >= proposals[proposalId].expirationTime
  @pre ((proposals[proposalId].yesCount + proposals[proposalId].noCount) < proposals[proposalId].voteMinParticipation) || (proposals[proposalId].yesCount * 1000000000000000000 < proposals[proposalId].voteSupportRequiredPct * (proposals[proposalId].yesCount + proposals[proposalId].noCount))
  @post __post.proposals[proposalId] == ProposalState.REJECTED
 */
function resolve(uint256 proposalId) public {
  Proposal storage proposal = proposals[proposalId];
  require(proposal.proposalState == ProposalState.OPEN);
  require(now >= proposal.expirationTime);
  uint256 yesCount = proposal.yesCount;
  uint256 noCount = proposal.noCount;
  uint256 totalCount = yesCount.add(noCount);
  if (totalCount >= proposal.voteMinParticipation &&
      yesCount.mul(Fractional.getDenominator()) >= proposal.voteSupportRequiredPct.mul(totalCount)) {
    _acceptProposal(proposalId);
  } else {
    _rejectProposal(proposalId);
  }
}

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK _acceptProposal
  @tag assume_completion
  @post __post.proposals[proposalId].proposalState == ProposalState.ACCEPTED
  @post forall j: uint256.(j >= 0 /\ j < proposal.changesCount) -> (params[proposal.changes[j].key].existed == true) && (params[proposal.changes[j].key].value == proposal.changes[j].value)
 */
function _acceptProposal(uint256 proposalId) internal {
  Proposal storage proposal = proposals[proposalId];
  proposal.proposalState = ProposalState.ACCEPTED;
  /*@CTK loop_propose
    @inv index <= proposal.changesCount
    @post index == proposal.changesCount
    @post !__should_return
   */
  for (uint256 index = 0; index < proposal.changesCount; ++index) {
      bytes32 key = proposal.changes[index].key;
      uint256 value = proposal.changes[index].value;
      params[key].existed = true;
      params[key].value = value;
      emit ParameterChanged(key, value);
    }
    emit ProposalAccepted(proposalId);
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK _rejectProposal
    @tag assume_completion
    @post __post.proposals[proposalId] == ProposalState.REJECTED
   */
  function _rejectProposal(uint256 proposalId) internal {
    Proposal storage proposal = proposals[proposalId];
    proposal.proposalState = ProposalState.REJECTED;
    emit ProposalRejected(proposalId);
  }
}
```

File BandRegistry.sol

```
pragma solidity 0.5.9;

import "openzeppelin-solidity/contracts/ownership/Ownable.sol";
import "./BandToken.sol";
import "./data/WhiteListInterface.sol";
import "./exchange/BandExchangeInterface.sol";


/// "BandRegistry" keeps the addresses of three main smart contracts inside of Band Protocol ecosystem:
///   1. "band" - Band Protocol's native ERC-20 token.
///   2. "exchange" - Decentralized exchange for converting ETH to Band and vice versa.
///   3. "whiteList" - Smart contract for validating non-malicious data consumers.
contract BandRegistry is Ownable {
  BandToken public band;
  BandExchangeInterface public exchange;
  WhiteListInterface public whiteList;

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK BandRegistry
    @tag assume_completion
    @post __post.band == _band
    @post __post.exchange == _exchange
   */
  constructor(BandToken _band, BandExchangeInterface _exchange) public {
    band = _band;
    exchange = _exchange;
  }

  /*@CTK verify_unconditional
    @pre address(whiteList) == 0
    @post __return == true
   */
  function verify(address reader) public view returns (bool) {
    if (address(whiteList) == address(0)) return true;
    return whiteList.verify(reader);
  }

  /*@CTK setWhiteList
    @tag assume_completion
    @post __post.whiteList == _whiteList
   */
  function setWhiteList(WhiteListInterface _whiteList) public onlyOwner {
    whiteList = _whiteList;
  }

  /*@CTK setExchange
    @tag assume_completion
    @post __post.exchange == _exchange
   */
  function setExchange(BandExchangeInterface _exchange) public onlyOwner {
    exchange = _exchange;
  }
}
```

File token/ERC20Base.sol

```
pragma solidity 0.5.9;

import "openzeppelin-solidity/contracts/token/ERC20/ERC20.sol";
import "openzeppelin-solidity/contracts/access/roles/MinterRole.sol";
import "./ERC20Interface.sol";


/// "ERC20Base" is the standard ERC-20 implementation that allows its minter to mint tokens. Both BandToken and
/// CommunityToken extend from ERC20Base. In addition to the standard functions, the class provides ‘transferAndCall‘
/// function, which performs a transfer and invokes the given function using the provided data. If the destination
/// contract uses "ERC20Acceptor" interface, it can verify that the caller properly sends appropriate amount of tokens.
contract ERC20Base is ERC20Interface, ERC20, MinterRole {
  string public name;
  string public symbol;
  uint8 public decimals = 18;

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK ERC20Base
    @tag assume_completion
    @post __post.name == _name
    @post __post.symbol == _symbol
   */
  constructor(string memory _name, string memory _symbol) public {
    name = _name;
    symbol = _symbol;
  }

  function transferAndCall(address to, uint256 value, bytes4 sig, bytes memory data) public returns (bool) {
    require(to != address(this));
    _transfer(msg.sender, to, value);
    (bool success,) = to.call(abi.encodePacked(sig, uint256(msg.sender), value, data));
    require(success);
    return true;
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK ERC20Base_mint
    @tag assume_completion
    @pre _minters.bearer[msg.sender]
    @post (to != address(0))
    @post (__post._totalSupply) == ((_totalSupply) + (value))
    @post (__post._balances[to]) == ((_balances[to]) + (value))
    @post __return
   */
  function mint(address to, uint256 value) public onlyMinter returns (bool) {
    _mint(to, value);
    return true;
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK ERC20Base_burn
    @tag assume_completion
    @pre _minters.bearer[msg.sender]
    @post (value <= _balances[from])
    @post (__post._totalSupply) == ((_totalSupply) - (value))
    @post (__post._balances[from]) == ((_balances[from]) - (value))
   */
  function burn(address from, uint256 value) public onlyMinter returns (bool) {
    _burn(from, value);
    return true;
  }
}
```

File token/SnapshotToken.sol

```
pragma solidity 0.5.9;

import "openzeppelin-solidity/contracts/math/SafeMath.sol";
import "./ERC20Base.sol";


contract SnapshotToken is ERC20Base {
  using SafeMath for uint256;

  /// IMPORTANT: votingPowers are kept as a linked list of ALL historical changes.
  /// - This allows the contract to figure out voting power of the address at any nonce ‘n‘, by
  /// searching for the node that has the biggest nonce that is not greater than ‘n‘.
  /// - For efficiency, nonce and power are packed into one uint256 integer, with the top 64 bits
/// representing nonce, and the bottom 192 bits representing voting power.
mapping (address => mapping(uint256 => uint256)) _votingPower;
mapping (address => uint256) public votingPowerChangeCount;
uint256 public votingPowerChangeNonce = 0;

/// Returns user voting power at the given index, that is, as of the user’s index^th voting power change
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK historicalVotingPowerAtIndex
  @tag assume_completion
  @post index <= votingPowerChangeCount[owner]
  @post __return == _votingPower[owner][index] & ((1 << 192) - 1)
 */
function historicalVotingPowerAtIndex(address owner, uint256 index) public view returns (uint256) {
  require(index <= votingPowerChangeCount[owner]);
  return _votingPower[owner][index] & ((1 << 192) - 1); // Lower 192 bits
}

/// Returns user voting power at the given time. Under the hood, this performs binary search
/// to look for the largest index at which the nonce is not greater than ’nonce’.
/// The voting power at that index is the returning value.
/*@CTK historicalVotingPowerAtNonce
  @tag assume_completion
  @post nonce <= votingPowerChangeNonce && nonce < (1 << 64)
 */
function historicalVotingPowerAtNonce(address owner, uint256 nonce) public view returns (uint256) {
  require(nonce <= votingPowerChangeNonce && nonce < (1 << 64));
  uint256 start = 0;
  uint256 end = votingPowerChangeCount[owner];
  /*@CTK loop_historicalVotingPowerAtNonce
    @inv start >= 0
    @inv end <= votingPowerChangeCount[owner]
    @inv start >= start__pre
    @inv end < end__pre
    @inv mid != mid__pre
    @post start >= end
    @post (_votingPower[owner][start] >> 192) <= nonce
    @post !__should_return
   */
  while (start < end) {
    uint256 mid = start.add(end).add(1).div(2); /// Use (start+end+1)/2 to prevent infinite loop.
    if ((_votingPower[owner][mid] >> 192) > nonce) { /// Upper 64-bit nonce
      /// If midTime > nonce, this mid can’t possibly be the answer.
      end = mid.sub(1);
    } else {
      /// Otherwise, search on the greater side, but still keep mid as a possible option.
      start = mid;
    }
  }
  return historicalVotingPowerAtIndex(owner, start);
}
```

```

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK Snapshot_transfer
  @tag assume_completion
  @post __post.votingPowerChangeNonce == votingPowerChangeNonce + 2
  @post __post._balances[from] == _balances[from] - value
  @post __post._balances[to] == _balances[to] + value
  @post __post.votingPowerChangeCount[from] == votingPowerChangeCount[from] + 1
  @post __post.votingPowerChangeCount[to] == votingPowerChangeCount[to] + 1
  @post __post._votingPower[from].length == _votingPower[from].length + 1
  @post __post._votingPower[to].length == _votingPower[to].length + 1
 */
function _transfer(address from, address to, uint256 value) internal {
  super._transfer(from, to, value);
  votingPowerChangeNonce = votingPowerChangeNonce.add(1);
  _changeVotingPower(from);
  _changeVotingPower(to);
}

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK Snapshot_mint
  @tag assume_completion
  @post __post.votingPowerChangeNonce == votingPowerChangeNonce + 2
  @post __post._balances[account] == _balances[account] + value
  @post __post._totalSupply == __post._totalSupply + value
  @post __post.votingPowerChangeCount[account] == votingPowerChangeCount[account] +
      1
  @post __post._votingPower[account].length == _votingPower[account].length + 1
 */
function _mint(address account, uint256 amount) internal {
  super._mint(account, amount);
  votingPowerChangeNonce = votingPowerChangeNonce.add(1);
  _changeVotingPower(account);
}

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK Snapshot_burn
  @tag assume_completion
  @post __post.votingPowerChangeNonce == votingPowerChangeNonce + 1
  @post __post._balances[account] == _balances[account] - value
  @post __post._totalSupply == __post._totalSupply - value
  @post __post.votingPowerChangeCount[account] == votingPowerChangeCount[account] +
      1
  @post __post._votingPower[account].length == _votingPower[account].length + 1
 */
function _burn(address account, uint256 amount) internal {
  super._burn(account, amount);
  votingPowerChangeNonce = votingPowerChangeNonce.add(1);
  _changeVotingPower(account);
}

//@CTK NO_OVERFLOW
```

```
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK _changeVotingPower
    @tag assume_completion
    @post _balances[account] < (1 << 192)
    @post votingPowerChangeNonce < (1 << 64)
    @post __post.votingPowerChangeCount[account] == votingPowerChangeCount[account] + 1
    @post _votingPower[account][__post.votingPowerChangeCount[account]] == ((votingPowerChangeNonce << 192) | __post._balances[account])
    @post __post._balances[account] == _balances[account]
   */
  function _changeVotingPower(address account) internal {
    uint256 currentIndex = votingPowerChangeCount[account];
    uint256 newPower = balanceOf(account);
    require(newPower < (1 << 192));
    require(votingPowerChangeNonce < (1 << 64));
    currentIndex = currentIndex.add(1);
    votingPowerChangeCount[account] = currentIndex;
    _votingPower[account][currentIndex] = (votingPowerChangeNonce << 192) | newPower;
  }
}
```

File token/LockableToken.sol

```
pragma solidity 0.5.9;

import "openzeppelin-solidity/contracts/math/SafeMath.sol";
import "openzeppelin-solidity/contracts/math/Math.sol";
import "openzeppelin-solidity/contracts/access/roles/CapperRole.sol";
import "./ERC20Base.sol";


/// "LockableToken" adds token locking functionality to ERC-20 smart contract. The authorized addresses (Cappers) are
/// allowed to lock tokens from any token holder to prevent token transfers up to that amount. If a token holder is
/// locked by multiple cappers, the maximum number is used as the amount of locked tokens.
contract LockableToken is ERC20Base, CapperRole {
  using SafeMath for uint256;

  event TokenLocked(address indexed locker, address indexed owner, uint256 value);
  event TokenUnlocked(address indexed locker, address indexed owner, uint256 value);

  uint256 constant NOT_FOUND = uint256(-1);

  struct TokenLock {
    address locker;
    uint256 value;
  }

  mapping (address => TokenLock[]) _locks;

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK getLockedToken
    @tag assume_completion
    @post forall j: uint. (j >= 0 /\ j < locks.length) -> __return >= _locks[owner][j].value
   */
  function getLockedToken(address owner) public view returns (uint256) {
    TokenLock[] storage locks = _locks[owner];
    uint256 maxLock = 0;
    /*@CTK loop_getLockedToken
      @inv i <= locks.length
      @post i == locks.length
      @post forall j: uint. (j >= 0 /\ j < locks.length) -> maxLock >= locks[j].value
      @post !__should_return
     */
    for (uint256 i = 0; i < locks.length; ++i) {
      maxLock = Math.max(maxLock, locks[i].value);
    }
    return maxLock;
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK getLockedTokenAt_FOUND
    @tag assume_completion
    @pre index != NOT_FOUND
    @post __return == _locks[owner][index].value
   */
   /*@CTK getLockedTokenAt_NOT_FOUND
     @tag assume_completion
     @pre index == NOT_FOUND
     @post __return == 0
    */
  function getLockedTokenAt(address owner, address locker) public view returns (uint256) {
    uint256 index = _getTokenLockIndex(owner, locker);
    if (index != NOT_FOUND) return _locks[owner][index].value;
    else return 0;
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK unlockedBalanceOf
    @tag assume_completion
    @post __return <= balanceOf(owner)
   */
  function unlockedBalanceOf(address owner) public view returns (uint256) {
    return balanceOf(owner).sub(getLockedToken(owner));
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK lock_NOT_FOUND
    @tag assume_completion
    @pre index != NOT_FOUND
    @post balanceOf(owner) >= (value + _locks[owner][index].value)
    @post __post._locks[owner][index].locker == msg.sender
    @post __post._locks[owner][index].value == (_locks[owner][index].value + value)
   */
  /*@CTK lock_FOUND
    @tag assume_completion
    @pre index == NOT_FOUND
    @post balanceOf(owner) >= value
    @post __post._locks[owner][_locks[owner].length].locker == msg.sender
    @post __post._locks[owner][_locks[owner].length].value == value
   */
  function lock(address owner, uint256 value) public onlyCapper returns (bool) {
    uint256 index = _getTokenLockIndex(owner, msg.sender);
    if (index != NOT_FOUND) {
      uint256 currentLock = _locks[owner][index].value;
      require(balanceOf(owner) >= currentLock.add(value));
      _locks[owner][index].value = currentLock.add(value);
    } else {
      require(balanceOf(owner) >= value);
      _locks[owner].push(TokenLock(msg.sender, value));
    }
    emit TokenLocked(msg.sender, owner, value);
    return true;
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK unlock
    @tag assume_completion
    @post index != NOT_FOUND
    @post _locks[owner][index].value >= value
    @post __post._locks[owner][index].value == _locks[owner][index].value - value
    @post __post._locks[owner].length == _locks[owner].length - 1
   */
  function unlock(address owner, uint256 value) public returns (bool) {
    uint256 index = _getTokenLockIndex(owner, msg.sender);
    require(index != NOT_FOUND);
    TokenLock[] storage locks = _locks[owner];
    require(locks[index].value >= value);
    locks[index].value = locks[index].value.sub(value);
    if (locks[index].value == 0) {
      if (index != locks.length - 1) {
        locks[index] = locks[locks.length - 1];
      }
      locks.pop();
    }
    emit TokenUnlocked(msg.sender, owner, value);
    return true;
  }

  /*@CTK _getTokenLockIndex
    @tag assume_completion
    @post (__return == NOT_FOUND) || (_locks[owner][__return].locker == locker)
   */
  function _getTokenLockIndex(address owner, address locker) internal view returns (uint256) {
    TokenLock[] storage locks = _locks[owner];
    /*@CTK loop_getTokenLockIndex
      @inv i <= locks.length
      @post (i < locks.length) -> (_locks[owner][i].locker == locker)
```

```
      @post !__should_return
     */
    for (uint256 i = 0; i < locks.length; ++i) {
      if (locks[i].locker == locker) return i;
    }
    return NOT_FOUND;
  }

  function _transfer(address from, address to, uint256 value) internal {
    require(unlockedBalanceOf(from) >= value);
    super._transfer(from, to, value);
  }

  function _burn(address account, uint256 value) internal {
    require(unlockedBalanceOf(account) >= value);
    super._burn(account, value);
  }
}
```

File token/VestingWallet.sol

```
pragma solidity 0.5.9;

import "openzeppelin-solidity/contracts/ownership/Ownable.sol";
import "openzeppelin-solidity/contracts/math/SafeMath.sol";
import "./ERC20Interface.sol";


contract VestingWallet is Ownable {
  using SafeMath for uint256;

  ERC20Interface public token;
  address public beneficiary;
  uint64 public cliffDuration;            /// Duration of the cliff, in months
  uint64 public endOfCliffTimestamp;      /// Timestamp when cliff ends that token starts vesting
  uint256 public totalValue;              /// Total vesting token
  uint256[] public eomTimestampsAfterCliff; /// Timestamps of all vesting months

  /*@CTK vesting_wallet
    @tag assume_completion
    @post __post.token == _token
    @post __post.beneficiary == _beneficiary
    @post __post.cliffDuration == _cliffDuration
    @post __post.endOfCliffTimestamp == _endOfCliffTimestamp
    @post __post.totalValue == _totalValue
    @post forall j: uint256. (j >= 0 /\ j < _eomTimestamps.length) -> __post.eomTimestampsAfterCliff[j] == _eomTimestamps[j]
    @post forall j: uint256. (j >= 0 /\ j < _eomTimestamps.length - 1) -> __post.eomTimestampsAfterCliff[j] <= __post.eomTimestampsAfterCliff[j + 1]
   */
  constructor(
    ERC20Interface _token,
    address _beneficiary,
    uint64 _cliffDuration,
    uint64 _endOfCliffTimestamp,
    uint256 _totalValue,
    uint256[] memory _eomTimestamps
  ) public {
    token = _token;
    beneficiary = _beneficiary;
    cliffDuration = _cliffDuration;
    endOfCliffTimestamp = _endOfCliffTimestamp;
    totalValue = _totalValue;
    /*@CTK loop_VestingWallet_Cliff
      @inv i <= _eomTimestamps.length
      @post i == _eomTimestamps.length
      @post !__should_return
     */
    for (uint256 i = 0; i < _eomTimestamps.length; ++i) {
      eomTimestampsAfterCliff.push(_eomTimestamps[i]);
    }
  }

  /*@CTK vesting_release
    @tag assume_completion
    @pre now > endOfCliffTimestamp
    @post eomTimestampsAfterCliff[__post.earliestUnvestedMonthIndex] > now
    @post (eomTimestampsAfterCliff.length - earliestUnvestedMonthIndex)
    @post __post.token._balances[address(this)] == token._balances[address(this)] -
        totalValue * (eomTimestampsAfterCliff.length - __post.
        earliestUnvestedMonthIndex) / (eomTimestampsAfterCliff.length + cliffDuration)
    @post __post.token._balances[beneficiary] == token._balances[beneficiary] +
        totalValue * (eomTimestampsAfterCliff.length - __post.
        earliestUnvestedMonthIndex) / (eomTimestampsAfterCliff.length + cliffDuration)
   */
  function release() public {
    require(now > endOfCliffTimestamp);
    uint256 earliestUnvestedMonthIndex = _getEarliestUnvestedMonthIndex();
    uint256 unvestedDuration = eomTimestampsAfterCliff.length.sub(
        earliestUnvestedMonthIndex);
    uint256 totalDuration = eomTimestampsAfterCliff.length.add(cliffDuration);
    uint256 unvestedToken = totalValue.mul(unvestedDuration).div(totalDuration);
    uint256 releasableToken = token.balanceOf(address(this)).sub(unvestedToken);
    require(releasableToken > 0 && token.transfer(beneficiary, releasableToken));
  }

  /*@CTK vesting_revoke
    @tag assume_completion
    @pre msg.sender == _owner
    @post __post.token._balances[msg.sender] == token._balances[msg.sender] + token.
        _balances[msg.sender]
   */
  function revoke() public onlyOwner {
    require(token.transfer(msg.sender, token.balanceOf(address(this))));
    selfdestruct(msg.sender);
  }

  /*@CTK vesting_getEarliestUnvestedMonthIndex
    @tag assume_completion
    @post __return <= eomTimestampsAfterCliff.length
   */
  function _getEarliestUnvestedMonthIndex() internal view returns (uint256) {
    /*@CTK loop_getEarliestUnvestedMonthIndex
      @inv i <= eomTimestampsAfterCliff.length
      @post i == eomTimestampsAfterCliff.length
      @post forall j: uint256. (j >= 0 /\ j < i) -> eomTimestampsAfterCliff[j] <= now
      @post (i < eomTimestampsAfterCliff.length) -> eomTimestampsAfterCliff[i] > now
      @post __should_return
     */
    for (uint256 i = 0; i < eomTimestampsAfterCliff.length; ++i) {
      if (now < eomTimestampsAfterCliff[i]) return i;
    }
    return eomTimestampsAfterCliff.length;
  }
}
```

File exchange/BondingCurve.sol

```
pragma solidity 0.5.9;

import "openzeppelin-solidity/contracts/math/SafeMath.sol";
import "../token/ERC20Acceptor.sol";
import "../token/ERC20Interface.sol";
import "../utils/Expression.sol";
import "../utils/Fractional.sol";
import "../Parameters.sol";


contract BondingCurve is ERC20Acceptor {
  using SafeMath for uint256;
  using Fractional for uint256;

  event Buy(address indexed buyer, uint256 bondedTokenAmount, uint256
      collateralTokenAmount);
  event Sell(address indexed seller, uint256 bondedTokenAmount, uint256
      collateralTokenAmount);
  event Deflate(address indexed burner, uint256 burnedAmount);
  event RevenueCollect(address indexed beneficiary, uint256 bondedTokenAmount);

  ERC20Interface public collateralToken;
  ERC20Interface public bondedToken;
  Parameters public params;

  uint256 public currentMintedTokens;
  uint256 public currentCollateral;
  uint256 public lastInflationTime = now;

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK BondingCurve
    @tag assume_completion
    @post __post.collateralToken == _collateralToken
    @post __post.bondedToken == _bondedToken
    @post __post.params == _params
   */
  constructor(ERC20Interface _collateralToken, ERC20Interface _bondedToken, Parameters
       _params) public {
    collateralToken = _collateralToken;
    bondedToken = _bondedToken;
    params = _params;
  }

  function getRevenueBeneficiary() public view returns (address) {
    address beneficiary = address(params.getRaw("bonding:revenue_beneficiary"));
    require(beneficiary != address(0));
    return beneficiary;
  }

  function getInflationRateNumerator() public view returns (uint256) {
    return params.getRaw("bonding:inflation_rate");
  }

  function getLiquiditySpreadNumerator() public view returns (uint256) {
    return params.getRaw("bonding:liquidity_spread");
  }

  function getCollateralExpression() public view returns (Expression) {
    return Expression(address(params.getRaw("bonding:curve_expression")));
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  function getCollateralAtSupply(uint256 tokenSupply) public view returns (uint256) {
    Expression collateralExpression = getCollateralExpression();
    uint256 collateralFromEquationAtCurrent = collateralExpression.evaluate(currentMintedTokens);
    uint256 collateralFromEquationAtSupply = collateralExpression.evaluate(tokenSupply);
    if (collateralFromEquationAtCurrent == 0) {
      return collateralFromEquationAtSupply;
    } else {
      return collateralFromEquationAtSupply.mul(currentCollateral).div(collateralFromEquationAtCurrent);
    }
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  function curveMultiplier() public view returns (uint256) {
    return currentCollateral.mul(Fractional.getDenominator()).div(getCollateralExpression().evaluate(currentMintedTokens));
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  function getBuyPrice(uint256 tokenValue) public view returns (uint256) {
    uint256 nextSupply = currentMintedTokens.add(tokenValue);
    return getCollateralAtSupply(nextSupply).sub(currentCollateral);
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK getSellPrice
    @tag assume_completion
    @post __return < currentCollateral
   */
  function getSellPrice(uint256 tokenValue) public view returns (uint256) {
    uint256 currentSupply = currentMintedTokens;
    require(currentSupply >= tokenValue);
    uint256 nextSupply = currentMintedTokens.sub(tokenValue);
    return currentCollateral.sub(getCollateralAtSupply(nextSupply));
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  modifier _adjustAutoInflation() {
    uint256 currentSupply = currentMintedTokens;
    if (lastInflationTime < now) {
      uint256 pastSeconds = now.sub(lastInflationTime);
      uint256 inflatingSupply = getInflationRateNumerator().mul(pastSeconds).mulFrac(
          currentSupply);
      if (inflatingSupply != 0) {
        currentMintedTokens = currentMintedTokens.add(inflatingSupply);
        _rewardBondingCurveOwner(inflatingSupply);
      }
    }
    lastInflationTime = now;
    _;
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK buy
    @tag assume_completion
    @post (msg.sender == collateralToken) || (msg.sender == buyer)
    @post __post.currentMintedTokens >= currentMintedTokens
    @post __post.currentCollateral >= currentCollateral
   */
  function buy(address buyer, uint256 priceLimit, uint256 buyAmount)
    public
    requireToken(collateralToken, buyer, priceLimit)
    _adjustAutoInflation
  {
    uint256 liquiditySpread = getLiquiditySpreadNumerator().mulFrac(buyAmount);
    uint256 totalMintAmount = buyAmount.add(liquiditySpread);
    uint256 buyPrice = getBuyPrice(totalMintAmount);
    require(buyPrice > 0 && buyPrice <= priceLimit);
    if (priceLimit > buyPrice) {
      require(collateralToken.transfer(buyer, priceLimit.sub(buyPrice)));
    }
    require(bondedToken.mint(buyer, buyAmount));
    if (liquiditySpread > 0) {
      _rewardBondingCurveOwner(liquiditySpread);
    }
    currentMintedTokens = currentMintedTokens.add(totalMintAmount);
    currentCollateral = currentCollateral.add(buyPrice);
    emit Buy(buyer, buyAmount, buyPrice);
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK sell
    @tag assume_completion
```

```
    @post (msg.sender == bondedToken) || (msg.sender == seller)
    @post __post.currentMintedTokens <= currentMintedTokens
    @post __post.currentCollateral <= currentCollateral
   */
  function sell(address seller, uint256 sellAmount, uint256 priceLimit)
    public
    requireToken(bondedToken, seller, sellAmount)
    _adjustAutoInflation
  {
    uint256 sellPrice = getSellPrice(sellAmount);
    require(sellPrice > 0 && sellPrice >= priceLimit);
    require(bondedToken.burn(address(this), sellAmount));
    require(collateralToken.transfer(seller, sellPrice));
    currentMintedTokens = currentMintedTokens.sub(sellAmount);
    currentCollateral = currentCollateral.sub(sellPrice);
    emit Sell(seller, sellAmount, sellPrice);
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK deflate
    @tag assume_completion
    @post __post.currentMintedTokens == currentMintedTokens - burnAmount
   */
  function deflate(address burner, uint256 burnAmount) public requireToken(bondedToken, burner, burnAmount) {
    require(bondedToken.burn(address(this), burnAmount));
    currentMintedTokens = currentMintedTokens.sub(burnAmount);
    emit Deflate(burner, burnAmount);
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  function _rewardBondingCurveOwner(uint256 rewardAmount) internal {
    address beneficiary = getRevenueBeneficiary();
    require(bondedToken.mint(beneficiary, rewardAmount));
    emit RevenueCollect(beneficiary, rewardAmount);
  }
}
```

File utils/Fractional.sol

```
pragma solidity 0.5.9;

import "openzeppelin-solidity/contracts/math/SafeMath.sol";


/// "Fractional" library facilitate fixed point decimal computation. In Band Protocol, fixed point decimal can be
/// represented using ‘uint256‘ data type. The decimal is fixed at 18 digits and ‘mulFrac‘ can be used to multiply
/// the fixed point decimal with an ordinary ‘uint256‘ value.
library Fractional {
  using SafeMath for uint256;
  uint256 internal constant DENOMINATOR = 1e18;

  /*@CTK getDenominator
```

```
    @tag assume_completion
    @post __return == 1000000000000000000
   */
  function getDenominator() internal pure returns (uint256) {
    return DENOMINATOR;
  }

  /*@CTK mulFrac
    @tag assume_completion
    @post __return == numerator * value / 1000000000000000000
   */
  function mulFrac(uint256 numerator, uint256 value) internal pure returns(uint256) {
    return numerator.mul(value).div(DENOMINATOR);
  }
}
```

File utils/Aggregator.sol

```
pragma solidity 0.5.9;

import "openzeppelin-solidity/contracts/math/Math.sol";


/// "Aggregator" interface contains one function, which describe how an array of unsigned integers should be processed
/// into a single unsigned integer result. The function will return ok = false if the aggregation fails.
interface Aggregator {
  function aggregate(uint256[] calldata data, uint256 size) external pure returns (uint256 result, bool ok);
}


/// "MedianAggregator" uses unweighted median as the aggregation method.
contract MedianAggregator is Aggregator {

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK MedianAggregator
    @tag assume_completion
    @post (size == 0) -> (result == 0) && (ok == false)
   */
  function aggregate(uint256[] calldata data, uint256 size) external pure returns (uint256 result, bool ok) {
    if (size == 0) return (0, false);
    uint256 middle = size / 2;
    uint256[] memory sData = new uint256[](middle + 2); // Only the first middle + 2 are needed
    /*@CTK loop_MedianAggregator
      @inv i <= size
      @post i == size
      @post !__should_return
     */
    for (uint256 i = 0; i < size; ++i) {
      uint256 loc = Math.min(i, middle + 1);
      sData[loc] = data[i];
      /*@CTK loop_loop_MedianAggregator
        @inv loc >= 0
        @post loc == 0
        @post !__should_return
       */
      for (; loc > 0; --loc) {
        if (sData[loc - 1] > sData[loc]) (sData[loc - 1], sData[loc]) = (sData[loc], sData[loc - 1]);
        else break;
      }
    }
    if (size % 2 == 0) {
      return (Math.average(sData[middle], sData[middle - 1]), true);
    } else {
      return (sData[middle], true);
    }
  }
}


/// "MajorityAggregator" uses majority (more than half of the same numbre) as the aggregation method.
contract MajorityAggregator is Aggregator {

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK MajorityAggregator
    @tag assume_completion
   */
  function aggregate(uint256[] calldata data, uint256 size) external pure returns (uint256 result, bool ok) {
    /*@CTK loop_MajorityAggregator
      @inv i >= 0
      @inv i <= size
      @post __should_return
     */
    for (uint256 i = 0; i < size; ++i) {
      uint256 count = 1;
      /*@CTK loop_loop_MajorityAggregator
        @inv j >= i + 1
        @inv j <= size
        @post (count > size / 2) || (j == size)
        @post __should_return
       */
      for (uint256 j = i + 1; j < size; ++j) {
        if (data[i] == data[j]) ++count;
      }
      if (count > size / 2) return (data[i], true);
    }
    return (0, false);
  }
}
```

File data/MultiSigTCD.sol

```
pragma solidity 0.5.9;

import "openzeppelin-solidity/contracts/math/SafeMath.sol";
import "../utils/Aggregator.sol";
import "./TCDBase.sol";
```

```


/// "MultiSigTCD" is a TCD that curates a list of trusted addresses. Data points from all reporters are aggregated
/// off-chain and reported using ‘report‘ function with ECDSA signatures. The contract verifies that all signatures
/// are valid and stores the aggregated value on its storage.
contract MultiSigTCD is TCDBase {
  using SafeMath for uint256;

  event DataPointUpdated(bytes key, uint256 value, QueryStatus status);

  struct DataPoint {
    uint256 value;
    uint64 timestamp;
    QueryStatus status;
  }

  mapping (bytes => DataPoint) private aggData;

  constructor(bytes8 _prefix, BondingCurve _bondingCurve, Parameters _params, BandRegistry _registry)
    public TCDBase(_prefix, _bondingCurve, _params, _registry) {}

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK MultiSig_queryPrice
    @tag assume_completion
   */
  function queryPrice() public view returns (uint256) {
    return params.get(prefix, "query_price");
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK report
    @tag assume_completion
   */
  function report(
    bytes calldata key,
    uint256[] calldata values,
    uint256[] calldata timestamps,
    uint8[] calldata v,
    bytes32[] calldata r,
    bytes32[] calldata s
  ) external {
    require(values.length.mul(3) > activeCount.mul(2));
    require(values.length == timestamps.length);
    address lastSigner = address(0);
    /*@CTK loop_report
      @inv timestamps[i] > aggData[key].timestamp
      @inv i <= values.length
      @post i == values.length
      @post !__should_return
     */
    for (uint256 i = 0; i < values.length; ++i) {
      require(timestamps[i] > aggData[key].timestamp);
      address recovered = ecrecover(keccak256(abi.encodePacked(
        "\x19Ethereum Signed Message:\n32",
        keccak256(abi.encodePacked(key, values[i], timestamps[i], address(this))))),
        v[i], r[i], s[i]
      );
      require(activeList[recovered] != NOT_FOUND);
      require(recovered > lastSigner);
      lastSigner = recovered;
    }
    _save(key, values);
  }

  function _save(bytes memory key, uint256[] memory values) private {
    Aggregator agg = Aggregator(address(params.get(prefix, "data_aggregator")));
    (uint256 result, bool ok) = agg.aggregate(values, values.length);
    QueryStatus status = ok ? QueryStatus.OK : QueryStatus.DISAGREEMENT;
    aggData[key] = DataPoint({
      value: result,
      timestamp: uint64(now),
      status: status
    });
    emit DataPointUpdated(key, result, status);
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK queryImpl
    @tag assume_completion
    @post (aggData[input].timestamp == 0) -> (status == QueryStatus.NOT_AVAILABLE)
    @post (aggData[input].timestamp != 0) && (aggData[input].status != QueryStatus.OK) -> (status == aggData[input].status)
    @post (aggData[input].timestamp != 0) && (aggData[input].status == QueryStatus.OK) -> (status == QueryStatus.OK)
   */
  function queryImpl(bytes memory input) internal returns (bytes32 output, uint256 updatedAt, QueryStatus status) {
    DataPoint storage data = aggData[input];
    if (data.timestamp == 0) return ("", 0, QueryStatus.NOT_AVAILABLE);
    if (data.status != QueryStatus.OK) return ("", data.timestamp, data.status);
    return (bytes32(data.value), data.timestamp, QueryStatus.OK);
  }
}
```

File data/TCRBase.sol

```
pragma solidity 0.5.9;

import "openzeppelin-solidity/contracts/math/SafeMath.sol";
import "openzeppelin-solidity/contracts/math/Math.sol";
import "../token/ERC20Acceptor.sol";
import "../utils/Expression.sol";
import "../utils/Fractional.sol";
import "../Parameters.sol";


contract TCRBase is ERC20Acceptor {
  using Fractional for uint256;
using SafeMath for uint256;

event ApplicationSubmitted(bytes32 data, address indexed proposer, uint256 listAt, uint256 deposit);
event EntryDeposited(bytes32 indexed data, uint256 value);
event EntryWithdrawn(bytes32 indexed data,uint256 value);
event EntryExited(bytes32 indexed data);
event ChallengeInitiated(bytes32 indexed data, uint256 indexed challengeId, address indexed challenger, uint256 stake, bytes32 reasonData, uint256 proposerVote, uint256 challengerVote);
event ChallengeVoteCommitted(uint256 indexed challengeId,address indexed voter, bytes32 commitValue, uint256 weight);
event ChallengeVoteRevealed(uint256 indexed challengeId,address indexed voter, bool voteKeep);
event ChallengeSuccess(bytes32 indexed data, uint256 indexed challengeId, uint256 voterRewardPool, uint256 challengerReward);
event ChallengeFailed(bytes32 indexed data, uint256 indexed challengeId, uint256 voterRewardPool, uint256 proposerReward);
event ChallengeInconclusive(bytes32 indexed data, uint256 indexed challengeId);
event ChallengeRewardClaimed(uint256 indexed challengeId, address indexed voter, uint256 reward);

Parameters public params;
SnapshotToken public token;
bytes8 public prefix;

/// A TCR entry is considered to exist in ’entries’ map iff its ’listedAt’ is nonzero.
struct Entry {
  address proposer;     /// The entry proposer
  uint256 deposit;      /// Amount token that is not on challenge stake
  uint256 listedAt;     /// Expiration time of entry’s ’pending’ status
  uint256 challengeId;  /// Id of challenge, applicable if not zero
}
enum ChallengeState { Invalid, Open, Kept, Removed, Inconclusive }
enum VoteStatus { Nothing, Committed, VoteKeep, VoteRemove, Claimed }

/// A challenge represent a challenge for a TCR entry.
struct Challenge {
  bytes32 entryData;         /// The hash of data that is in question
  bytes32 reasonData;        /// The hash of reason for this challenge
  address challenger;        /// The challenger
  uint256 rewardPool;        /// Remaining reward pool. Relevant after resolved.
  uint256 remainingRewardVotes; /// Remaining voting power to claim rewards.
  uint256 commitEndTime;
  uint256 revealEndTime;
  uint256 snapshotNonce;
  uint256 voteRemoveRequiredPct;
  uint256 voteMinParticipation;
  uint256 keepCount;
  uint256 removeCount;
  uint256 totalCommitCount;
  mapping (address => bytes32) voteCommits;
  mapping (address => VoteStatus) voteStatuses;
  ChallengeState state;
}

mapping (bytes32 => Entry) public entries;
mapping (uint256 => Challenge) public challenges;
uint256 nextChallengeNonce = 1;

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK TCRBase
  @tag assume_completion
  @post __post.params == _params
  @post __post.prefix == _prefix
 */
constructor(bytes8 _prefix, Parameters _params) public {
  params = _params;
  prefix = _prefix;
  token = _params.token();
}

modifier entryMustExist(bytes32 data) {
  require(entries[data].listedAt > 0);
  _;
}

modifier entryMustNotExist(bytes32 data) {
  require(entries[data].listedAt == 0);
  _;
}

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK isEntryActive
  @tag assume_completion
  @post __return == (entries[data].listedAt != 0) && (now >= entries[data].listedAt)
 */
function isEntryActive(bytes32 data) public view returns (bool) {
  uint256 listedAt = entries[data].listedAt;
  return listedAt != 0 && now >= listedAt;
}

/*@CTK getVoteStatus
  @tag assume_completion
  @post __return == challenges[challengeId].voteStatuses[voter]
 */
function getVoteStatus(uint256 challengeId, address voter) public view returns (
    VoteStatus) {
  return challenges[challengeId].voteStatuses[voter];
}

/*@CTK currentMinDeposit_before_depreciation_cliff
  @tag assume_completion
  @pre now < entries[entryData].listedAt
  @post __return == minDeposit
 */
/*@CTK currentMinDeposit_after_depreciation_cliff
  @tag assume_completion
  @pre now >= entries[entryData].listedAt
  @post __return < minDeposit
 */
```

```
function currentMinDeposit(bytes32 entryData) public view entryMustExist(entryData) returns (uint256) {
  Entry storage entry = entries[entryData];
  uint256 minDeposit = params.get(prefix, "min_deposit");
  if (now < entry.listedAt) {
    return minDeposit;
  } else {
    address depositDecayFunction = address(params.get(prefix, "deposit_decay_function"));
    if (depositDecayFunction == address(0)) return minDeposit;
    else return Expression(depositDecayFunction).evaluate(now.sub(entry.listedAt)).mulFrac(minDeposit);
  }
}

/// Apply a new entry to the TCR. The applicant must stake token at least ‘min_deposit‘.
/// Application will get auto-approved if no challenge happens in ‘apply_stage_length‘ seconds.
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK applyEntry
  @tag assume_completion
  @pre (entries[data].listedAt == 0)
  @pre (msg.sender == token) || (msg.sender == proposer)
  @post __post.entries[data].proposer == proposer
  @post __post.entries[data].deposit == deposit
  @post __post.entries[data].listedAt >= now
 */
function applyEntry(address proposer, uint256 stake, bytes32 data)
  public
  requireToken(token, proposer, stake)
  entryMustNotExist(data)
{
  require(stake >= params.get(prefix, "min_deposit"));
  Entry storage entry = entries[data];
  entry.proposer = proposer;
  entry.deposit = stake;
  entry.listedAt = now.add(params.get(prefix, "apply_stage_length"));
  emit ApplicationSubmitted(data, proposer, entry.listedAt, stake);
}

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK deposit
  @tag assume_completion
  @pre (entries[data].listedAt > 0)
  @pre (msg.sender == token) || (msg.sender == depositor)
  @post (entries[data].proposer == depositor)
  @post __post.entries[data].deposit == entries[data].deposit + amount
 */
function deposit(address depositor, uint256 amount, bytes32 data)
  public
  requireToken(token, depositor, amount)
  entryMustExist(data)
{
  Entry storage entry = entries[data];
  require(entry.proposer == depositor);
  entry.deposit = entry.deposit.add(amount);
  emit EntryDeposited(data, amount);
}

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK withdraw_proposer
  @tag assume_completion
  @pre (entries[data].listedAt > 0)
  @pre entries[data].proposer == msg.sender
  @post entry.deposit > amount
  @post __post.entries[data].deposit == entries[data].deposit - amount
  @post __post.token._balances[address(token)] == token._balances[address(token)] - amount
  @post __post.token._balances[msg.sender] == token._balances[msg.sender] + amount
 */
/*@CTK withdraw_others
  @tag assume_completion
  @pre (entries[data].listedAt > 0)
  @pre (entries[data].proposer != msg.sender)
  @post (entry.deposit >= amount)
  @post __post.entries[data].deposit == entries[data].deposit - amount
  @post __post.token._balances[address(token)] == token._balances[address(token)] - amount
  @post __post.token._balances[msg.sender] == token._balances[msg.sender] + amount
 */
function withdraw(bytes32 data, uint256 amount)
  public
  entryMustExist(data)
{
  Entry storage entry = entries[data];
  require(entry.proposer == msg.sender);
  if (entry.challengeId == 0) {
    require(entry.deposit >= amount.add(currentMinDeposit(data)));
  } else {
    require(entry.deposit >= amount);
  }
  entry.deposit = entry.deposit.sub(amount);
  require(token.transfer(msg.sender, amount));
  emit EntryWithdrawn(data, amount);
}

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK exit
  @tag assume_completion
  @pre (entries[data].listedAt > 0)
  @pre (entries[data].proposer == msg.sender)
  @pre (entries[data].challengeId == 0)
  @post __post.token._balances[address(token)] == token._balances[address(token)] - entries[data].deposit
  @post __post.token._balances[entries[data].proposer] == token._balances[entries[data].proposer] + entries[data].deposit
  @post __post.entries[data].proposer == 0
```

```
 */
function exit(bytes32 data) public entryMustExist(data) {
  Entry storage entry = entries[data];
  require(entry.proposer == msg.sender);
  require(entry.challengeId == 0);
  _deleteEntry(data);
  emit EntryExited(data);
}

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK initiateChallenge
  @tag assume_completion
  @post (entries[data].listedAt > 0)
  @post (msg.sender == token) || (msg.sender == depositor)
  @post (entries[data].challengeId == 0) && (entries[data].proposer != challenger)
  @post (now <= entries[entryData].listedAt) -> (challengeDeposit >= entries[data].deposit)
  @post __post.entries[data].deposit == entries[data].deposit - stake
  @post __post.entries[data].challengeId == challengeId
  @post __post.challenges[challengeId].voteStatuses[entries[data].proposer] == VoteStatus.VoteKeep
  @post __post.challenges[challengeId].voteStatuses[challenger] == VoteStatus.VoteRemove
 */
function initiateChallenge(address challenger, uint256 challengeDeposit, bytes32 data, bytes32 reasonData)
  public
  requireToken(token, challenger, challengeDeposit)
  entryMustExist(data)
{
  Entry storage entry = entries[data];
  require(entry.challengeId == 0 && entry.proposer != challenger);
  uint256 stake = Math.min(entry.deposit, currentMinDeposit(data));
  require(challengeDeposit >= stake);
  if (challengeDeposit != stake) {
    require(token.transfer(challenger, challengeDeposit.sub(stake)));
  }
  entry.deposit = entry.deposit.sub(stake);
  uint256 challengeId = nextChallengeNonce;
  uint256 proposerVote = token.historicalVotingPowerAtNonce(entry.proposer, token.votingPowerChangeNonce());
  uint256 challengerVote = token.historicalVotingPowerAtNonce(challenger, token.votingPowerChangeNonce());
  nextChallengeNonce = challengeId.add(1);
  challenges[challengeId] = Challenge({
    entryData: data,
    reasonData: reasonData,
    challenger: challenger,
    rewardPool: stake,
    remainingRewardVotes: 0,
    commitEndTime: now.add(params.get(prefix, "commit_time")),
    revealEndTime: now.add(params.get(prefix, "commit_time")).add(params.get(prefix, "reveal_time")),
    snapshotNonce: token.votingPowerChangeNonce(),
    voteRemoveRequiredPct: params.get(prefix, "support_required_pct"),
    voteMinParticipation: params.get(prefix, "min_participation_pct").mulFrac(token.
        totalSupply()),
    keepCount: proposerVote,
    removeCount: challengerVote,
    totalCommitCount: proposerVote.add(challengerVote),
    state: ChallengeState.Open
  });
  entry.challengeId = challengeId;
  challenges[challengeId].voteStatuses[entry.proposer] = VoteStatus.VoteKeep;
  challenges[challengeId].voteStatuses[challenger] = VoteStatus.VoteRemove;
  emit ChallengeInitiated(data, challengeId, challenger, stake, reasonData, proposerVote, challengerVote);
}

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK commitVote
  @tag assume_completion
  @post (challenges[challengeId].state == ChallengeState.Open) && (now < challenges[challengeId].commitEndTime)
  @post (challenges[challengeId].state == ChallengeState.Open) && (now < challenges[challengeId].commitEndTime)
  @post __post.challenges[challengeId].voteCommits[msg.sender] == commitValue
  @post __post.challenges[challengeId].voteStatuses[msg.sender] == VoteStatus.Committed
  @post __post.challenges[challengeId].totalCommitCount > challenges[challengeId].totalCommitCount
 */
function commitVote(uint256 challengeId, bytes32 commitValue) public {
  Challenge storage challenge = challenges[challengeId];
  require(challenge.state == ChallengeState.Open && now < challenge.commitEndTime);
  require(challenge.voteStatuses[msg.sender] == VoteStatus.Nothing);
  challenge.voteCommits[msg.sender] = commitValue;
  challenge.voteStatuses[msg.sender] = VoteStatus.Committed;
  uint256 weight = token.historicalVotingPowerAtNonce(msg.sender, challenge.snapshotNonce);
  challenge.totalCommitCount = challenge.totalCommitCount.add(weight);
  emit ChallengeVoteCommitted(challengeId, msg.sender, commitValue, weight);
}

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK revealVote
  @tag assume_completion
  @post (challenges[challengeId].state == ChallengeState.Open)
  @post (now >= challenges[challengeId].commitEndTime && now < challenges[challengeId].revealEndTime)
  @post (challenges[challengeId].voteStatuses[voter] == VoteStatus.Committed)
  @post (challenges[challengeId].voteCommits[voter] == keccak256(abi.encodePacked(voteKeep, salt)))
  @post (voteKeep) -> (__post.challenges[challengeId].keepCount > challenges[challengeId].keepCount) && (__post.challenges[challengeId].voteStatuses[voter] == VoteStatus.VoteKeep)
  @post !(voteKeep) -> (__post.challenges[challengeId].removeCount > challenges[challengeId].removeCount) && (__post.challenges[challengeId].voteStatuses[voter] == VoteStatus.VoteRemove)
 */
```

```
function revealVote(address voter, uint256 challengeId, bool voteKeep, uint256 salt) public {
  Challenge storage challenge = challenges[challengeId];
  require(challenge.state == ChallengeState.Open);
  require(now >= challenge.commitEndTime && now < challenge.revealEndTime);
  require(challenge.voteStatuses[voter] == VoteStatus.Committed);
  require(challenge.voteCommits[voter] == keccak256(abi.encodePacked(voteKeep, salt)));
  uint256 weight = token.historicalVotingPowerAtNonce(voter, challenge.snapshotNonce);
  if (voteKeep) {
    challenge.keepCount = challenge.keepCount.add(weight);
    challenge.voteStatuses[voter] = VoteStatus.VoteKeep;
  } else {
    challenge.removeCount = challenge.removeCount.add(weight);
    challenge.voteStatuses[voter] = VoteStatus.VoteRemove;
  }
  emit ChallengeVoteRevealed(challengeId, voter, voteKeep);
}

/// Resolve TCR challenge. If the challenge succeeds, the entry will be removed and the challenger
/// gets the reward. Otherwise, the entry's ‘deposit‘ gets bumped by the reward.
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK resolveChallenge_kept
  @tag assume_completion
  @pre (challenges[challengeId] == ChallengeState.Open)
  @pre (entries[challenges[challengeId].entryData].challengeId == challengeId)
  @pre __post.entries[challenges[challengeId].entryData].challengeId == 0
  @pre result == ChallengeState.Kept
  @post __post.entries[challenges[challengeId].entryData].deposit == entries[challenges[challengeId].entryData].deposit + __post.winnerTotalReward
  @post __post.challenges[challengeId].rewardPool == challenges[challengeId].rewardPool - __post.proposerVoteReward
  @post (__post.entries[challenges[challengeId].entryData].deposit - entries[challenges[challengeId].entryData].deposit) == (challenges[challengeId].rewardPool - __post.challenges[challengeId].rewardPool) + challenges[challengeId].rewardPool
  @post __post.challenges[challengeId].remainingRewardVotes == challenges[challengeId].keepCount - __post.proposerVote
  @post __post.challenges[challengeId].voteStatuses[entries[challenges[challengeId].entryData].proposer] == VoteStatus.Claimed
 */
/*@CTK resolveChallenge_remove
  @tag assume_completion
  @pre (challenges[challengeId] == ChallengeState.Open)
  @pre (entries[challenges[challengeId].entryData].challengeId == challengeId)
  @pre __post.entries[challenges[challengeId].entryData].challengeId == 0
  @pre result == ChallengeState.Removed
  @post __post.tokens._balances[address(token)] == tokens._balances[address(token)] - __post.winnerTotalReward
  @post __post.tokens._balances[challenges[challengeId].challenger] == tokens._balances[challenges[challengeId].challenger] + __post.winnerTotalReward
  @post __post.challenges[challengeId].rewardPool == __post.rewardPool - __post.challengerVoteReward
  @post (__post.tokens._balances[challenges[challengeId].challenger] - tokens.
      _balances[challenges[challengeId].challenger]) == (challenges[challengeId].
      rewardPool - __post.challenges[challengeId].rewardPool) + challenges[
      challengeId].rewardPool
  @post __post.challenges[challengeId].remainingRewardVotes == challenges[
      challengeId].removeCount - __post.challengerVote
  @post __post.entries[challenges[challengeId].entryData].proposer == 0
  @post __post.challenges[challengeId].voteStatuses[challenges[challengeId].
      challenger] == VoteStatus.Claimed
 */
/*@CTK resolveChallenge_inconclusive
  @tag assume_completion
  @pre (challenges[challengeId] == ChallengeState.Open)
  @pre (entries[challenges[challengeId].entryData].challengeId == challengeId)
  @pre __post.entries[challenges[challengeId].entryData].challengeId == 0
  @pre (result == ChallengeState.Inconclusive)
  @post __post.entries[challenges[challengeId].entryData].deposit == entries[
      challenges[challengeId].entryData].deposit + challenges[challengeId].
      rewardPool
  @post __post.tokens._balances[address(token)] == tokens._balances[address(token)]
      - challenges[challengeId].rewardPool
  @post __post.tokens._balances[challenges[challengeId].challenger] == tokens.
      _balances[challenges[challengeId].challenger] + challenges[challengeId].
      rewardPool
  @post __post.challenges[challengeId].rewardPool == 0
 */
/*@CTK resolveChallenge_invalid
  @tag assume_completion
  @pre (challenges[challengeId] == ChallengeState.Open)
  @pre (entries[challenges[challengeId].entryData].challengeId == challengeId)
  @pre __post.entries[challenges[challengeId].entryData].challengeId == 0
  @pre (result != ChallengeState.Kept) && (result != ChallengeState.Removed) && (
      result != ChallengeState.Inconclusive)
  @post __reverted
 */
function resolveChallenge(uint256 challengeId) public {
  Challenge storage challenge = challenges[challengeId];
  require(challenge.state == ChallengeState.Open);
  ChallengeState result = _getChallengeResult(challenge);
  challenge.state = result;
  bytes32 data = challenge.entryData;
  Entry storage entry = entries[data];
  assert(entry.challengeId == challengeId);
  entry.challengeId = 0;
  uint256 challengerStake = challenge.rewardPool;
  uint256 winnerExtraReward = params.get(prefix, "dispensation_percentage").mulFrac(
      challengerStake);
  uint256 winnerTotalReward = challengerStake.add(winnerExtraReward);
  uint256 rewardPool = challengerStake.sub(winnerExtraReward);
  if (result == ChallengeState.Kept) {
    uint256 proposerVote = token.historicalVotingPowerAtNonce(entry.proposer,
        challenge.snapshotNonce);
    uint256 proposerVoteReward = rewardPool.mul(proposerVote).div(challenge.
        keepCount);
    winnerTotalReward = winnerTotalReward.add(proposerVoteReward);
    entry.deposit = entry.deposit.add(winnerTotalReward);
    challenge.rewardPool = rewardPool.sub(proposerVoteReward);
    challenge.remainingRewardVotes = challenge.keepCount.sub(proposerVote);
    challenge.voteStatuses[entry.proposer] = VoteStatus.Claimed;
```

```
      emit ChallengeFailed(data, challengeId, challenge.rewardPool, winnerTotalReward);
    } else if (result == ChallengeState.Removed) {
      uint256 challengerVote = token.historicalVotingPowerAtNonce(challenge.challenger, challenge.snapshotNonce);
      uint256 challengerVoteReward = rewardPool.mul(challengerVote).div(challenge.removeCount);
      winnerTotalReward = winnerTotalReward.add(challengerVoteReward);
      require(token.transfer(challenge.challenger, winnerTotalReward));
      challenge.rewardPool = rewardPool.sub(challengerVoteReward);
      challenge.remainingRewardVotes = challenge.removeCount.sub(challengerVote);
      _deleteEntry(data);
      challenge.voteStatuses[challenge.challenger] = VoteStatus.Claimed;
      emit ChallengeSuccess(data, challengeId, challenge.rewardPool, winnerTotalReward);
    } else if (result == ChallengeState.Inconclusive) {
      entry.deposit = entry.deposit.add(challengerStake);
      require(token.transfer(challenge.challenger, challengerStake));
      challenge.rewardPool = 0;
      emit ChallengeInconclusive(data, challengeId);
    } else {
      assert(false);
    }
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK claimReward
    @tag assume_completion
    @post (challenges[challengeId].remainingRewardVotes > 0)
    @post ((challenges[challengeId].state == ChallengeState.Kept) && (challenges[challengeId].voteStatuses[voter] == VoteStatus.VoteKeep)) || ((challenges[challengeId].state == ChallengeState.Removed) && (challenges[challengeId].voteStatuses[voter] == VoteStatus.VoteRemove))
    @post __post.challenges[challengeId].voteStatuses[voter] == VoteStatus.Claimed
    @post __post.challenges[challengeId].remainingRewardVotes <= challenges[challengeId].rewardPool
    @post __post.challenges[challengeId].rewardPool <= challenges[challengeId].rewardPool
    @post (token._balances[address(token)] - __post.token._balances[address(token)]) == (__post.token._balances[voter] - token._balances[voter])
   */
  function claimReward(address voter, uint256 challengeId) public {
    Challenge storage challenge = challenges[challengeId];
    require(challenge.remainingRewardVotes > 0);
    if (challenge.state == ChallengeState.Kept) {
      require(challenge.voteStatuses[voter] == VoteStatus.VoteKeep);
    } else if (challenge.state == ChallengeState.Removed) {
      require(challenge.voteStatuses[voter] == VoteStatus.VoteRemove);
    } else {
      revert();
    }
    challenge.voteStatuses[voter] = VoteStatus.Claimed;
    uint256 weight = token.historicalVotingPowerAtNonce(voter, challenge.snapshotNonce);
    if (weight > 0) {
      uint256 remainingRewardPool = challenge.rewardPool;
      uint256 remainingRewardVotes = challenge.remainingRewardVotes;
      uint256 reward = remainingRewardPool.mul(weight).div(remainingRewardVotes);
      challenge.remainingRewardVotes = remainingRewardVotes.sub(weight);
      challenge.rewardPool = remainingRewardPool.sub(reward);
      require(token.transfer(voter, reward));
      emit ChallengeRewardClaimed(challengeId, voter, reward);
    }
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK _getChallengeResult_inconclusive
    @tag assume_completion
    @pre challenge.state == ChallengeState.Open
    @pre now >= challenge.commitEndTime
    @pre (challenge.totalCommitCount < challenge.voteMinParticipation) || ((challenge.keepCount == 0) && (challenge.removeCount == 0))
    @post __return ChallengeState.Inconclusive
   */
  /*@CTK _getChallengeResult_conclusive
    @tag assume_completion
    @pre challenge.state == ChallengeState.Open
    @pre now >= challenge.commitEndTime
    @pre (challenge.totalCommitCount >= challenge.voteMinParticipation) && ((challenge.keepCount != 0) || (challenge.removeCount != 0))
    @post (challenge.removeCount * 1000000000000000000 >= challenge.voteRemoveRequiredPct * (challenge.keepCount + challenge.keepCount.removeCount)) -> __return == ChallengeState.Removed
    @post (challenge.removeCount * 1000000000000000000 < challenge.voteRemoveRequiredPct * (challenge.keepCount + challenge.keepCount.removeCount)) -> __return == ChallengeState.Kept
   */
  function _getChallengeResult(Challenge storage challenge) internal view returns (ChallengeState) {
    assert(challenge.state == ChallengeState.Open);
    require(now >= challenge.commitEndTime);
    if (challenge.totalCommitCount < challenge.voteMinParticipation) {
      return ChallengeState.Inconclusive;
    }
    uint256 keepCount = challenge.keepCount;
    uint256 removeCount = challenge.removeCount;
    if (keepCount == 0 && removeCount == 0) {
      return ChallengeState.Inconclusive;
    }
    if (removeCount.mul(Fractional.getDenominator()) >= challenge.voteRemoveRequiredPct.mul(keepCount.add(removeCount))) {
      return ChallengeState.Removed;
    } else {
      return ChallengeState.Kept;
    }
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK _deleteEntry_has_deposit
    @tag assume_completion
```

```
    @post __post.token._balances[address(token)] == token._balances[address(token)] - entries[data].deposit
    @post __post.token._balances[entries[data].proposer] == token._balances[entries[data].proposer] + entries[data].deposit
    @post __post.entries[data].proposer == 0
   */
  function _deleteEntry(bytes32 data) internal {
    uint256 entryDeposit = entries[data].deposit;
    address proposer = entries[data].proposer;
    if (entryDeposit > 0) {
      require(token.transfer(proposer, entryDeposit));
    }
    delete entries[data];
  }
}
```

File data/AggTCD.sol

```
pragma solidity 0.5.9;

import "openzeppelin-solidity/contracts/math/SafeMath.sol";
import "../utils/Aggregator.sol";
import "./TCDBase.sol";


/// "AggTCD" is a TCD that curates a list of smart contract addresses. Each smart contract must implement ‘get(bytes)‘
/// function that returns a value given a key. Data points are aggregated using the aggregator smart contract as
/// specified using key ‘{prefix}:data_aggregator‘.
contract AggTCD is TCDBase {
  using SafeMath for uint256;

  constructor(bytes8 _prefix, BondingCurve _bondingCurve, Parameters _params, BandRegistry _registry)
    public TCDBase(_prefix, _bondingCurve, _params, _registry) {}

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  function queryPrice() public view returns (uint256) {
    return params.get(prefix, "query_price");
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK AggTCD_queryImpl
    @tag assume_completion
   */
  function queryImpl(bytes memory input) internal returns (bytes32 output, uint256 updatedAt, QueryStatus status) {
    uint256[] memory data = new uint256[](activeCount);
    uint256 size = 0;
    address dataSourceAddress = activeList[ACTIVE_GUARD];
    /*@CTK loop_AggTCD
      @post dataSourceAddress == ACTIVE_GUARD
      @post !__should_return
     */
    while (dataSourceAddress != ACTIVE_GUARD) {
      (bool ok, bytes memory ret) = dataSourceAddress.call(abi.encodeWithSignature("get(bytes)", input));
      if (ok && ret.length == 32) {
        uint256 value = abi.decode(ret, (uint256));
        data[size++] = value;
      }
      dataSourceAddress = activeList[dataSourceAddress];
    }
    if (size == 0 || size.mul(3) < activeCount.mul(2)) return ("", 0, QueryStatus.NOT_AVAILABLE);
    Aggregator agg = Aggregator(address(params.get(prefix, "data_aggregator")));
    (uint256 result, bool ok) = agg.aggregate(data, size);
    if (!ok) return ("", now, QueryStatus.DISAGREEMENT);
    else return (bytes32(result), now, QueryStatus.OK);
  }
}
```

File data/TCDBase.sol

```
pragma solidity 0.5.9;

import "openzeppelin-solidity/contracts/math/SafeMath.sol";
import "./QueryInterface.sol";
import "../utils/Fractional.sol";
import "../exchange/BondingCurve.sol";
import "../token/LockableToken.sol";
import "../Parameters.sol";


/// "TCDBase" is the base class for Band Protocol’s Token-Curated DataSources implementation. The contract essentially
/// keeps track of a sorted list of trusted data sources, based on the total amount of token stake the data sources
/// have. Any one can apply for a new data source using ‘register‘ function. Token holders can ‘stake‘ or ‘unstake‘
/// for any existing data sources. This class is abstract, so it needs to be extended by a subclass that utilizes
/// the list of active data sources (See AggTCD and MultiSigTCD). Fees are collected in ETH and are converted to
/// dataset tokens during ‘distributeFee‘ function call.
contract TCDBase is QueryInterface {
  using Fractional for uint256;
  using SafeMath for uint256;

  event DataSourceRegistered(address indexed dataSource, address indexed owner, uint256 stake);
  event DataSourceStaked(address indexed dataSource, address indexed participant, uint256 stake);
  event DataSourceUnstaked(address indexed dataSource, address indexed participant, uint256 unstake);
  event FeeDistributed(address indexed dataSource, uint256 totalReward, uint256 ownerReward);
  event WithdrawReceiptCreated(uint256 receiptIndex, address indexed owner, uint256 amount, uint64 withdrawTime);
  event WithdrawReceiptUnlocked(uint256 receiptIndex, address indexed owner, uint256 amount);

  enum Order {EQ, LT, GT}
```

```

struct DataSourceInfo {
  address owner;
  uint256 stake;
  uint256 totalOwnerships;
  mapping (address => uint256) tokenLocks;
  mapping (address => uint256) ownerships;
}

struct WithdrawReceipt {
  address owner;
  uint256 amount;
  uint64 withdrawTime;
  bool isWithdrawn;
}

mapping (address => DataSourceInfo) public infoMap;
mapping (address => address) private activeList;
mapping (address => address) private reserveList;
uint256 public activeCount;
uint256 public reserveCount;

address constant internal NOT_FOUND = address(0x00);
address constant internal ACTIVE_GUARD = address(0x01);
address constant internal RESERVE_GUARD = address(0x02);
WithdrawReceipt[] public withdrawReceipts;

BondingCurve public bondingCurve;
Parameters public params;
LockableToken public token;
uint256 public undistributedReward;
bytes8 public prefix;

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK TCDBase
  @tag assume_completion
  @post __post.bondingCurve == _bondingCurve
  @post __post.params == _params
  @post __post.prefix == _prefix
  @post __post.activeList[ACTIVE_GUARD] == ACTIVE_GUARD
  @post __post.reserveList[RESERVE_GUARD] == RESERVE_GUARD
 */
constructor(bytes8 _prefix, BondingCurve _bondingCurve, Parameters _params,
    BandRegistry _registry) public QueryInterface(_registry) {
  bondingCurve = _bondingCurve;
  params = _params;
  prefix = _prefix;
  token = LockableToken(address(_bondingCurve.bondedToken()));
  _registry.band().approve(address(_bondingCurve), 2 ** 256 - 1);
  activeList[ACTIVE_GUARD] = ACTIVE_GUARD;
  reserveList[RESERVE_GUARD] = RESERVE_GUARD;
}

/*@CTK getOwnership
  @tag assume_completion
  @post __return == infoMap[dataSource].ownerships[staker]
 */
function getOwnership(address dataSource, address staker) public view returns (uint256) {
  return infoMap[dataSource].ownerships[staker];
}

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK getStake_none
  @tag assume_completion
  @pre infoMap[dataSource].totalOwnerships == 0
  @post __return == 0
 */
/*@CTK getStake
  @tag assume_completion
  @pre infoMap[dataSource].totalOwnerships > 0
  @post __return == infoMap[dataSource].ownerships[staker] * infoMap[dataSource].stake / infoMap[dataSource].totalOwnerships
 */
function getStake(address dataSource, address staker) public view returns (uint256) {
  DataSourceInfo storage provider = infoMap[dataSource];
  if (provider.totalOwnerships == 0) return 0;
  return provider.ownerships[staker].mul(provider.stake).div(provider.totalOwnerships);
}

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK register
  @tag assume_completion
  @pre infoMap[dataSource].totalOwnerships == 0
  @pre initialStake > 0
  @post __post.infoMap[dataSource].owner == msg.sender
  @post __post.infoMap[dataSource].stake == initialStake
  @post __post.infoMap[dataSource].totalOwnerships == initialStake
  @post __post.infoMap[dataSource].ownerships[msg.sender] == initialStake
  @post __post.infoMap[dataSource].tokenLocks[msg.sender] == initialStake
 */
function register(address dataSource, address prevDataSource, uint256 initialStake) public {
  require(token.lock(msg.sender, initialStake));
  require(infoMap[dataSource].totalOwnerships == 0);
  require(initialStake > 0 && initialStake >= params.get(prefix, "min_provider_stake"));
  infoMap[dataSource] = DataSourceInfo({
    owner: msg.sender,
    stake: initialStake,
    totalOwnerships: initialStake
  });
  infoMap[dataSource].ownerships[msg.sender] = initialStake;
  infoMap[dataSource].tokenLocks[msg.sender] = initialStake;
  emit DataSourceRegistered(dataSource, msg.sender, initialStake);
  _addDataSource(dataSource, prevDataSource);
  _rebalanceLists();
}
```

```

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK stake
  @tag assume_completion
  @post __post.infoMap[dataSource].tokenLocks[msg.sender] == (infoMap[dataSource].tokenLocks[msg.sender] + value)
 */
function stake(address dataSource, address prevDataSource, address newPrevDataSource, uint256 value) public {
  require(token.lock(msg.sender, value));
  _removeDataSource(dataSource, prevDataSource);
  DataSourceInfo storage provider = infoMap[dataSource];
  uint256 newStakerTokenLock = provider.tokenLocks[msg.sender].add(value);
  provider.tokenLocks[msg.sender] = newStakerTokenLock;
  _stake(msg.sender, value, dataSource);
  if (getStake(dataSource, provider.owner) >= params.get(prefix, "min_provider_stake")) {
    _addDataSource(dataSource, newPrevDataSource);
  }
  _rebalanceLists();
}

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK unstake
  @tag assume_completion
  @post withdrawOwnership <= infoMap[dataSource].ownerships[msg.sender]
  @post __post.infoMap[dataSource].stake == infoMap[dataSource].stake - (infoMap[dataSource].stake * withdrawOwnership) / (infoMap[dataSource].stake.totalOwnerships)
  @post __post.infoMap[dataSource].totalOwnerships == infoMap[dataSource].totalOwnerships - withdrawOwnership
  @post __post.infoMap[dataSource].ownerships[msg.sender] == infoMap[dataSource].ownerships[msg.sender] - withdrawOwnership
  @post __post.infoMap[dataSource].tokenLocks[msg.sender] == infoMap[dataSource].ownerships[msg.sender] * infoMap[dataSource].stake / infoMap[dataSource].totalOwnerships - withdrawOwnership
 */
function unstake(address dataSource, address prevDataSource, address newPrevDataSource, uint256 withdrawOwnership) public {
  DataSourceInfo storage provider = infoMap[dataSource];
  require(withdrawOwnership <= provider.ownerships[msg.sender]);
  _removeDataSource(dataSource, prevDataSource);
  uint256 newOwnership = provider.totalOwnerships.sub(withdrawOwnership);
  uint256 currentStakerStake = getStake(dataSource, msg.sender);
  if (currentStakerStake > provider.tokenLocks[msg.sender]){
    uint256 unrealizedStake = currentStakerStake.sub(provider.tokenLocks[msg.sender]);
    require(token.transfer(msg.sender, unrealizedStake));
    require(token.lock(msg.sender, unrealizedStake));
  }
  uint256 withdrawAmount = provider.stake.mul(withdrawOwnership).div(provider.totalOwnerships);
  uint256 newStake = provider.stake.sub(withdrawAmount);
  uint256 newStakerTokenLock = currentStakerStake.sub(withdrawAmount);
```

```
    uint256 newStakerOwnership = provider.ownerships[msg.sender].sub(withdrawOwnership);
    provider.stake = newStake;
    provider.totalOwnerships = newOwnership;
    provider.ownerships[msg.sender] = newStakerOwnership;
    provider.tokenLocks[msg.sender] = newStakerTokenLock;
    uint256 delay;
    if (msg.sender == provider.owner && (delay = params.get(prefix, "withdraw_delay")) > 0) {
      uint256 withdrawTime = now.add(delay);
      require(withdrawTime < (1 << 64));
      withdrawReceipts.push(WithdrawReceipt({
        owner: provider.owner,
        amount: withdrawAmount,
        withdrawTime: uint64(withdrawTime),
        isWithdrawn: false
      }));
      emit WithdrawReceiptCreated(withdrawReceipts.length - 1, provider.owner, withdrawAmount, uint64(withdrawTime));
    } else {
      require(token.unlock(msg.sender, withdrawAmount));
    }
    emit DataSourceUnstaked(dataSource, msg.sender, withdrawAmount);
    if (getStake(dataSource, provider.owner) >= params.get(prefix, "min_provider_stake")) {
      _addDataSource(dataSource, newPrevDataSource);
    }
    _rebalanceLists();
  }

  //@CTK NO_ASF
  /*@CTK distributeFee
    @tag spec
    @tag is_pure
    @post address(this).balance > 0
    @post __post.undistributedReward == undistributedReward + tokenAmount
    providerReward == __post.undistributedReward / activeCount
   */
  function distributeFee(uint256 tokenAmount) public {
    require(address(this).balance > 0);
    registry.exchange().convertFromEthToBand.value(address(this).balance)();
    bondingCurve.buy(address(this), registry.band().balanceOf(address(this)), tokenAmount);
    undistributedReward = undistributedReward.add(tokenAmount);
    uint256 providerReward = undistributedReward.div(activeCount);
    uint256 ownerPercentage = params.get(prefix, "owner_revenue_pct");
    uint256 ownerReward = ownerPercentage.mulFrac(providerReward);
    uint256 stakeIncreased = providerReward.sub(ownerReward);
    address dataSourceAddress = activeList[ACTIVE_GUARD];
    /*@CTK loop_distributeFee
      @inv forall i: address. (this.activeList[i] != NOT_FOUND) /\ (this.activeList[i] != ACTIVE_GUARD)) -> (this.infoMap[this.activeList[i]].stake) >= (this__pre.infoMap[this.activeList[i]].stake)
      @inv undistributedReward <= undistributedReward__pre
      @post forall i: address. (this.activeList[i] == this__pre.activeList[i])
      @post !__should_return
     */
    while (dataSourceAddress != ACTIVE_GUARD) {
```

```
    DataSourceInfo storage provider = infoMap[dataSourceAddress];
    provider.stake = provider.stake.add(stakeIncreased);
    if (ownerReward > 0) _stake(provider.owner, ownerReward, dataSourceAddress);
    undistributedReward = undistributedReward.sub(providerReward);
    emit FeeDistributed(dataSourceAddress, providerReward, ownerReward);
    dataSourceAddress = activeList[dataSourceAddress];
  }
}

/*@CTK unlockTokenFromReceipt
  @tag assume_completion
  @post !(withdrawReceipts[receiptId].isWithdrawn)
  @post (now >= withdrawReceipts[receiptId].withdrawTime)
  @post __post.withdrawReceipts[receiptId].isWithdrawn == true
 */
function unlockTokenFromReceipt(uint256 receiptId) public {
  WithdrawReceipt storage receipt = withdrawReceipts[receiptId];
  require(!receipt.isWithdrawn && now >= receipt.withdrawTime);
  receipt.isWithdrawn = true;
  require(token.unlock(receipt.owner, receipt.amount));
  emit WithdrawReceiptUnlocked(receiptId, receipt.owner, receipt.amount);
}

/*@CTK _stake
  @tag assume_completion
  @pre infoMap[dataSource].totalOwnerships > 0
  @post __post.infoMap[dataSource].ownerships[staker] == (infoMap[dataSource].ownerhips[staker]) + (infoMap[dataSource].totalOwnerships) * (infoMap[dataSource].stake + value) / (infoMap[dataSource].stake) - (infoMap[dataSource].totalOwnerships)
  @post __post.infoMap[dataSource].stake == (infoMap[dataSource].stake + value)
  @post __post.infoMap[dataSource].totalOwnerships == (infoMap[dataSource].totalOwnerships) * (infoMap[dataSource].stake + value) / (infoMap[dataSource].stake)
 */
function _stake(address staker, uint256 value, address dataSource) internal {
  DataSourceInfo storage provider = infoMap[dataSource];
  require(provider.totalOwnerships > 0);
  uint256 newStake = provider.stake.add(value);
  uint256 newtotalOwnerships = newStake.mul(provider.totalOwnerships).div(provider.stake);
  uint256 newStakerOwnership = provider.ownerships[staker].add(newtotalOwnerships.sub(provider.totalOwnerships));
  provider.ownerships[staker] = newStakerOwnership;
  provider.stake = newStake;
  provider.totalOwnerships = newtotalOwnerships;
  emit DataSourceStaked(dataSource, staker, value);
}

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK _compare_stake_same
  @post (uint(dataSourceLeft) == uint(dataSourceRight)) -> __return == Order.EQ
 */
/*@CTK _compare_stake_untie
  @pre uint(dataSourceLeft) != uint(dataSourceRight)
  @pre infoMap[dataSourceLeft].stake != infoMap[dataSourceRight].stake
```

```
    @post (infoMap[dataSourceLeft].stake < infoMap[dataSourceRight].stake) -> __return == Order.LT
    @post (infoMap[dataSourceLeft].stake > infoMap[dataSourceRight].stake) -> __return == Order.GT
   */
  /*@CTK _compare_stake_tie
    @pre dataSourceLeft != dataSourceRight
    @pre infoMap[dataSourceLeft].stake == infoMap[dataSourceRight].stake
    @post (dataSourceLeft < dataSourceRight) -> __return == Order.LT
    @post (dataSourceLeft >= dataSourceRight) -> __return == Order.GT
   */
  function _compare(address dataSourceLeft, address dataSourceRight) internal view returns (Order) {
    if (dataSourceLeft == dataSourceRight) return Order.EQ;
    DataSourceInfo storage leftProvider = infoMap[dataSourceLeft];
    DataSourceInfo storage rightProvider = infoMap[dataSourceRight];
    if (leftProvider.stake != rightProvider.stake) return leftProvider.stake < rightProvider.stake ? Order.LT : Order.GT;
    return uint256(dataSourceLeft) < uint256(dataSourceRight) ? Order.LT : Order.GT; /// Arbitrary tie-breaker
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK _findPrevDataSource_in_activeList
    @tag assume_completion
    @pre (activeCount != 0) && (infoMap[dataSource].stake >= infoMap[activeList[ACTIVE_GUARD]].stake)
   */
  /*@CTK _findPrevDataSource_in_reserveList
    @tag assume_completion
    @pre (activeCount == 0) || (infoMap[dataSource].stake < infoMap[activeList[ACTIVE_GUARD]].stake)
    @pre reserveCount != 0
   */
  /*@CTK _findPrevDataSource_default
    @tag assume_completion
    @pre (activeCount == 0) || (infoMap[dataSource].stake < infoMap[activeList[ACTIVE_GUARD]].stake)
    @pre reserveCount == 0
    @post __return == RESERVE_GUARD
   */
  function _findPrevDataSource(address dataSource) internal view returns (address) {
    if (activeCount != 0 && _compare(dataSource, activeList[ACTIVE_GUARD]) != Order.LT) {
      address currentIndex = ACTIVE_GUARD;
      /*@CTK loop_findActivePosition
        @inv forall i: address. (this.activeList[i] == this__pre.activeList[i])
        @inv forall i: address. ((this.activeList[i] != NOT_FOUND) /\ (i != ACTIVE_GUARD) /\ (this.activeList[i] != ACTIVE_GUARD)) -> (this.infoMap[i].stake) <= (this.infoMap[this.activeList[i]].stake)
        @post (this.infoMap[currentIndex].stake) <= (this.infoMap[dataSource].stake)
        @post !__should_return
       */
      while (activeList[currentIndex] != ACTIVE_GUARD) {
        address nextIndex = activeList[currentIndex];
        if (_compare(dataSource, nextIndex) == Order.GT) currentIndex = nextIndex;
        else break;
      }
      return currentIndex;
    } else if (reserveCount != 0) {
      address currentIndex = RESERVE_GUARD;
      /*@CTK loop_findReservePosition
        @inv forall i: address. (this.reserveList[i] == this__pre.reserveList[i])
        @inv forall i: address. ((this.reserveList[i] != NOT_FOUND) /\ (this.reserveList[i] != RESERVE_GUARD) /\ (i != RESERVE_GUARD)) -> (this.infoMap[i].stake) >= (this.infoMap[this.reserveList[i]].stake)
        @post this.infoMap[currentIndex].stake <= this.infoMap[dataSource].stake
        @post !__should_return
       */
      while (reserveList[currentIndex] != RESERVE_GUARD) {
        address nextIndex = reserveList[currentIndex];
        if (_compare(dataSource, nextIndex) == Order.LT) currentIndex = nextIndex;
        else break;
      }
      return currentIndex;
    } else {
      return RESERVE_GUARD;
    }
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK _addDataSource_activeList
    @tag assume_completion
    @pre activeList[prevDataSource] != NOT_FOUND
    @post (prevDataSource == ACTIVE_GUARD) -> (reserveCount == 0) || (infoMap[dataSource].stake >= infoMap[reserveList[RESERVE_GUARD]].stake)
    @post (prevDataSource != ACTIVE_GUARD) -> (infoMap[dataSource].stake >= infoMap[prevDataSource].stake)
    @post (activeList[prevDataSource] == ACTIVE_GUARD) || (infoMap[activeList[prevDataSource]].stake > infoMap[dataSource].stake)
    @post __post.activeList[dataSource] == activeList[prevDataSource]
    @post __post.activeList[prevDataSource] == dataSource
    @post __post.activeCount == activeCount + 1
   */
  /*@CTK _addDataSource_reserveList
    @tag assume_completion
    @pre activeList[prevDataSource] == NOT_FOUND
    @pre reserveList[prevDataSource] != NOT_FOUND
    @post (prevDataSource == RESERVE_GUARD) -> (activeCount == 0) && infoMap[activeList[ACTIVE_GUARD]].stake >= infoMap[dataSource].stake
    @post (prevDataSource != RESERVE_GUARD) -> infoMap[prevDataSource].stake >= infoMap[dataSource].stake
    @post (reserveList[prevDataSource] == RESERVE_GUARD) || (infoMap[dataSource].stake >= infoMap[reserveList[prevDataSource]])
    @post __post.reserveList[dataSource] == reserveList[prevDataSource]
    @post __post.reserveList[prevDataSource] == dataSource
    @post __post.reserveCount == reserveCount + 1
   */
  /*@CTK _addDataSource_nonapplicable
    @tag assume_completion
    @pre activeList[prevDataSource] == NOT_FOUND
    @pre reserveList[prevDataSource] == NOT_FOUND
    @post __reverted
   */
  function _addDataSource(address dataSource, address _prevDataSource) internal {
    address prevDataSource = _prevDataSource == NOT_FOUND ? _findPrevDataSource(dataSource) : _prevDataSource;
    if (activeList[prevDataSource] != NOT_FOUND) {
      if (prevDataSource == ACTIVE_GUARD) require(reserveCount == 0 || _compare(dataSource, reserveList[RESERVE_GUARD]) == Order.GT);
      else require(_compare(dataSource, prevDataSource) == Order.GT);
      require(activeList[prevDataSource] == ACTIVE_GUARD || _compare(activeList[prevDataSource], dataSource) == Order.GT);
      activeList[dataSource] = activeList[prevDataSource];
      activeList[prevDataSource] = dataSource;
      activeCount++;
    } else if (reserveList[prevDataSource] != NOT_FOUND) {
      if (prevDataSource == RESERVE_GUARD) require(activeCount == 0 || _compare(activeList[ACTIVE_GUARD], dataSource) == Order.GT);
      else require(_compare(prevDataSource, dataSource) == Order.GT);
      require(reserveList[prevDataSource] == RESERVE_GUARD || _compare(dataSource, reserveList[prevDataSource]) == Order.GT);
      reserveList[dataSource] = reserveList[prevDataSource];
      reserveList[prevDataSource] = dataSource;
      reserveCount++;
    } else {
      revert();
    }
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK _removeDataSource_activeList
    @tag assume_completion
    @pre activeList[dataSource] != NOT_FOUND && reserveList[dataSource] != NOT_FOUND
    @pre (activeList[prevDataSource] != NOT_FOUND)
    @post (dataSource != ACTIVE_GUARD)
    @post (activeList[prevDataSource] == dataSource)
    @post (__post.activeList[prevDataSource] == activeList[dataSource])
    @post (__post.activeList[dataSource] == NOT_FOUND)
    @post (__post.activeCount == activeCount - 1)
    @post (__post.activeCount >= 0)
   */
  /*@CTK _removeDataSource_reserveList
    @tag assume_completion
    @pre activeList[dataSource] != NOT_FOUND && reserveList[dataSource] != NOT_FOUND
    @pre (reserveList[prevDataSource] != NOT_FOUND)
    @post (dataSource != RESERVE_GUARD)
    @post (reserveList[prevDataSource] == dataSource)
    @post (__post.reserveList[prevDataSource] == reserveList[dataSource])
    @post (__post.reserveList[dataSource] == NOT_FOUND)
    @post (__post.reserveCount == reserveCount - 1)
    @post (__post.reserveCount >= 0)
   */
  function _removeDataSource(address dataSource, address _prevDataSource) internal {
    if (activeList[dataSource] == NOT_FOUND && reserveList[dataSource] == NOT_FOUND) return;
    address prevDataSource = _prevDataSource == NOT_FOUND ? _findPrevDataSource(dataSource) : _prevDataSource;
    if (activeList[prevDataSource] != NOT_FOUND) {
      require(dataSource != ACTIVE_GUARD);
      require(activeList[prevDataSource] == dataSource);
      activeList[prevDataSource] = activeList[dataSource];
      activeList[dataSource] = NOT_FOUND;
      activeCount--;
    } else if (reserveList[prevDataSource] != NOT_FOUND) {
      require(dataSource != RESERVE_GUARD);
      require(reserveList[prevDataSource] == dataSource);
      reserveList[prevDataSource] = reserveList[dataSource];
      reserveList[dataSource] = NOT_FOUND;
      reserveCount--;
    }
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK _rebalanceLists
    @tag assume_completion
    @post forall i: address. ((__post.activeList[i] != NOT_FOUND) /\ (i != ACTIVE_GUARD) /\ (__post.activeList[i] != ACTIVE_GUARD)) -> (__post.infoMap[i].stake) <= (__post.infoMap[__post.activeList[i]].stake)
    @post forall i: address. ((__post.reserveList[i] != NOT_FOUND) /\ (__post.reserveList[i] != RESERVE_GUARD) /\ (i != RESERVE_GUARD)) -> (__post.infoMap[i].stake) >= (__post.infoMap[__post.reserveList[i]].stake)
    @post __post.infoMap[RESERVE_GUARD].stake <= __post.infoMap[ACTIVE_GUARD].stake
   */
  function _rebalanceLists() internal {
    uint256 maxProviderCount = params.get(prefix, "max_provider_count");
    /*@CTK loop_rebalanceLists_active_active_supplement
      @inv activeCount <= maxProviderCount
      @inv reserveCount >= 0
      @post (activeCount == maxProviderCount) || (reserveCount == 0)
      @post !__should_return
     */
    while (activeCount < maxProviderCount && reserveCount > 0) {
      address dataSource = reserveList[RESERVE_GUARD];
      _removeDataSource(dataSource, RESERVE_GUARD);
      _addDataSource(dataSource, ACTIVE_GUARD);
    }
    /*@CTK loop_rebalanceLists_active_cleanup
      @post activeCount < maxProviderCount
      @post !__should_return
     */
    while (activeCount > maxProviderCount) {
      address dataSource = activeList[ACTIVE_GUARD];
      _removeDataSource(dataSource, ACTIVE_GUARD);
      _addDataSource(dataSource, RESERVE_GUARD);
    }
  }
}
```

File data/OffchainAggTCD.sol

```
pragma solidity 0.5.9;

import "openzeppelin-solidity/contracts/math/SafeMath.sol";
import "./TCDBase.sol";
```

```


/// "OffchainAggTCD" is a TCD that curates a list of trusted addresses. Data points from all reporters are aggregated
/// off-chain and reported using ‘report‘ function with ECDSA signatures. Data providers are responsible for combining
/// data points into one aggregated value together with timestamp and status, which will be reported to this contract.
contract OffchainAggTCD is TCDBase {
  using SafeMath for uint256;

  event DataUpdated(bytes key, uint256 value, uint64 timestamp, QueryStatus status);

  struct DataPoint {
    uint256 value;
    uint64 timestamp;
    QueryStatus status;
  }

  mapping (bytes => DataPoint) private aggData;

  constructor(bytes8 _prefix, BondingCurve _bondingCurve, Parameters _params, BandRegistry _registry)
    public TCDBase(_prefix, _bondingCurve, _params, _registry) {}

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK OffchainAggTCD_queryPrice
    @tag assume_completion
   */
  function queryPrice() public view returns (uint256) {
    return params.get(prefix, "query_price");
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK report
    @tag assume_completion
    @post v.length == r.length
    @post v.length == s.length
    @post v.length > activeCount * (2 / 3)
   */
  function report(
    bytes calldata key, uint256 value, uint64 timestamp, QueryStatus status,
    uint8[] calldata v, bytes32[] calldata r, bytes32[] calldata s
  ) external {
    require(v.length == r.length && v.length == s.length);
    require(v.length.mul(3) > activeCount.mul(2));
    bytes32 message = keccak256(abi.encodePacked(
      "\x19Ethereum Signed Message:\n32",
      keccak256(abi.encodePacked(key, value, timestamp, status, address(this))))
    );
    address lastSigner = address(0);
    /*@CTK loop_report
      @inv activeList[recovered] != NOT_FOUND
      @inv recovered > lastSigner
      @inv i <= values.length
      @post i == values.length
      @post !__should_return
     */
    for (uint256 i = 0; i < v.length; ++i) {
      address recovered = ecrecover(message, v[i], r[i], s[i]);
      require(activeList[recovered] != NOT_FOUND);
      require(recovered > lastSigner);
      lastSigner = recovered;
    }
    require(timestamp > aggData[key].timestamp && uint256(timestamp) <= now);
    aggData[key] = DataPoint({
      value: value,
      timestamp: timestamp,
      status: status
    });
    emit DataUpdated(key, value, timestamp, status);
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK queryImpl
    @tag assume_completion
    @post (aggData[input].timestamp == 0) -> (status == QueryStatus.NOT_AVAILABLE)
    @post (aggData[input].timestamp != 0) && (aggData[input].status != QueryStatus.OK) -> (status == aggData[input].status)
    @post (aggData[input].timestamp != 0) && (aggData[input].status == QueryStatus.OK) -> (status == QueryStatus.OK)
   */
  function queryImpl(bytes memory input) internal returns (bytes32 output, uint256 updatedAt, QueryStatus status) {
    DataPoint storage data = aggData[input];
    if (data.timestamp == 0) return ("", 0, QueryStatus.NOT_AVAILABLE);
    if (data.status != QueryStatus.OK) return ("", data.timestamp, data.status);
    return (bytes32(data.value), data.timestamp, QueryStatus.OK);
  }
}
```