CONSENSYS
Diligence

Thesis - tBTC and keep Audit

Date	February 2020
Lead Auditor	Martin Ortner
Co-auditors	Alexander Wade

- 1 Executive Summary
  - 1.1 Scope
  - 1.2 Objectives
  - 1.3 Audit Log - Phase 1
  - 1.4 Audit Log - Phase 2
- 2 Recommendations
  - 2.1 Perform extensive system simulation and integration tests prior to release
  - 2.2 Consider reducing tBTC Deposit term or locking stake when in-use
  - 2.3 Disallow overfunding of tBTC Deposits
  - 2.4 Improve error handling in `bitcoin-spv`
  - 2.5 Simplify the deposit flow
  - 2.6 Remove funding fraud states in Deposit
  - 2.7 Improve signer bond seize efficiency
  - 2.8 Improve `TBTCSystem` lot size updates
  - 2.9 Explicitly track current and previous state/flow instead of deriving it from side-effects
  - 2.10 Consider emitting events for security-critical actions
  - 2.11 Review all comments
  - 2.12 Review and update the specification and documentation
  - 2.13 Review all constants and avoid changing them for testing purposes
  - 2.14 Avoid overlapping phases when using timed periods
  - 2.15 Follow best practices when upgrading and changing system variables
  - 2.16 Initialization of proxy contracts
  - 2.17 Keep group should prove that they are capable of signing a message
  - 2.18 Improve Input validation
  - 2.19 Client - Add Security Linting step to CI pipeline
  - 2.20 Client - Ensure nodes cannot be booted off the network

- 2.21 Review the Code Quality recommendations in Appendix 1
- 3 System Overview
  - 3.1 tBTC
  - 3.2 bitcoin-spv
  - 3.3 keep-tecdsa
  - 3.4 sortition-pools
  - 3.5 keep-core
- 4 Security Specification
  - 4.1 Oracles and Network Conditions
  - 4.2 Staking Token Distribution
  - 4.3 Signer collateral requirements
  - 4.4 Dependency - `bitcoin-spv`
  - 4.5 Overview - tBTC
- 5 Issues
  - 5.1 `TokenStaking.recoverStake` allows instant stake undelegation Critical ✓ Addressed
  - 5.2 Improper length validation in BLS signature library allows RNG manipulation Critical ✓ Addressed
  - 5.3 tbtc - the tecdsa keep is never closed, signer bonds are not released Critical ✓ Addressed
  - 5.4 tbtc - No access control in `TBTCSystem.requestNewKeep` Critical ✓ Addressed
  - 5.5 Unpredictable behavior due to front running or general bad timing Major ✓ Addressed
  - 5.6 keep-core - reportRelayEntryTimeout creates an incentive for nodes to race for rewards potentially wasting gas and it creates an opportunity for front-running Major ✓ Addressed
  - 5.7 keep-core - reportUnauthorizedSigning fraud proof is not bound to reporter and can be front-run Major ✓ Addressed
  - 5.8 keep-core - operator contracts disabled via panic button can be re-enabled by RegistryKeeper Major ✓ Addressed
  - 5.9 tbtc - State transitions are not always enforced Major ✓ Addressed
  - 5.10 tbtc - Funder loses payment to keep if signing group is not established in time Major Pending
  - 5.11 tbtc - Ethereum block gas limit imposes a fundamental limitation on SPV proofs Major ✓ Addressed

- 5.12 bitcoin-spv - SPV proofs do not support transactions with larger numbers of inputs and outputs Major Pending
- 5.13 bitcoin-spv - multiple integer under-/overflows Major ✓ Addressed
- 5.14 tbtc - Unreachable state `LIQUIDATION_IN_PROGRESS` Major ✓ Addressed
- 5.15 tbtc - various deposit state transitions can be front-run (e.g. fraud proofs, timeouts) Major Won't Fix
- 5.16 tbtc - Anyone can emit log events due to missing access control Major ✓ Addressed
- 5.17 `DKGResultVerification.verify` unsafe packing in signed data Medium ✓ Addressed
- 5.18 keep-core - Service contract callbacks can be abused to call into other contracts Medium ✓ Addressed
- 5.19 tbtc - Disallow signatures with high-s values in `DepositRedemption.provideRedemptionSignature` Medium ✓ Addressed
- 5.20 Consistent use of `SafeERC20` for external tokens Medium ✓ Addressed
- 5.21 Initialize implementations for proxy contracts and protect initialization methods Medium ✓ Addressed
- 5.22 keep-tecdsa - If caller sends more than is contained in the signer subsidy pool, the value is burned Medium ✓ Addressed
- 5.23 keep-core - TokenGrant and TokenStaking allow staking zero amount of tokens and front-running Medium ✓ Addressed
- 5.24 tbtc - Inconsistency between `increaseRedemptionFee` and `provideRedemptionProof` may create un-provable redemptions Medium ✓ Addressed
- 5.25 keep-tecdsa - keep cannot be closed if a members bond was seized or fully reassigned Medium ✓ Addressed
- 5.26 tbtc - `provideFundingECDSAFraudProof` attempts to burn non-existent funds Medium ✓ Addressed
- 5.27 bitcoin-spv - Bitcoin output script length is not checked in `wpkhSpendSighash` Medium Won't Fix
- 5.28 tbtc - liquidationInitiator can block purchaseSignerBondsAtAuction indefinitely Medium ✓ Addressed
- 5.29 bitcoin-spv - `verifyHash256Merkle` allows existence proofs for the same leaf in multiple locations in the tree Medium Won't Fix

- 5.30 keep-core - stake operator should not be eligible if `undelegatedAt` is set Minor ✓ Addressed
- 5.31 keep-core - Specification inconsistency: TokenStaking amount to be slashed/seized Minor ✓ Addressed
- 5.32 keep-tecdsa - Change state-mutability of `checkSignatureFraud` to view Minor ✓ Addressed
- 5.33 keep-core - Specification inconsistency: `TokenStaking.slash()` is never called Minor ✓ Addressed
- 5.34 tbtc - Remove `notifyDepositExpiryCourtesyCall` and allow `exitCourtesyCall` exiting the courtesy call at term Minor ✓ Addressed
- 5.35 keep-tecdsa - withdraw should check for zero value transfer Minor ✓ Addressed
- 5.36 keep-core - TokenStaking owner should be protected from `slash()` and `seize()` during initializationPeriod Minor ✓ Addressed
- 5.37 tbtc - Signer collusion may bypass `increaseRedemptionFee` flow Minor ✓ Addressed
- 5.38 tbtc - liquidating a deposit does not send the complete remainder of the contract balance to recipients Minor ✓ Addressed
- 5.39 tbtc - `approveAndCall` unused return parameter Minor ✓ Addressed
- 5.40 bitcoin-spv - Unnecessary memory allocation in `BTCUtils` Minor Pending
- 5.41 bitcoin-spv - `ValidateSPV.validateHeaderChain` does not completely validate input Minor Won't Fix
- 5.42 bitcoin-spv - unnecessary intermediate cast Minor ✓ Addressed
- 5.43 bitcoin-spv - unnecessary logic in `BytesLib.toBytes32()` Minor ✓ Addressed
- 5.44 bitcoin-spv - redundant functionality Minor Won't Fix
- 5.45 bitcoin-spv - unnecessary type correction Minor ✓ Addressed
- 5.46 tbtc - Restrict access to fallback function in `Deposit.sol` Minor ✓ Addressed
- 5.47 tbtc - Where possible, a specific contract type should be used rather than `address` Minor ✓ Addressed
- 5.48 tbtc - Variable shadowing in `DepositFactory` Minor ✓ Addressed
- 5.49 tbtc - Values may contain dirty lower-order bits Minor Pending
- 5.50 tbtc - Revert error string may be malformed Minor Pending

    - 5.51 tbtc - Where possible, use `constant` rather than state variables Minor ✓ Addressed
    - 5.52 tbtc - Variable shadowing in `TBTCDepositToken` constructor Minor ✓ Addressed
- Appendix 1 - Code Quality Recommendations
    - A.1.1 Possible faulty initialization process in `KeepRandomBeaconOperator`
    - A.1.2 Incomplete/Outdated comment and TODO's
    - A.1.3 Code duplication
    - A.1.4 Variable naming
    - A.1.5 Share interface definitions instead of re-defining them
    - A.1.6 Visually distinguish internal from public API
    - A.1.7 Pin Solidity Version
    - A.1.8 Use of general-purpose third-party libraries (e.g. SafeMath)
    - A.1.9 Use enums when referencing a predefined list of contextual information
    - A.1.10 Unused return values
- Appendix 2 - Files in Scope
- Appendix 3 - Artifacts
    - A.3.1 MythX
    - A.3.2 Ethlint
    - A.3.3 Surya
- Appendix 4 - Disclosure

1 Executive Summary

In January 2020, Thesis asked us to conduct a security assessment of tBTC: a trust-minimized, redeemable, Bitcoin-backed ERC20 token. tBTC utilizes and builds on functionality provided by Summa and the Keep Network.

We performed this assessment from February 03 to March 27, 2020. The assessment primarily focused on tBTC alongside its associated components. The engagement was conducted by Martin Ortner and Alexander Wade over the course of twelve person-weeks.

In addition to the review of tBTC, a review was performed of the cryptographic constructions and algorithms used in the Keep Network. A complete report of this portion of the engagement can be found here.

1.1 Scope

We analyzed code located in the following repositories at the provided commits:

Repository	Audit Revision
keep-network/tbtc	#dcb1148025d6a1238b49a80fd56d8ca0beb93781
summa-tx/bitcoin-spv	#f5e4da091a1c97e6432c2d70eba434edb189f919
keep-network/keep-tecdsa – keep-network/sortition-pools	#c69871d252378c63ab47ab3f652de0a63b09eea5 #32523a74bb5fa51345de05f756ca8a9ecf246282
keep-network/keep-core	#b76b418f04bc94030d10aff18220d8e560a2ab09

Third party dependencies not explicitly mentioned in the above list (e.g. `summa-tx/relay-sol`) were out of scope for the audit.

tBTC interacts with the Keep Network via customized interfaces from `keep-network/keep-tecdsa`, which itself uses `keep-network/sortition-pools`. The keep random beacon used for signer group election (`keep-network/keep-core`) builds on an implementation of BLS signatures on the altbn128 curve. The source code is located in five repositories with the following dependencies as seen from the tBTC solution:

- `keep-network/tbtc`
  - `summa-tx/bitcoin-spv`
  - `keep-network/keep-tecdsa`
    - `keep-network/sortition-pools`
  - `keep-network/keep-core`
- `keep-network/keep-core` (independent solution)

Together with the client, it was established that the main focus for the review would be the smart contracts in the listed repositories, with a secondary focus on reviewing the keep client (located in `keep-core`).

A complete list of files in scope can be found in the Appendix.

1.2 Objectives

Given the limited time available and ongoing development on some components in scope, we elected to begin with a top-down approach centered around tBTC as the focal point. We

started by understanding the architecture and design of high-risk components first, before diving into various system components to verify security assumptions.

Our primary objectives were to:

1. Ensure that the system is implemented consistently with the intended functionality, and without unintended edge cases.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.
3. Ensure that there is no way to break the TBTC-BTC peg and that it is as difficult as possible to abscond with deposited funds for the backing ECDSA keep.

We also sought opportunities to improve the quality of the code either by reducing the complexity, or improving clarity and readability.

1.3 Audit Log - Phase 1

The primary engagement (Feb 03 - Feb 28) was scheduled as follows:

Week 1	Week 2	Week 3	Week 4
- ramp up `tbtc` - review `bitcoin-spv`	- `bitcoin-spv` - tBTC Deposits	- tBTC Deposits - ramp up keep	- keep - `keep-tecdsa` - `sortition-pools`

Week 1

During the first week, our efforts were directed towards tBTC: understanding the intention of its design and how it uses `bitcoin-spv` to validate spv proofs and other Bitcoin transaction information. This involved defining key risk factors and potential vulnerabilities requiring further investigation. Key findings were shared with the client in an end-of-week sync meeting.

By the end of the first week, the tBTC codebase was modified from its initial audit commit to the revision v1-audit. The client also provided a frozen codebase for keep-network/keep-core. keep-network/keep-tecdsa was still undergoing changes.

Week 2

During the second week, we reviewed changes made to tBTC during the previous week. We also began a more detailed review of the tBTC codebase; in particular, tBTC Deposit flows and the investigation of potential vulnerabilities. Key findings were shared with the client in an end-of-week sync meeting and filed in the client repository where applicable. keep-network/keep-tecdsa was still undergoing changes by the end of week two.

The audit team informed the client that given the size and complexity of the audit there might not be enough time to cover all parts of the initial scope. Together with the client, it was determined that we would spend the next week finishing the review of tBTC Deposit flows before transitioning our review to `keep-core`.

Week 3

During the third week, we reviewed tBTC Deposit flows and started transitioning from tBTC to `keep-core`, maintaining a focus on the functionality of `keep-core` that was most relevant to tBTC.

The audit revision for the `keep-tecdsa` codebase was provided in the second half of the week and tagged as keep-tecdsa#v0.8.0. Additionally, the sortition-pools#v0.1.1 repository referenced by `keep-tecdsa` was added to the audit's scope.

The cryptographic review that was planned to start this week had to be delayed due to availability problems with our cryptographer. The review of the keep client was temporarily set out of scope to ensure sufficient attention was given to the smart contracts. Key findings and questions were shared immediately via the client collaboration channel and discussed in an end-of-week sync meeting.

Week 4

During the fourth week, we focused on `keep-core` and the now frozen `keep-tecdsa` implementation. The week was kicked off by the client providing a walkthrough of the relevant code of `keep-tecdsa`. Key findings and questions were shared immediately via the client collaboration channel and discussed in an end-of-week sync meeting. The **preliminary report** outlining recommendations and findings was prepared towards the end of the week targeting delivery for the following Monday.

Two-week hiatus

A two-week hiatus allowing the client to address discussion points, recommendations, and issues found during the audit was planned from March 02 to March 13.

The engagement was scheduled to be continued for a final two-week review from March 16 to March 27.

1.4 Audit Log - Phase 2

The final phase of the engagement was scheduled as follows:

Week 1	Week 2
- review fixes made during hiatus - review `keep-core`	- surface-level review of `keep-core` client - finalize report

Week 1

During the first week after providing the initial report, we focused on continuing our efforts with `keep-core` and reviewing the feedback and fixes that were provided for the initial report. A secondary goal was to start reviewing the client implementations in `keep-core`. The client provided a high-level walkthrough of the keep client codebase and the audit team shared the sources for the tBTC state diagram (see Overview - tBTC). The audit codebase was updated to the following revisions:

- `tbtc`: fbb2018c41456d19ec20eb28a17070ee2b10eb5d (noted above)
- `keep-tecdsa`: 2aab1f755e437d6e816c34a4fd354025cea5de3a (v0.10.0-rc)
- `keep-core`: 9f8b13fe54cc627548746d7e64b77d6aa50b94e1 (v0.11.0-rc] (provided on friday)
- `sortition-pools`: no update provided
- `bitcoin-spv`: no update provided

Week 2

During the second week, we continued with our focus on `keep-core` and started reviewing the client logic that is interacting with the smart contracts. The **final report** outlining recommendations and findings including client feedback and a review of provided fixes was prepared towards the end of the week targeting delivery for the following Monday. In addition to that the cryptographic review was finalized and prepared for the delivery on Monday.

2 Recommendations

During the course of our review, we identified a few possible improvements that are not security issues but can bring value to the developers and the people who want to interact with the system.

2.1 Perform extensive system simulation and integration tests prior to release

Any highly-complex system benefits massively from integration testing. tBTC and the Keep Network are no exception: the two products tie together multiple different technologies (Bitcoin, Ethereum, sMPC, …) using mission-critical smart contracts. What's more, the smart contracts in question implement strict timing windows for operations as well as steep penalties if those windows are missed.

Due in part to ongoing development on the codebase under review, no integration tests existed for the duration of this engagement. Although components of the system can be examined in relative isolation, the ability to review the system as a coherent whole is invaluable. By mimicking a production environment, integration testing helps uncover (among other things) simple issues that might otherwise only be discovered in production: misconfigurations, incorrect system-wide constants, and more.

Integration testing may also be used to simulate system behavior under a wide variety of network conditions. Due to this system's heavy reliance on coordination between multiple off-chain networks, preparation for release should not be considered complete until the system is stress-tested in multiple non-ideal environments.

2.2 Consider reducing tBTC Deposit term or locking stake when in-use

A tBTC deposit reaches its term after 180 days. During this 180 day period, signers must maintain custody of the backing BTC. Should they attempt to commit fraud, they are punished in two ways:

1. Their bonds are seized. In the case of tBTC, this should be roughly equivalent to 150% of the value of the backing BTC, in ETH.
2. Their stake is slashed.

Token stake can be undelegated and withdrawn in only 90 days. Although the security model of tBTC is mostly reliant on the seizing of signer bonds (rather than stake slashing),

this will almost certainly be abused if a signer acts maliciously.

Consider either disallowing undelegation when a signer is a member of an active keep, or reducing tBTC deposit term to under 90 days.

2.3 Disallow overfunding of tBTC Deposits

All deposits have an associated lot size, or amount of BTC. When initially funding a deposit, the funder is expected to send the entire lot-size-worth of BTC in a single transaction. In other words, funding a deposit over the course of two transactions is not supported and will result in a loss of funds.

However, overfunding a deposit is allowed: a funder can send more than the lot-size-worth of BTC to the deposit address. Consider disallowing this behavior, as it encourages users to circumvent the provided UI.

2.4 Improve error handling in `bitcoin-spv`

Several of our findings detailed potential error states in `bitcoin-spv`. Overall, the `bitcoin-spv` libraries tend to "fail silently," returning "garbage" values when error states are achieved, rather than reverting. This tendency places a larger burden on the library's users, requiring them to understand more about the library's function to use it safely. While this is a valid expectation, it is typically not realistic.

Additionally, implementing error handling in `bitcoin-spv` will allow for more negative test cases, improving overall code quality and test coverage.

2.5 Simplify the deposit flow

> *UPDATE: This recommendation has been addressed with the following statement: While simplifying the deposit flow is of interest, it's unlikely there will be time beforehand. However, we're looking at the specific recommendations of deduplication and removing path-tracking states.*

Deposit flow is highly complicated, so simplifying it wherever possible should be a priority. If possible, reduce the number states and transitions a TDT tracked deposit can be in. This also reduces the number of interactions with the deposit and therefore saves users gas.

Avoid adding states for the sole purpose of tracking what path a deposit came from and deduplicate redundant states (`LIQUIDATION`).

2.6 Remove funding fraud states in Deposit

> *UPDATE: This recommendation has been addressed with the following statement: We're investigating whether merging fraud funding with regular fraud states can be done (as part of 5.7 "tbtc - State transitions are not always enforced Major", see below).*

Deposit flow includes two types of fraud-proof. The first is submitted during the `AWAITING_BTC_FUNDING_PROOF` state and punishes signers for fraud committed during the funding stage. The second is submitted during most other states and punishes signers for fraud committed outside the funding stage.

Because the punishment differs across these two fraud-proof submission methods, there is occasionally incentive to commit fraud in the funding stage, advance the deposit state to post-funding, and submit a fraud-proof using the post-funding fraud-proof functions. In particular, post-funding fraud-proofs award the fraud-proof initiator with a cut of the bonds seized from signers, whereas funding fraud-proofs do not.

Rather than include additional complexity with different incentives, merge the two fraud submission methods and make them available throughout Deposit flow.

2.7 Improve signer bond seize efficiency

`BondedECDSAKeep.seizeSignerBonds` iterates over a keep's members, queries each member's bond amount, and seizes each member's bond individually - netting `2 * members.length` external calls:

```
    function seizeSignerBonds() external onlyOwner onlyWhenActive {
        markAsClosed();

        for (uint256 i = 0; i < members.length; i++) {
            uint256 amount = keepBonding.bondAmount(
                members[i],
                address(this),
                uint256(address(this))
```

```
            );

            keepBonding.seizeBond(
                members[i],
                uint256(address(this)),
                amount,
                address(uint160(owner))
            );
        }
    }
```

Additionally, `keepBonding.seizeBond` makes another external call to `address payable destination` for a total of `3 * members.length` calls. If any of these fail, the entire call fails and signer bonds cannot be seized.

Because this function is so crucial to the security properties of tBTC, consider switching to a pull payment system.

2.8 Improve `TBTCSystem` lot size updates

> *UPDATE: This recommendation has been addressed with the following statement: We are looking into making this change.*

`TBTCSystem` currently tracks allowed BTC lot sizes in an array, `lotSizesSatoshis`. Tracking lot sizes with an array is highly inefficient, as updates and queries require costly iteration.

Remove the array and replace it with a mapping from `uint lotSize => bool supported`. Use a setter function to allow updates of supported lot sizes. Use a getter function to query currently-allowed lot sizes.

Additionally, the setter function should include strict checks to determine whether a lot size is valid:

- `1 BTC` should always be allowed
- `0` should not be allowed
- A reasonable minimum should be enforced. One potential option is `0.001 BTC`
- A reasonable maximum should be enforced. One potential option is `10 BTC`.

2.9 Explicitly track current and previous state/flow instead of deriving it from side-effects

> *UPDATE: This recommendation has been addressed with the following statement: Of the two listed examples, only one is valid (purchaseSignerBondsAtAuction uses tdtHolder solely to implement the mechanic of "sending TBTC to the vending machine should be equivalent to burning that TBTC" efficiently; the two branches are effectively the same, except that the vending machine does not have a built-in way to handle an incoming token transfer and implement the "burn" mechanic itself). We're looking into what we can do here, and whether there are any other places that this note applies.*

We recommend explicitly tracking the origin flow/state or even the transition history in the deposit instead of deriving it from side-effects or assuming other variables contain certain values.

- `purchaseSignerBondsAtAuction` uses the `tdtHolder` address to distinguish whether a liquidation was started for an `active` deposit or not.

```
if(tdtHolder == _d.VendingMachine){
_tbtcToken.burnFrom(msg.sender, lotSizeTbtc);  // burn minimal amount to c
}
else{
_tbtcToken.transferFrom(msg.sender, tdtHolder, lotSizeTbtc);
}
```

- `startSignerFraudLiquidation` derives the origin flow from the `acutionTTBTCAmount()`

```
if (_d.auctionTBTCAmount() == 0) {
// we came from the redemption flow
_d.setLiquidated();
_d.redeemerAddress.transfer(_seized);
_d.logLiquidated();
return;
}
```

2.10 Consider emitting events for security-critical actions

> *UPDATE: This recommendation has been addressed with the following statement: Much of this has been done through the adjustments to governance actions and a few of the other PRs.*

Events can be an easy way to produce an audit trail for security-critical actions performed on the contract system. Furthermore, these events can be used to build a custom monitoring and intrusion detection system that alarms the operators of a potential upcoming attack campaign or misuse of the system and may allow cutting reaction time ensuring the safety of the system.

2.11 Review all comments

> *UPDATE: This recommendation has been addressed with the following statement: We'll be looking at comment, spec, and doc updates before launch for sure.*

As developers, we often forget to update comments when making changes because comments do not affect us immediately. However, the presence of TODO's in code implies that the codebase is not yet ready for production. This can be an oversight or a sign that code is still undergoing changes.

Make sure to review all of the comments after the code was frozen.

2.12 Review and update the specification and documentation

> *UPDATE: This recommendation has been addressed with the following statement: We'll be looking at comment, spec, and doc updates before launch for sure.*

During an audit, we typically verify that a system complies with its design and specification documents. Our review of tBTC uncovered multiple inaccuracies between the code and details in the documentation of both tbtc and keep/random-beacon. Most of these inaccuracies likely stem from recent changes to the codebase that have not yet been updated in the documentation.

We have shared a list of inconsistencies for tBTC with the client, some of which were:

- non-existent state `SIGNER_MARGIN_CALLED` mentioned in the specification
- transitions to `FRAUD_LIQUIDATION_IN_PROGRESS`

The specification states that `provideECDSAFraudProof` and `provideSPVFraudProof` transitions from the following states, which is inconsistent with the implementation:

> *from AWAITING_SIGNER_SETUP*
> *AWAITING_BTC_FUNDING_PROOF*
> *ACTIVE*
> *AWAITING_WITHDRAWAL_SIGNATURE*
> *AWAITING_WITHDRAWAL_PROOF*
> *SIGNER_MARGIN_CALLED*
> *to*
> *FRAUD_LIQUIDATION_IN_PROGRESS*

- the state `SIGNER_MARGIN_CALLED` does not exist
- the fraud-proof methods cannot be used to transitions from `AWAITING_SIGNER_SETUP`, `AWAITING_BTC_FUNDING_PROOF` to `FRAUD_LIQUIDATION_IN_PROGRESS`
- `LIQUIDATION_IN_PROGRESS` is reachable via fraud-proof

The specification mentions that in the redemption flow the state `LIQUIDATION_IN_PROGRESS` is reachable via an `ECDSA or BTC fraud-proof`.

> *Reachable exterior states*
> *LIQUIDATION_IN_PROGRESS*
> *via an ECDSA or BTC fraud-proof*
> *via a state timeout*

However, the correct state after providing a fraud-proof from redemption should be `FRAUD_LIQUIDATION_IN_PROGRESS`.

2.13 Review all constants and avoid changing them for testing purposes

> *UPDATE: This recommendation has been addressed with the following statement: We're thinking through how this might work as we start setting up a system test harness.*

Multiple system constants have been tuned for testing and were not reset to production values for the frozen audit commits. We strongly recommend avoiding permanently

changing system variables for testing. Instead, test classes and mock contracts should override constants where applicable.

Note, too, that changing important system variables for testing creates a gap where the actual system configuration for production might not receive as much testing as an artificial test scenario.

```
uint256 public constant DEPOSIT_TERM_LENGTH = 180 * 24 * 60 * 60; // 180 days
uint256 public constant TX_PROOF_DIFFICULTY_FACTOR = 1; // TODO: decreased for
```

2.14 Avoid overlapping phases when using timed periods

> *UPDATE: This recommendation has been addressed with https://github.com/keep-network/keep-core/issues/1443*

Where possible, states should be clearly distinguished from each other with no overlap. It should be avoided that objects can be in two states at the same time.

For example, in `keep-core/TokenStaking`, stake can be in the `initializationPeriod`, `active`, or `active_and_waiting_for_undelegation`.

`cancelStake` checks that the stake is within the `initializationPeriod` like so:

```
require(
    block.number <= operators[_operator].createdAt.add(initializationPeriod),
    "Initialization period is over"
);
```

`eligibleStake` verifies a stake is not in `initializationPeriod` like so:

```
bool isActive = block.number >= operator.createdAt.add(initializationPeriod);
```

In the case when `block.number == operators[_operator].createdAt.add(initializationPeriod)`, stake creation time satisfies both of these conditions and is in two states at the same time.

2.15 Follow best practices when upgrading and changing system variables

> *UPDATE: This recommendation has been addressed with the following statement: We've taken timelocks up as part of some of the listed issues above as well.*

Changing the behavior of system components via upgrading the smart contracts or modification of shared settings should be transparent and predictable for users and allow them to act on forthcoming changes. Changes that take effect immediately may allow for manipulation opportunities for the party executing the change by front-running other transactions or by setting and resetting parameters for their own profit.

We recommend implementing a time-lock that informs users of planned changes and gives them sufficient time to react to an unwanted change. It is also recommended to use a multisig contract or other transparent governance mechanisms to initiate changes.

2.16 Initialization of proxy contracts

> *UPDATE: This recommendation has been addressed with the following statement: Again, this was flagged in issues, so see above for handling.*

Ensure that implementations for proxy contracts are either initialized in the constructor when being deployed or the initialization method (and storage-changing functionality) is protected from being called by anyone. Consider rejecting calls to state reading/writing methods for contracts that are pending initialization.

It should generally be technically enforced that contracts are initialized in the same transaction as they are deployed or upgraded. This is especially true if the initialization method cannot be protected and may be called by third parties.

Ensure the existing storage layout does not change when upgrading the implementation.

2.17 Keep group should prove that they are capable of signing a message

> *UPDATE: This recommendation has been addressed with the following statement: Again, this was flagged in issues, so see above for handling.*

When a new keep is formed members start the DKG process and prove to the `BondedECDSAKeep` contract that they are capable of participating in signing requests by

submitting the public key to the contract. The fact that the keep formation succeeded is visible to consumers of the keep by checking the contracts `publicKey` state variable which is only set if all members confirmed the pubkey.

While this proves that all members submitted the same on-chain observable value `pubKey`, it does not prove that the group is capable of signing data. For example, once members confirm the public key, the funder in tBTC is able to move her deposit to the next state, sending BTC to the keep address. Given that funds are at risk we would recommend ensuring the funder (or any other consumer of the keep) that the keep group is indeed willing and capable of fulfilling signing requests. This could be accomplished by providing a message (e.g. keep owner address) signed for the funder.

The process of forming a keep group may also be susceptible to a minority attack where at least one member blocks the setup of the keep group by not confirming or confirming a wrong pubkey. The keep cannot recover from this attack, the member submitting the wrong pubKey cannot re-submit a valid one again, the key generation will time out without a pubKey being set for the keep. The keep is not activated and unable to perform signing requests. There are only three ways to proceed:

- no action, bonds stay locked in the keep
- signer bonds are seized by the owner (deposit)
- keep is closed by the owner (deposit) in which case the member bonds are released

In tBTC one would call `notifySignerSetupFailure` on the deposit now to terminate it. In any case the funder the tBTC side loses the initial payment to the keep because the keep setup was blocked and the signer bonds are not seized. On the keep side of things, all keep members bonds are locked up as the tBTC deposit does not close the keep freeing the bonds. This scenario presents a case where by investing one member bond, this member can cause the tBTC funder and the rest of the keep members to lose funds (keep payment, bonds).

There is no recommended mitigation for this other than ensuring that keeps never fail to setup. The tBTC funder cannot be reimbursed for their loss as creating a keep incurs costs. The honest majority of keep members could be reimbursed but that might open up other attack vectors. A possible solution could be to dynamically match members to a keep until all of them were able to prove that they are capable of signing for the keep but that might require major changes to the system.

2.18 Improve Input validation

> *UPDATE: This recommendation has been addressed with the following statement: We are looking into where it's feasible to improve this in available time, and whether we need to look at doing a deeper pass and delaying release.*

Input validation checks should be explicit and well documented as part of the code's documentation. This is to make sure that smart-contracts are robust against erroneous inputs and reduce the potential attack surface for exploitation.

It is good practice to verify the method's input as early as possible and only perform further actions if the validation succeeds. Methods can be split into an external or public API that performs initial checks and subsequently calls an internal method that performs the action.

There is a lack of input validation throughout the codebases under audit. For example, during the audit, we suggested implementing more restrict input validation for `bitcoin-spv` to make error conditions more explicit. Methods receiving addresses should check whether the address is valid before storing it especially if it cannot be changed afterward (optionally checking EXTCODESIZE). Known upper and lower bounds for variables must be enforced. For example, it should not be allowed to create a keep group of size zero, zero amount stake or withdraw zero eth from the staking contract. We recommend designing methods to explicitly fail early for unexpected input to allow better error handling and reduce the potential attack surface. The Checks-Effects-Interactions pattern should be used for methods and implicit error handling should be avoided (e.g. method throws because of out of bounds access in array).

2.19 Client - Add Security Linting step to CI pipeline

It is recommended to add a security-linting step to keep-core and keep-ecdsa making sure that minimum security requirements are enforced for every changeset.

The golangci-lint project is a convenient linter-aggregator that can be used for this purpose.

keep-core

```
pkg/beacon/relay/group/message_filter.go:86:2: S1008: should use 'return <expr
        if message.SenderID() == memberIndex {
        ^
cmd/start.go:23:2: `bootstrapFlag` is unused (varcheck)
        bootstrapFlag = "bootstrap"
        ^
```

```
pkg/chain/ethereum/utility.go:49:5: ineffectual assignment to `err` (ineffassi
        _, err = euc.keepRandomBeaconServiceContract.WatchRelayEntryRequested(
           ^
pkg/chain/ethereum/lib.go:53:6: `errorCallback` is unused (deadcode)
type errorCallback func(err error) (eout error)
     ^
pkg/chain/ethereum/lib.go:56:6: `sum256` is unused (deadcode)
func sum256(data []byte) (digest [32]byte) {
     ^
pkg/beacon/relay/dkg/result/signing.go:12:6: `dkgResultSignature` is unused (c
type dkgResultSignature = []byte
     ^
config/config.go:15:7: G101: Potential hardcoded credentials (gosec)
const passwordEnvVariable = "KEEP_ETHEREUM_PASSWORD"
      ^
pkg/beacon/relay/gjkr/gjkr.go:22:29: Error return value of `channel.RegisterUn
        channel.RegisterUnmarshaler(func() net.TaggedUnmarshaler {
                                    ^
pkg/beacon/relay/gjkr/gjkr.go:25:29: Error return value of `channel.RegisterUn
        channel.RegisterUnmarshaler(func() net.TaggedUnmarshaler {
                                    ^
pkg/beacon/relay/gjkr/gjkr.go:28:29: Error return value of `channel.RegisterUn
        channel.RegisterUnmarshaler(func() net.TaggedUnmarshaler {
                                    ^
pkg/beacon/relay/gjkr/protocol.go:83:37: Error return value of `sm.evidenceLog
                sm.evidenceLog.PutEphemeralMessage(ephemeralPubKeyMessage)
                                                   ^
pkg/beacon/relay/gjkr/protocol.go:312:39: Error return value of `cvm.evidenceL
                cvm.evidenceLog.PutPeerSharesMessage(sharesMessage)
                                                     ^
pkg/net/libp2p/channel.go:330:35: Error return value of `c.pubsub.UnregisterTo
        c.pubsub.UnregisterTopicValidator(c.name)
                                          ^
pkg/chain/ethereum/lib.go:58:9: Error return value of `h.Write` is not checked
        h.Write(data)
               ^
pkg/chain/ethereum/utility.go:36:15: Error return value of `promise.Fail` is n
                promise.Fail(err)
                             ^
pkg/chain/ethereum/utility.go:41:15: Error return value of `promise.Fail` is n
```

```
                promise.Fail(err)
                             ^
pkg/chain/ethereum/utility.go:56:64: Error return value of `euc.keepRandomBeac
                        euc.keepRandomBeaconServiceContract.WatchRelayEntryGer

pkg/chain/ethereum/utility.go:58:21: Error return value of `promise.Fulfill` i
                                       promise.Fulfill(&event.EntryGenerated{
                                                       ^
pkg/chain/ethereum/utility.go:76:15: Error return value of `promise.Fail` is r
                promise.Fail(err)
                             ^
pkg/internal/dkgtest/dkgtest.go:104:45: Error return value of `(github.com/kee
        chain.ThresholdRelay().OnDKGResultSubmitted(
                                                    ^
pkg/internal/entrytest/entrytest.go:110:46: Error return value of `(github.com
        chain.ThresholdRelay().OnRelayEntrySubmitted(
                                                     ^
pkg/beacon/beacon.go:65:34: Error return value of `relayChain.OnRelayEntryRequ
        relayChain.OnRelayEntryRequested(func(request *event.Request) {
                                         ^
pkg/beacon/beacon.go:91:36: Error return value of `relayChain.OnGroupSelection
        relayChain.OnGroupSelectionStarted(func(event *event.GroupSelectionSta
                                           ^
pkg/beacon/beacon.go:130:30: Error return value of `relayChain.OnGroupRegister
        relayChain.OnGroupRegistered(func(registration *event.GroupRegistratio
                                     ^
cmd/network.go:78:26: Error return value of `stakeMonitor.StakeTokens` is not
        stakeMonitor.StakeTokens(key.NetworkPubKeyToEthAddress(
                                 ^
cmd/network.go:81:26: Error return value of `stakeMonitor.StakeTokens` is not
        stakeMonitor.StakeTokens(key.NetworkPubKeyToEthAddress(
                                 ^
pkg/beacon/relay/node.go:21:2: `mutex` is unused (structcheck)
        mutex sync.Mutex
        ^
pkg/net/libp2p/channel.go:45:17: SA1019: peerstore.Peerstore is deprecated: us
        peerStore      peerstore.Peerstore
                       ^
pkg/net/libp2p/channel.go:184:9: SA1019: c.pubsub.Publish is deprecated: use p
        return c.pubsub.Publish(c.name, messageBytes)
```

```
               ^
pkg/net/libp2p/channel_manager.go:25:12: SA1019: peerstore.Peerstore is deprec
        peerStore peerstore.Peerstore
                  ^
pkg/net/libp2p/channel_manager.go:96:14: SA1019: cm.pubsub.Subscribe is deprec
        sub, err := cm.pubsub.Subscribe(name)
                    ^
pkg/net/libp2p/libp2p.go:159:17: SA1019: peer.IDB58Decode is deprecated: Use D
        peerID, err := peer.IDB58Decode(connectedPeer)
                       ^
pkg/net/libp2p/libp2p.go:181:17: SA1019: peer.IDB58Decode is deprecated: Use D
        peerID, err := peer.IDB58Decode(peerHash)
                       ^
pkg/net/libp2p/libp2p.go:435:51: SA1019: peerstore.PeerInfo is deprecated: use
func extractMultiAddrFromPeers(peers []string) ([]peerstore.PeerInfo, error) {
                                                  ^
pkg/net/libp2p/libp2p.go:436:18: SA1019: peerstore.PeerInfo is deprecated: use
        var peerInfos []peerstore.PeerInfo
                       ^
pkg/net/libp2p/libp2p.go:443:20: SA1019: peerstore.InfoFromP2pAddr is deprecat
                peerInfo, err := peerstore.InfoFromP2pAddr(ipfsaddr)
                                 ^
pkg/net/libp2p/unicast_channel_manager.go:141:21: SA1019: peer.IDB58Decode is
        remotePeer, err := peer.IDB58Decode(peerID.String())
                           ^
pkg/beacon/relay/node.go:128:2: S1023: redundant `return` statement (gosimple)
        return
        ^
pkg/beacon/relay/node.go:68:6: S1004: should use bytes.Equal(selectedStaker, n
                if bytes.Compare(selectedStaker, n.Staker.Address()) == 0 {
                   ^
pkg/beacon/relay/dkg/result/states.go:76:10: S1004: should use bytes.Equal(pha
                return bytes.Compare(phaseMessage.publicKey, msg.SenderPublicK
                       ^
pkg/net/watchtower/watchtower.go:53:2: S1005: should write `checking := g.peer
        checking, _ := g.peerCrossList[peer]
        ^
cmd/relay.go:81:2: S1000: should use a simple channel send/receive instead of
        select {
        ^
```

```
cmd/start.go:125:2: S1000: should use a simple channel send/receive instead of
        select {
        ^
pkg/chain/ethereum/ethereum.go:215:3: S1000: should use for range instead of f
                for {
                ^
pkg/chain/ethereum/ethereum.go:420:3: S1000: should use for range instead of f
                for {
                ^
pkg/beacon/relay/group/group.go:139:17: func `(*Group).isThresholdSatisfied` i
pkg/beacon/relay/group/group.go:132:17: func `(*Group).eliminatedMembersCount`
```

keep-ecdsa

```
pkg/chain/eth/local/local.go:62:15: G404: Use of weak random number generator
        handlerID := rand.Int()
                     ^
pkg/chain/eth/local/local.go:83:15: G404: Use of weak random number generator
        handlerID := rand.Int()
                     ^
internal/config/config.go:13:7: G101: Potential hardcoded credentials (gosec)
const passwordEnvVariable = "KEEP_ETHEREUM_PASSWORD"
      ^
pkg/ecdsa/tss/message.go:43:38: Error return value of `broadcastChannel.Regist
        broadcastChannel.RegisterUnmarshaler(func() net.TaggedUnmarshaler {
                                             ^
pkg/ecdsa/tss/message.go:46:38: Error return value of `broadcastChannel.Regist
        broadcastChannel.RegisterUnmarshaler(func() net.TaggedUnmarshaler {
                                             ^
pkg/ecdsa/tss/message.go:49:38: Error return value of `broadcastChannel.Regist
        broadcastChannel.RegisterUnmarshaler(func() net.TaggedUnmarshaler {
                                             ^
pkg/ecdsa/tss/network.go:266:14: Error return value of `b.broadcast` is not ch
                b.broadcast(ctx, protocolMessage)
                           ^
pkg/ecdsa/tss/network.go:279:12: Error return value of `b.sendTo` is not check
                        b.sendTo(destinationTransportID, protocolMessage)
                                ^
pkg/ecdsa/tss/network.go:294:26: Error return value of `broadcastChannel.Send`
```

```
        if broadcastChannel.Send(ctx, msg); err != nil {
                                ^
pkg/client/client.go:105:40: Error return value of `ethereumChain.OnBondedECDS
        ethereumChain.OnBondedECDSAKeepCreated(func(event *eth.BondedECDSAKeep
                                               ^
pkg/node/node.go:136:2: lostcancel: the monitoringCancel function is not used
        monitoringCtx, monitoringCancel := context.WithTimeout(
        ^
pkg/node/node.go:157:2: lostcancel: this return statement may be reached witho
        return signer, nil
        ^
cmd/start.go:163:2: S1000: should use a simple channel send/receive instead of
        select {
        ^
pkg/ecdsa/tss/protocol_announce.go:77:3: S1023: redundant `return` statement (
                return
                ^
pkg/ecdsa/tss/protocol_ready.go:82:3: S1023: redundant `return` statement (gos
                return
                ^
```

2.20 Client - Ensure nodes cannot be booted off the network

A major threat to the system is that an actor may be able to boot nodes off the network by causing a panic while interacting with them. Someone who is able to permanently or temporarily reduce the amount of responsible nodes in the system may be able directly harm the network or turn things to their favor. The threat scenario is somewhat similar to the one the go-ethereum project is facing. We therefore recommend to design the software with security zones in mind, ensuring that sub-routines that are handling untrusted input cannot terminate the application e.g. because a panic condition has been triggered. We also recommend to set-up a fuzz-testing instance with go-fuzz especially for parts that are parsing/handling untrusted input, in an effort to find yet uncaught potentially triggerable panic conditions. Where feasible, we recommend to safeguard critical functionality that is handling untrusted data by trying to recover from panic events instead of terminating the application while still logging the error condition.

2.21 Review the Code Quality recommendations in Appendix 1

> *UPDATE: This recommendation has been addressed with the following statement: We are looking into several of these as we attach linting and similar limitations consistently across the repositories in question, and also looking at making some of the changes linters wouldn't necessarily catch.*

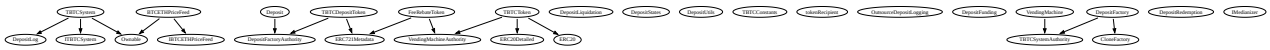
Other comments related to readability and best practices are listed in Appendix 1

3 System Overview

This section describes the top-level contracts, their inheritance structure, actors, permissions and contract interactions of the initial system under audit, not including fundamental changes the system has undergone after providing the initial report. Please refer to Section 4 - Security Specification for a security-centric view on the system.

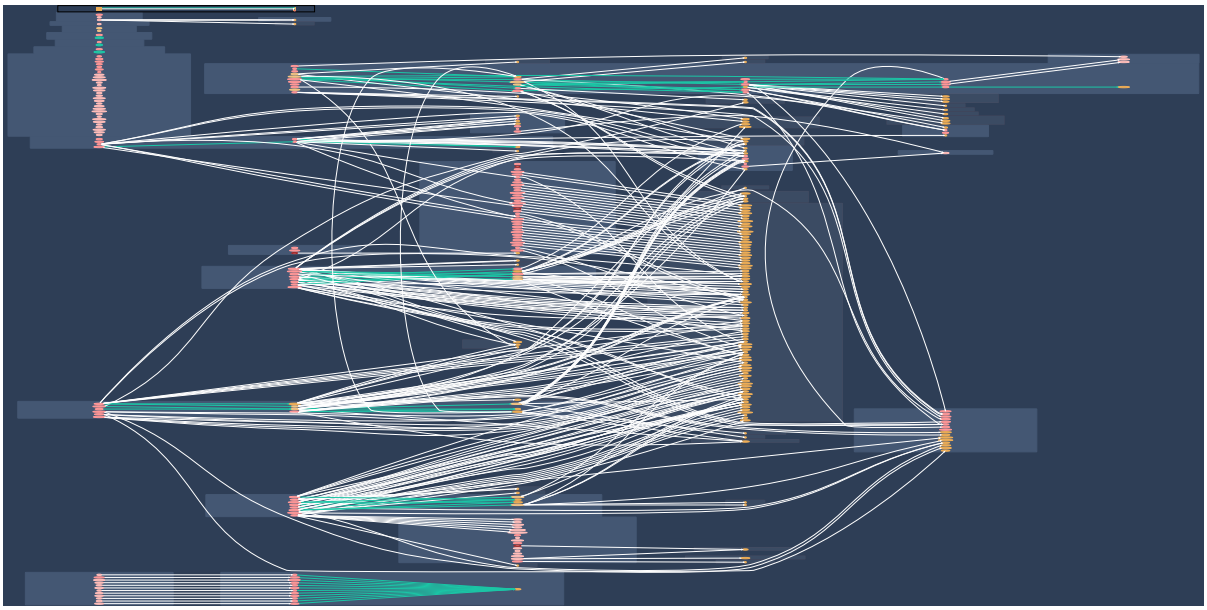
3.1 tBTC

Inheritance Structure (without `usingFor`)



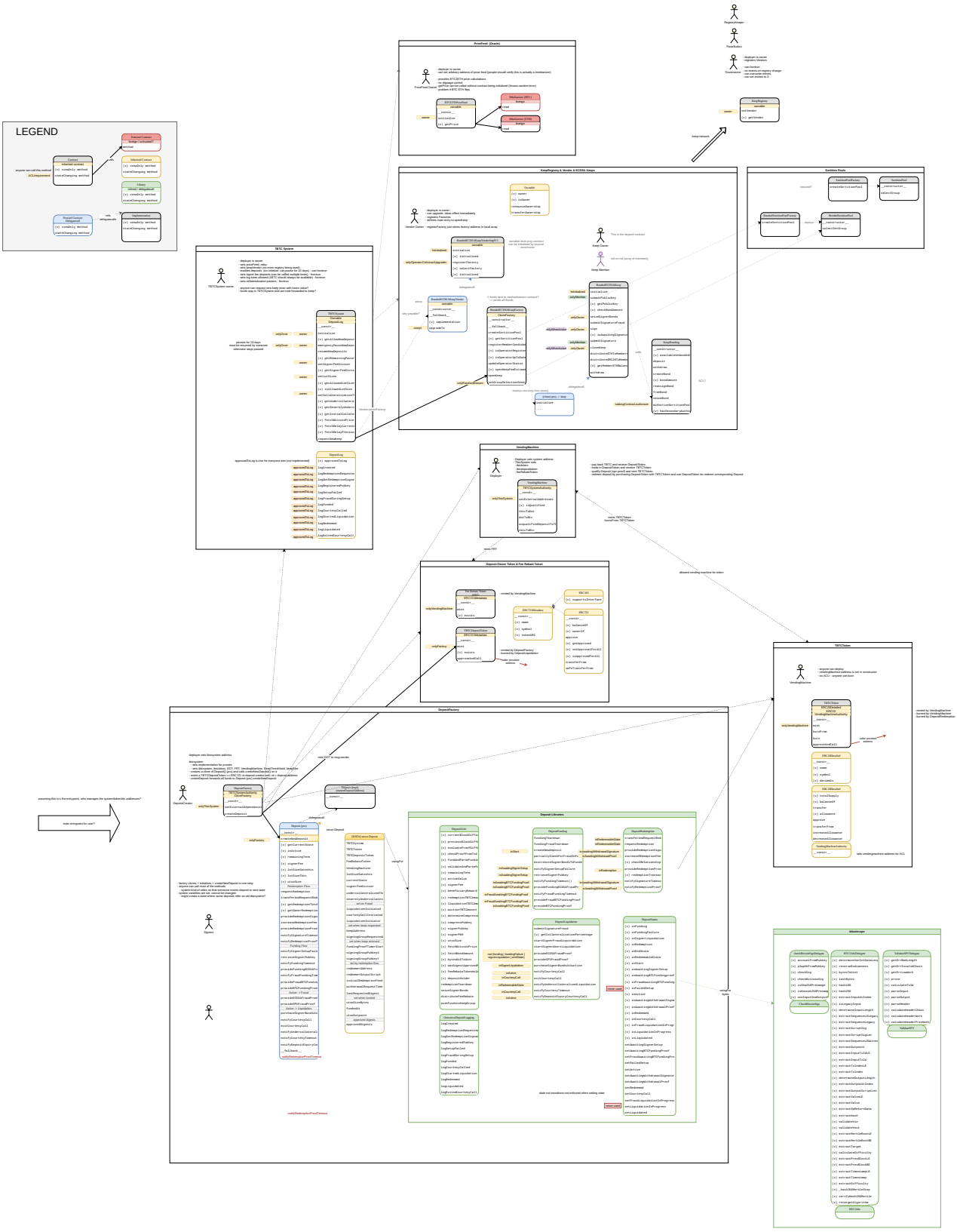
Inheritance graph

Call Graph



Function call graph and contract interaction

System Overview



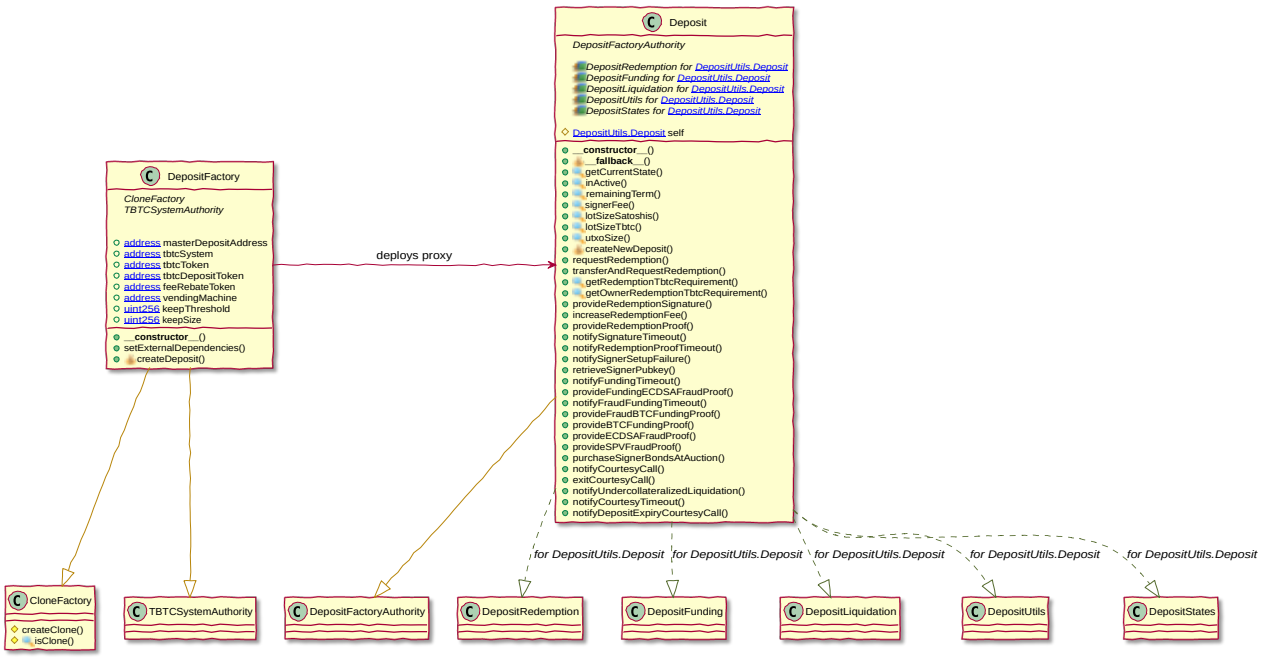
tBTC System Outline

The following components are referencing contracts in one of the other repositories that are in scope for the audit:

Contract	Repository
`DepositFunding`	`bitcoin-spv`

Contract	Repository
`DepositLiquidation`	`bitcoin-spv`
`DepositRedemption`	`bitcoin-spv`
`DepositUtils`	`bitcoin-spv`
`TBTCSystem`	`keep-tecdsa`

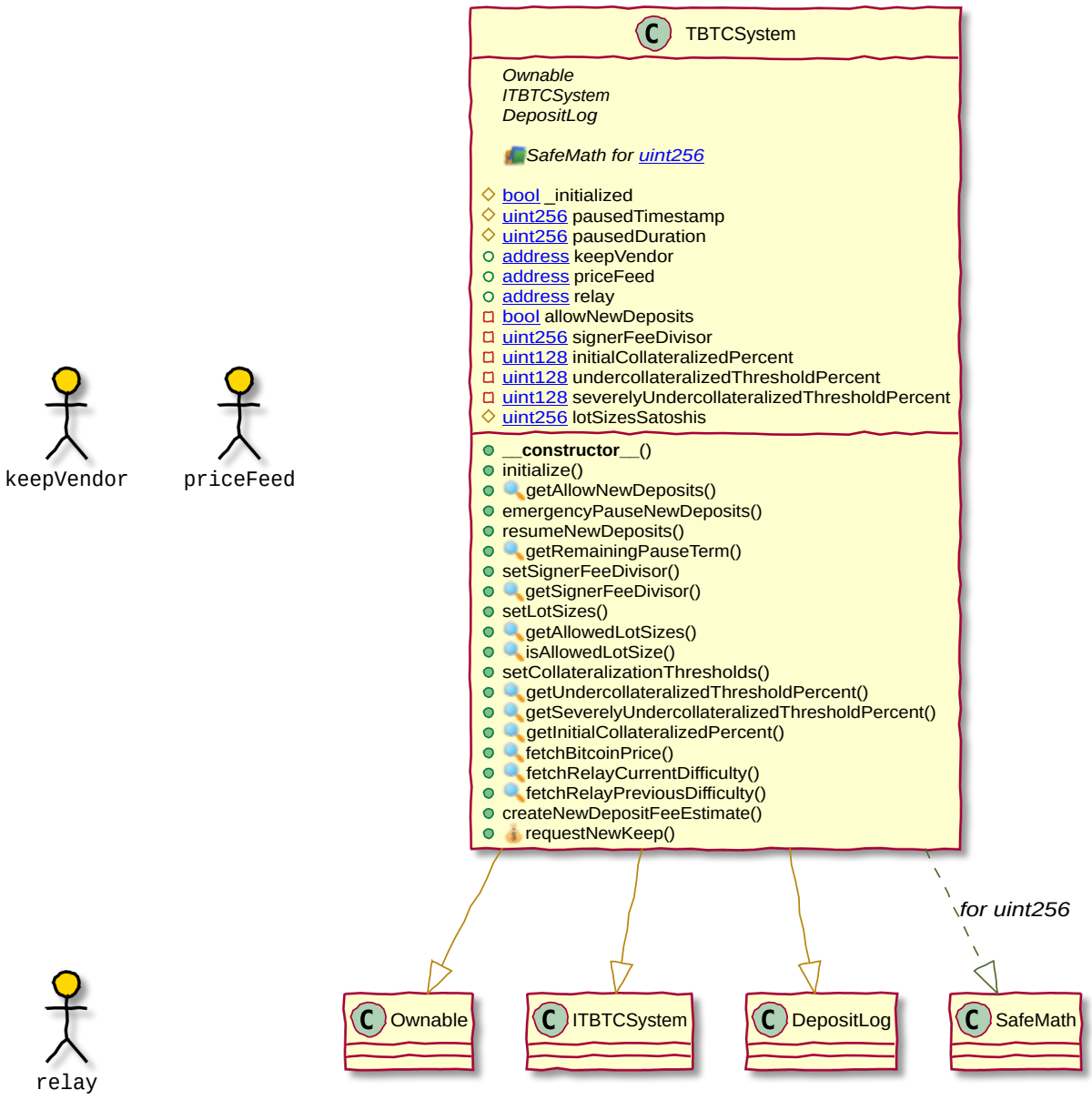
Deposits



tBTC Deposit Contract

- main entry point for users to mint `TBTC` by creating a `TBTCDepositToken` tracked deposit.
- deposits can be in various states starting with a funding flow that ultimately reached the active state, handling, and reporting of timeouts and frauds as well as undercollateralization. A deposit can also be redeemed or liquidated.

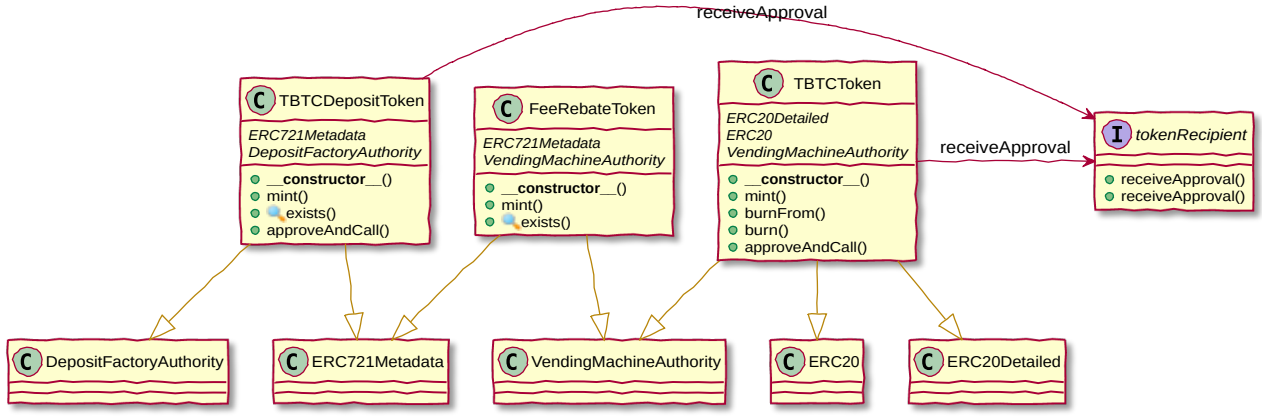
TBTCSystem



tBTC System Contract

- emits log events from Deposit contracts
- holds system variables
- is called when creating a new deposit to request a new keep (and pay the keep)

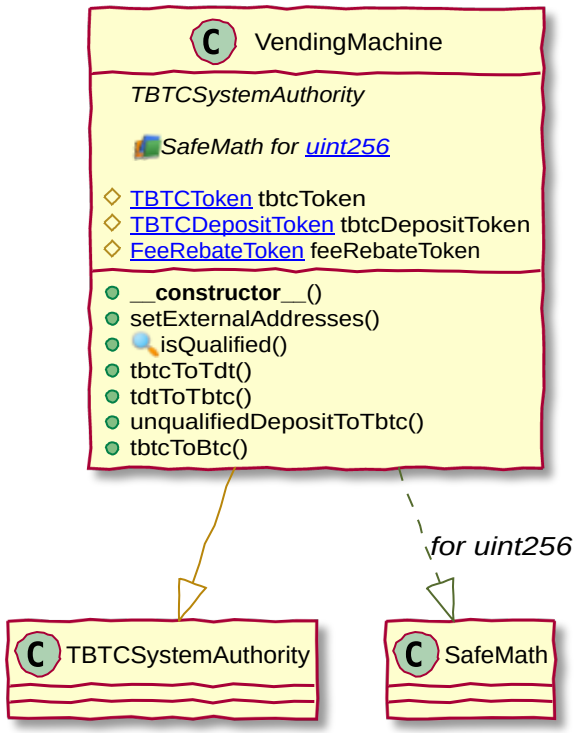
Tokens



tBTC Token Contracts

- `TBTCDepositToken` - a standard NFT (`ERC721`) that tracks a deposit to the `tBTC` system. Can be exchanged for `TBTCToken` (`ERC20`). `approveAndCall` calls out to token recipient.
- FeeRebateToken - a standard NFT (`ERC721`) ...
- `TBTCToken` - a standard `ERC20` token and the system currency. `TBTC` is backed by `BTC` collateral. `approveAndCall` calls out to token recipient.

VendingMachine



tBTC VendingMachine Contract

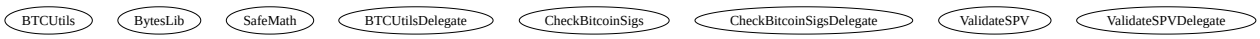
- can be used to redeem `TBTC` for `TBTCDepositToken` .

- can be used to redeem `TBTCDepositToken` for `TBTC`.
- can be used to redeem `BTC` for `TBTC` via the deposit redemption flow.

Oracle

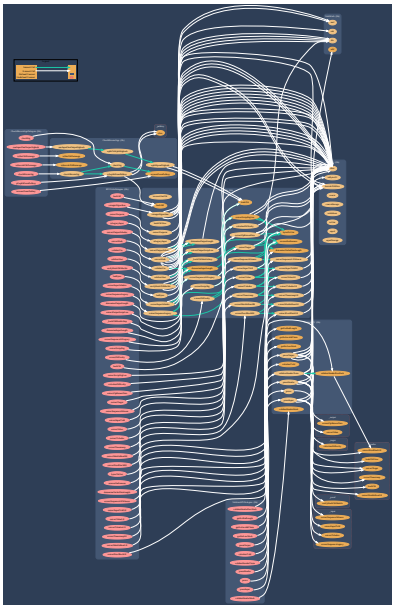
3.2 bitcoin-spv

Inheritance Structure (without `usingFor`)



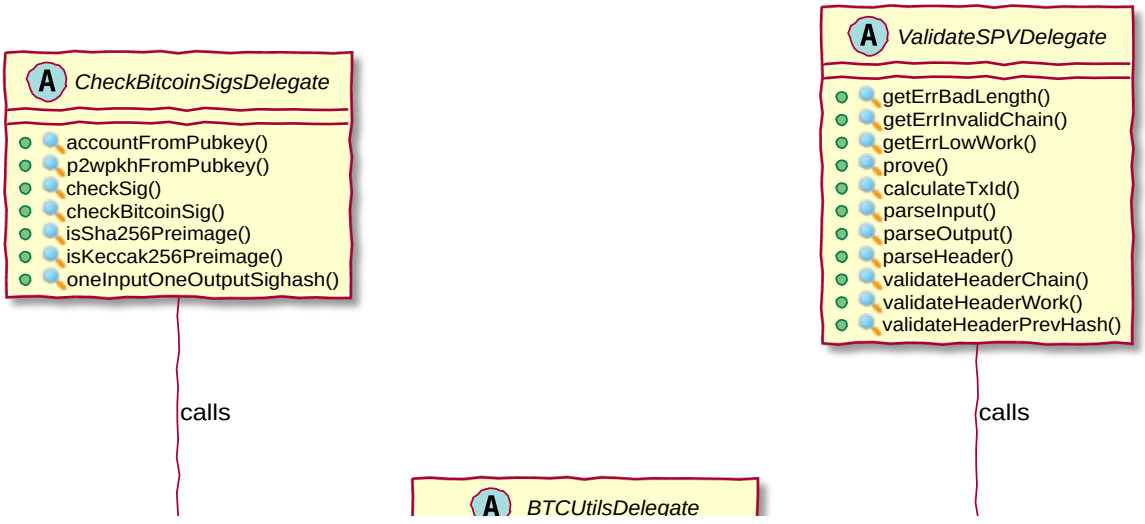
Inheritance graph

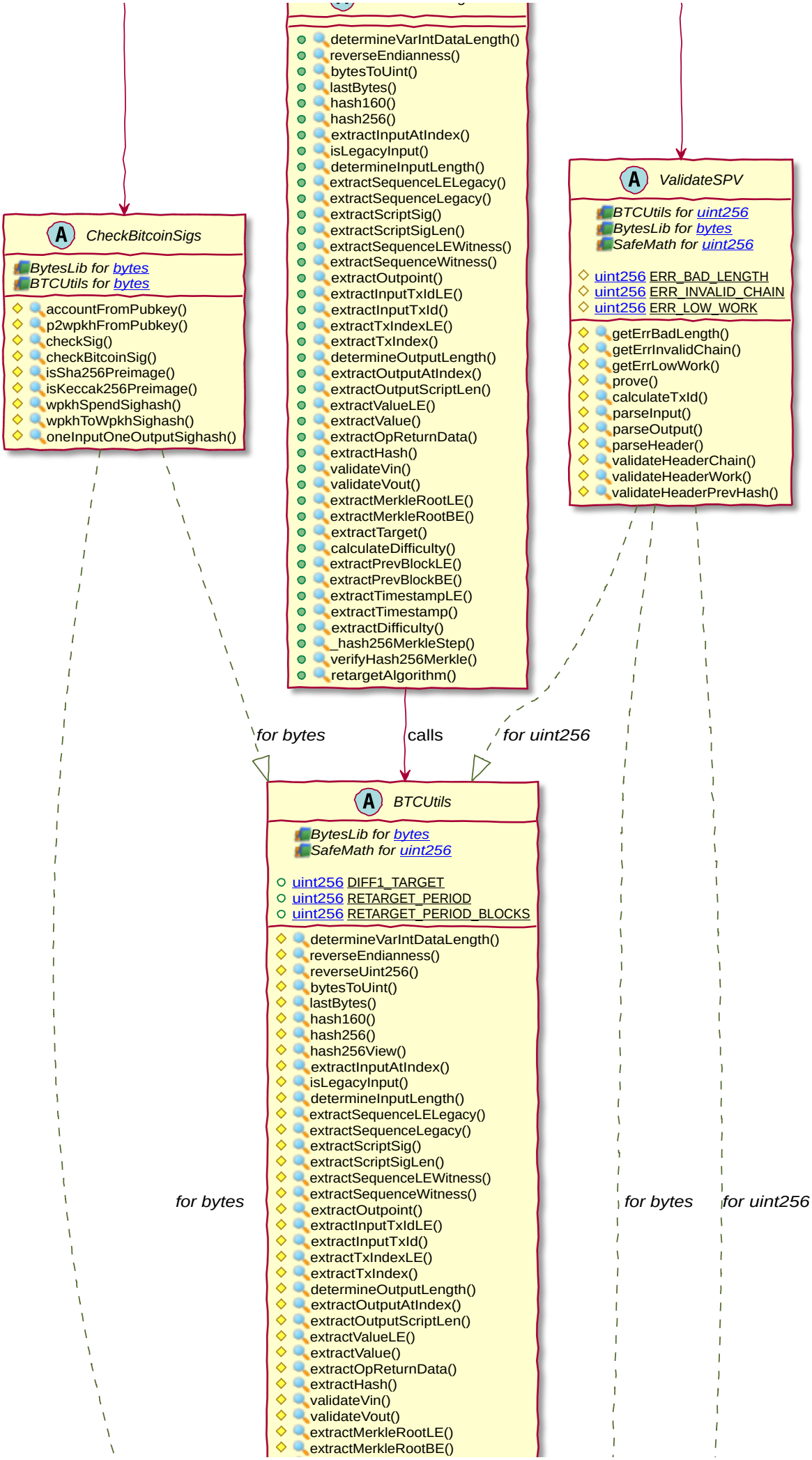
Call Graph

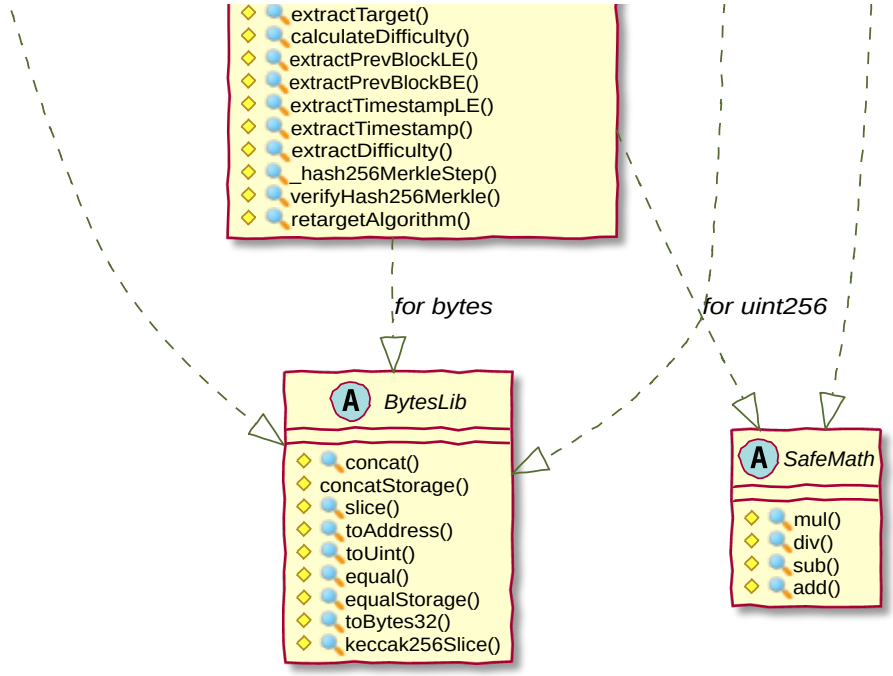


Function call graph and contract interaction

Contracts




CheckBitcoinSigs
BytesLib for bytes
BTCUtils for bytes
accountFromPubkey()
p2wpkhFromPubkey()
checkSig()
checkBitcoinSig()
isSha256Preimage()
isKeccak256Preimage()
wpkhSpendSighash()
wpkhToWpkhSighash()
oneInputOneOutputSighash()
determineVarIntDataLength()
reverseEndianness()
bytesToUint()
lastBytes()
hash160()
hash256()
extractInputAtIndex()
isLegacyInput()
determineInputLength()
extractSequenceLELegacy()
extractSequenceLegacy()
extractScriptSig()
extractScriptSigLen()
extractSequenceLEWitness()
extractSequenceWitness()
extractOutpoint()
extractInputTxIdLE()
extractInputTxId()
extractTxIndexLE()
extractTxIndex()
determineOutputLength()
extractOutputAtIndex()
extractOutputScriptLen()
extractValueLE()
extractValue()
extractOpReturnData()
extractHash()
validateVin()
validateVout()
extractMerkleRootLE()
extractMerkleRootBE()
extractTarget()
calculateDifficulty()
extractPrevBlockLE()
extractPrevBlockBE()
extractTimestampLE()
extractTimestamp()
extractDifficulty()
_hash256MerkleStep()
verifyHash256Merkle()
retargetAlgorithm()
ValidateSPV
BTCUtils for uint256
BytesLib for bytes
SafeMath for uint256
uint256 ERR_BAD_LENGTH
uint256 ERR_INVALID_CHAIN
uint256 ERR_LOW_WORK
getErrBadLength()
getErrInvalidChain()
getErrLowWork()
prove()
calculateTxId()
parseInput()
parseOutput()
parseHeader()
validateHeaderChain()
validateHeaderWork()
validateHeaderPrevHash()
for bytes
calls
for uint256
BTCUtils
BytesLib for bytes
SafeMath for uint256
uint256 DIFF1_TARGET
uint256 RETARGET_PERIOD
uint256 RETARGET_PERIOD_BLOCKS
determineVarIntDataLength()
reverseEndianness()
reverseUint256()
bytesToUint()
lastBytes()
hash160()
hash256()
hash256View()
extractInputAtIndex()
isLegacyInput()
determineInputLength()
extractSequenceLELegacy()
extractSequenceLegacy()
extractScriptSig()
extractScriptSigLen()
extractSequenceLEWitness()
extractSequenceWitness()
extractOutpoint()
extractInputTxIdLE()
extractInputTxId()
extractTxIndexLE()
extractTxIndex()
determineOutputLength()
extractOutputAtIndex()
extractOutputScriptLen()
extractValueLE()
extractValue()
extractOpReturnData()
extractHash()
validateVin()
validateVout()
extractMerkleRootLE()
extractMerkleRootBE()
for bytes
for bytes
for uint256



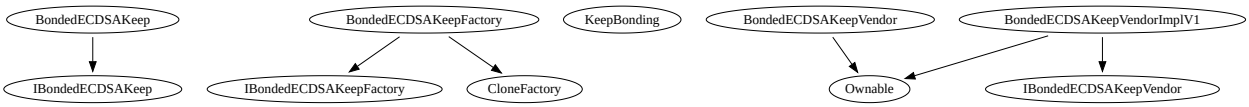
bitcoin-spv Outline

> `bitcoin-spv` *is a low-level toolkit for working with Bitcoin from other blockchains. It supplies a set of pure functions that can be used to validate almost all Bitcoin transactions and headers, as well as higher-level functions that can evaluate header chains and transaction inclusion proofs.*

The `bitcoin-spv` project is utilized by `tBTC` deposits in the broader `tBTC` and `keep` system. All contracts are library contracts and the contracts with the `*Delegate` suffix are used to ensure the library is deployed and `delegatecall`'d instead of having the compiler inline the functionality. There are three main contracts, `BTCUtils`, `CheckBitcoinSigs`, and `ValidateSPV`.

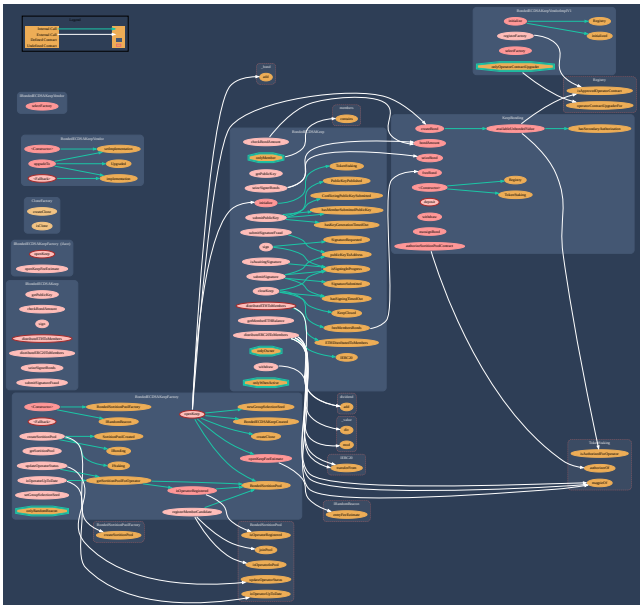
3.3 keep-tecdsa

Inheritance Structure (without `usingFor`)



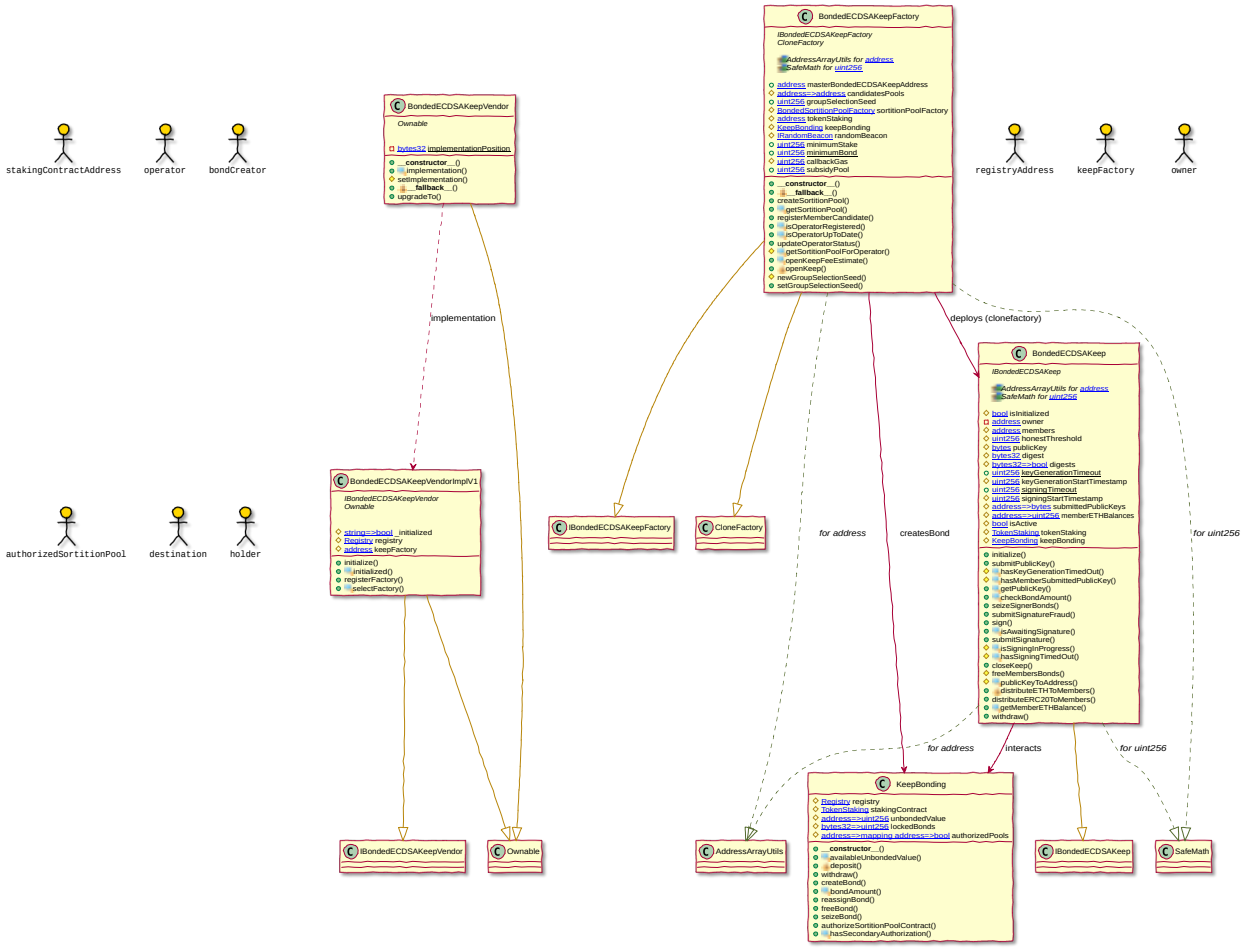
Inheritance graph

Call Graph



Function call graph and contract interaction

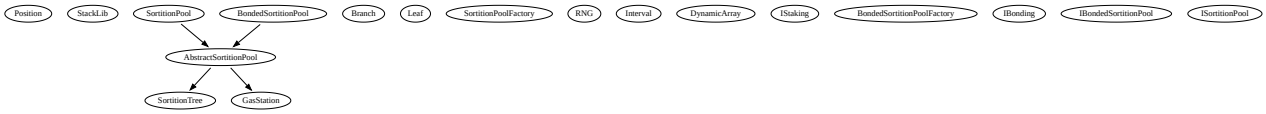
Contracts



Contract System Outline

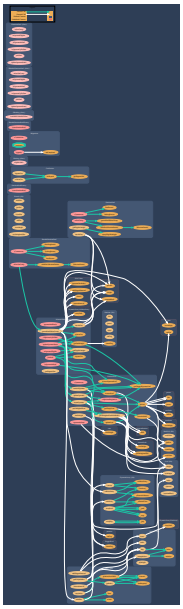
3.4 sortition-pools

Inheritance Structure (without `usingFor`)



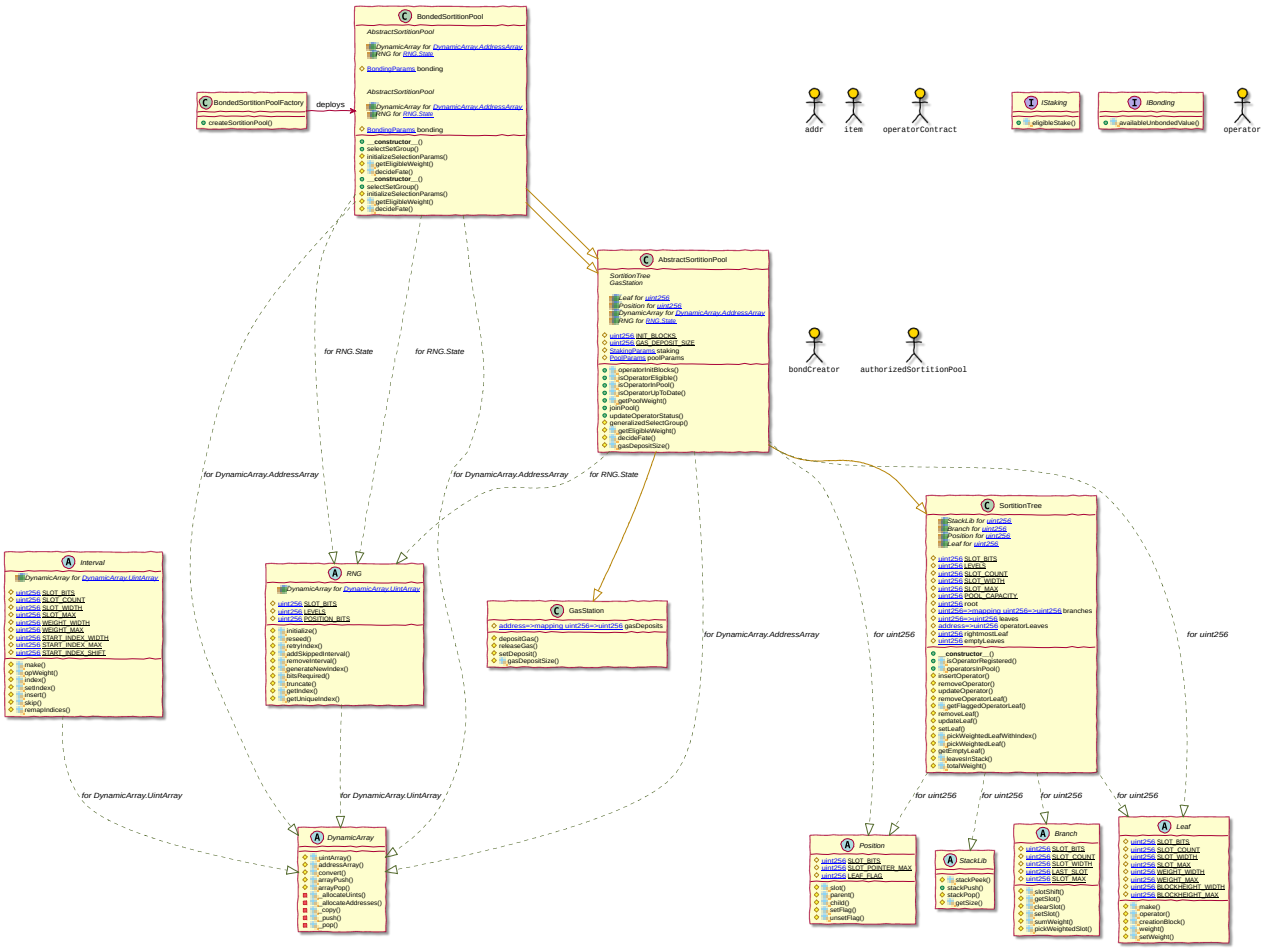
Inheritance graph

Call Graph



Function call graph and contract interaction

Contracts



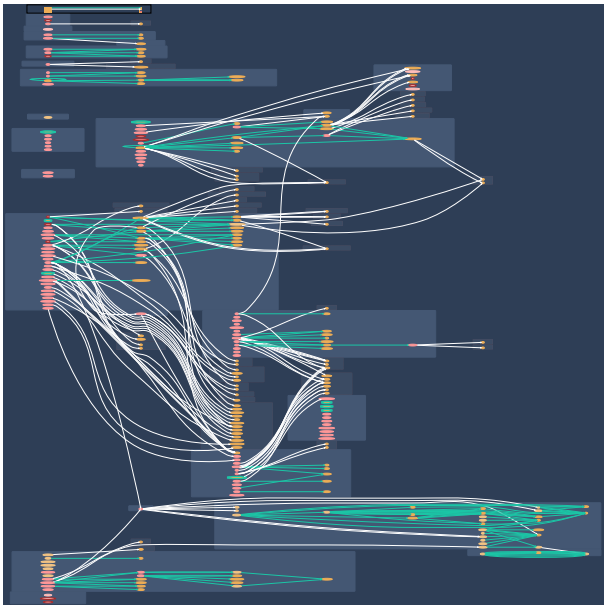
Contract System Outline

3.5 keep-core

Inheritance Structure (without `usingFor`)

Inheritance graph

Call Graph



Function call graph and contract interaction

4 Security Specification

This section describes, **from a security perspective**, the expected behavior of the system under audit. It is not a substitute for documentation. The purpose of this section is to identify specific security properties that were validated by the audit team.

4.1 Oracles and Network Conditions

The project as a whole uses several sources of information for events external to Ethereum. In particular, the following oracles are used:

- Maker's Medianizer price feed oracle is used to calculate the current price of Bitcoin relative to Ether. This value is used to calculate collateral ratios for new and existing deposits, as well as liquidation thresholds for existing collateralized deposits.
- A specialized Bitcoin difficulty relay is provided by Summa (`summa-tx/relays`). The relay ingests Bitcoin blockheaders and tracks the difficulty of Bitcoin proof of work over time. These values are used to ensure SPV proofs for redeemed deposits are sufficiently "confirmed." Note that the specific behavior of the relay is out-of-scope for this audit, but is mentioned here as it is a prominent external dependency.

The following two sources of information may not fit the strict definition of an "oracle," but are mentioned here because they each introduce an oracle-like dependency on external networks:

- The Keep Network acts as a random beacon. On request, this beacon generates random numbers and supplies them to a smart contract within the system. These random numbers are used to seed the group selection algorithm, which determines which signers become custodians of each Bitcoin deposit.
- The aforementioned signers must communicate with each other, with Bitcoin, and with the tBTC smart contracts in order to effectively manage deposits.

Reliance on external sources of data often introduces several points of failure. The efficacy of the aforementioned systems may depend on several factors:

- **Bitcoin network conditions:** tBTC depends on the relative reliance of the Bitcoin network during many stages of the deposit creation and redemption process. Because many stages of the deposit process are expected to happen within specific windows of time, a period of high stress in the Bitcoin network may impact the reliability of these timing windows significantly. The system may have difficulty coping with relatively longer confirmation times for deposit creation and redemption, as well as relatively higher transaction fees.
- **Ethereum network conditions:**
  - **Chain reorgs:** Signing groups are only authorized to create signatures when requested via tBTC smart contracts. In the event of a chain re-org, signers may find their authorization for an already-published signature removed by the re-org. In this case, it may be possible to submit a no-longer-authorized signature for ECDSA fraud, punishing the signing group.
  - **Fluctuating block gas limit:** Ethereum's gas limit fluctuates over time in response to slight adjustments by miners. Among its other effects, gas limit fluctuation has a significant impact on the size of Bitcoin transactions that can be validated via SPV proof in the tBTC system contracts. Measuring this impact is crucially important during deposit creation and redemption, as even with conservative estimates, only relatively smaller Bitcoin transactions can be verified. See this issue for details.
  - **Fluctuating gas prices:** Several components of the random beacon attempt to estimate cost (in wei) of various on-chain operations. The typical pattern used is a hardcoded gas amount multiplied by 30 GWei. The resulting amount is required to be passed in as `CALLVALUE` to many functions in the random beacon contracts. Although 30 GWei is a conservative estimate during periods of low network congestion, there is abundant historical evidence that gas prices often rise well above this value. Using these functions during periods of higher network congestion could result in participants receiving service at a significant discount.

Conversely, it could result in participants providing service with insufficient compensation.
    - **Changing opcode prices:** As mentioned above, many gas values are hardcoded. Should opcode prices change in a future fork, many contracts may need to undergo a costly upgrade process - or otherwise stop working correctly.
- **Keep network conditions:** The integrity of the Keep Network is assumed for many components of the system. Should the Keep Network cease functioning correctly, random numbers may no longer be submitted to the system contracts at expected intervals. In this case, it may become possible to game the signer group selection process and create signing groups comprised of a malicious majority of signers.

4.2 Staking Token Distribution

The keep random beacon's purpose is in part to ensure that signers are not able to manipulate group selection. Ideally, signers are geographically distinct and do not know each other. A large factor in whether this is feasible is the distribution method for the staking token required for signer eligibility.

In order to be eligible for selection, participants are required to stake tokens. Currently, staking tokens are only available for purchase direct from Thesis, making it highly likely that staking members have few degrees of separation from each other. Although other distribution methods are planned in the future, the first iteration of this system may be more prone to signer collusion.

4.3 Signer collateral requirements

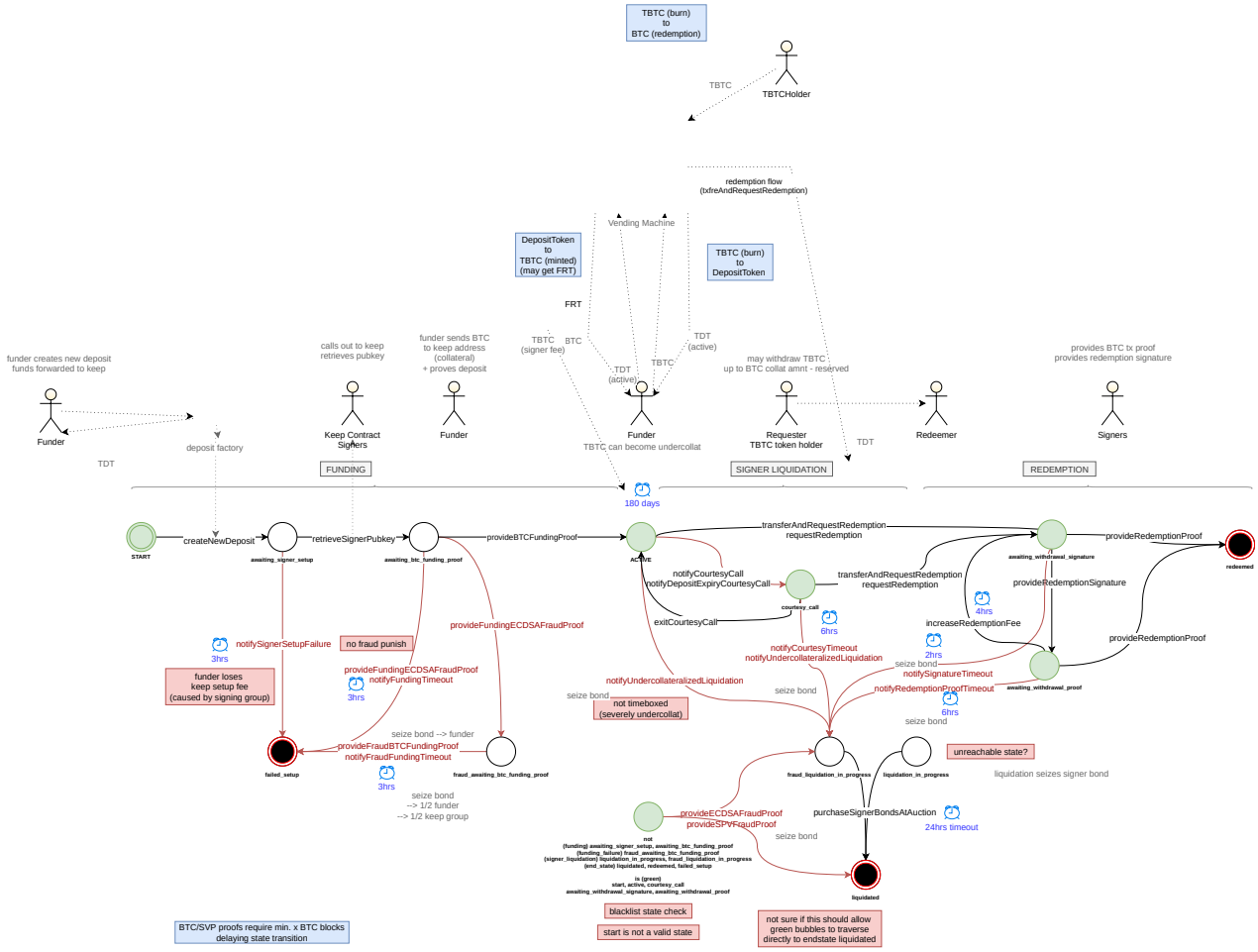
tBTC Deposits can stay active for up to 180 days, with a lot size maximum of 1 BTC. During a this active term, signers are required to lock away approximately 150% of the backing BTC's value in ETH. Should the system experience higher-than-expected creation of Deposits, the amount of collateral available to be locked up may be depleted and deposits will not be able to be opened until others are redeemed.

4.4 Dependency - `bitcoin-spv`

The contract consuming the libraries functionality is expected to provide well-formed data that was verified by BTC nodes and is included in a block. The SPV verification itself

involves complex security assumptions that are out of scope for the library itself. The security and trust model needs to be established with the consuming contract.

4.5 Overview - tBTC



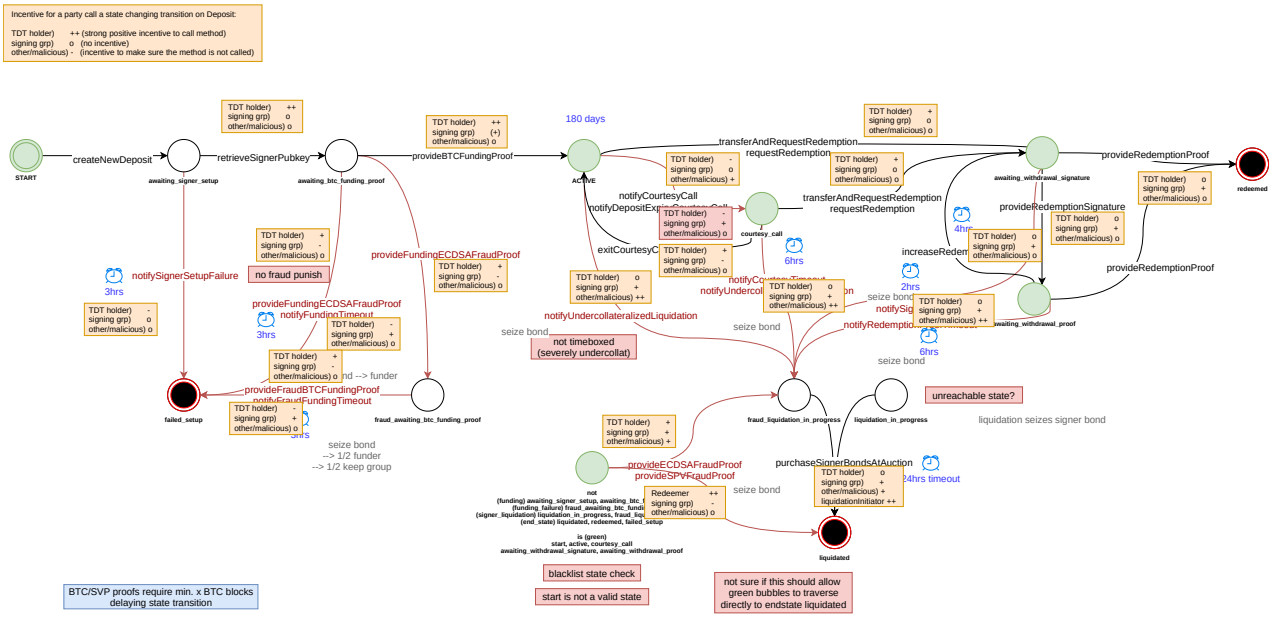
tBTC Deposit States and Transitions

Actors

The relevant actors are as follows:

- **Funder**
- **TBTCDepositTokenHolder**
- **SigningGroup**
- **VendingMachine**
- **Other Accounts**

tBTC Deposit Flow state transition incentives



tBTC Deposit States and Transition Incentives

This section analyzes the incentives of various actors in the system to interact and spend gas on causing a state transition for a certain Deposit.

Part of the security in the deposit flow relies on the fact that someone initiates state transitions (success path, timeout, erroneous or fraudulent behavior) as they become available. For example, if the signing group fails to form in time (3 hrs) someone is supposed to call `notifySignerSetupFailure` on the deposit to move it to the `failed_setup` end-state. On the other hand, if the signer setup succeeds, the `funder` is incentivized to push the deposit to the next state (`awaiting_btc_funding_proof`) and proceed to send `BTC` collateral to the address maintained by the signing group.

There are various incentives for parties in the system to invest gas in moving a deposit to another state as the transition becomes available, however, there are also incentives for not causing a transition.

Transitions

Funding: notifySignerSetupFailure

While the `TDT` holders main objective for the funding flow is to push the deposit to the `active` state as quickly as possible to redeem `TBTC` for `TDT` and earn `FRT` , there is no incentive for the `TDT` holder to call out missing timed milestones for her deposit. In one case the funder is even punished for the signing group failing to provide a valid pubKey in time leading to the funder losing the initial payment to the keep.

- **TDT Holder:** counter incentive. may want to avoid losing the initial payment to keep hoping the signing group returns a pubKey after the timeout passed.
- **Signing Group:** no incentive to spend gas.
- **Others/Malicious:** no incentive to spend gas.

Unless someone (an automatism) spends gas on terminating the deposit there is a good chance it may stay in this state even after the timeout passed.

Funding: retrieveSignerPubkey

There's a strong incentive for the `TDT` holder to move forward being able to deposit BTC in the signer group controlled address. The signer group may provide keys to the keep contract while they have only little incentive (signer fee) to spend gas on calling `retrieveSignerPubkey`. The funder might end up having to call the method to proceed to the next stage.

- **TDT Holder:** incentive to push the deposit to be active.
- **Signing Group:** no incentive to spend gas. incentive for TDT is higher.
- **Others/Malicious:** no incentive to spend gas.

Funding: notifyFundingTimeout

funding timeout passed.

- **TDT Holder:** counter incentive. may want to avoid losing the initial payment to keep or potential punishment for not funding in time.
- **Signing Group:** honest signing group: no incentive to spend gas. Unhonest signing group: might want to front-run an attempt of TDT holder to call out `provideFundingECDSAFraudProof` by terminating the deposit with `notifyFundingTimeout`.
- **Others/Malicious:** no incentive to spend gas.

Funding: provideFundingECDSAFraudProof after funding timeout passed

Someone reports fraud of the signing group after the TDT holder fails to provide collateral in time. The signing group is not punished for the fraud. However, the code assumes punishment for the funder to not fund and not report in time. Additionally, if the timeout passes and the funder reports fraud but provided BTC collateral and was not able to call `provideBTCFundingProof` to proceed to the next stage (Note: funding proof can only be

called with a delay as it requires sufficient accumulated difficulty in the header chain - x times BTC block time) the funder may lose both, the BTC collateral and can get punished for not providing a proof in time.

- **TDT Holder:** incentive to report fraud. However, TDT holder is punished as well as the deposit is terminated. counter-incentive to actually call out the fraud after timeout passed as the current code assumes punishment of the funder. The funder is incentivized to call `notifyFundingTimeout` instead which does not explicitly punish any party.
- **Signing Group:** counter incentive. might want to front-run an attempt of TDT holder to call out `provideFundingECDSAFraudProof` by terminating the deposit with `notifyFundingTimeout`. However, the signing group is not punished for potentially committing fraud. This might actually allow the signing group to steal BTC collateral without being punished if the TDT holder is late enough for not being able to successfully call `provideBTCFundingProof` before the `notifyFundingTimeout`.
- **Others/Malicious:** no incentive to spend gas.

Note: `provideBTCFundingProof` cannot be called immediately after entering `waiting_for_btc_funding_proof` as it is implicitly delayed because it requires accumulated work (x times BTC block time). In order to not lose funds, the TDT holder must ensure that `provideBTCFundingProof` is called before `notifyFundingTimeout` passes.

Note: This path does not emit `logSetupFailed`.

Funding: provideFundingECDSAFraudProof

The signing party committed fraud. Someone calls out the fraud in before

- **TDT Holder:** incentive to report fraud to be compensated with the signer bond to make up for the potentially lost funds. Ideally is able to recover the complete signer bond when being able to provide BTC funding proof, or otherwise recovers half of the funds.
- **Signing Group:** counter incentive. incentive to call `provideBTCFundingProof` in case the TDT holder actually funded the transaction to avoid `provideFundingECDSAFraudProof`. Furthermore, they can try to call `provideECDSAFraudProof` to also seize the signer bond before anyone else does.
- **Others/Malicious:** no incentive to spend gas. Not awarded any funds for reporting fraud.

Note: The ideal time for signers to commit fraud is right at the time the transition to `provideBTCFundingProof` becomes available. This way they avoid that the funder gets exclusive rights to get the signer bond awarded and they can try to report fraud themselves.

Funding: notifyFraudFundingTimeout

The signing party committed fraud. The TDT holder did not prove BTC funding in time.

- **TDT Holder:** counter incentive to call out the timeout. signer bond is awarded 50% to funder, 50% to signer group.
- **Signing Group:** incentive to call this transition before TDT holder calls `provideFraudBTCFundingProof` to at least recover half of the signer bond.
- **Others/Malicious:** no incentive to spend gas. Not awarded any funds for reporting fraud.

Note: Even though committing fraud the signer group receives half of the deposit. TDT holder is only partially compensated, losing funds if they provided BTC without being able to prove it (timeout) and potentially winning funds (depending on the keep payment) if they did not transfer BTC.

Note: Potential reward for a group of signers stealing the BTC collateral and calling out the timeout before TDT holder does is BTC collateral + 50% signer bond.

Funding: provideFraudBTCFundingProof

The signing party committed fraud. The TDT holder provided proof of transferring at minimum `lotSizeSatoshis` BTC to the signer group address.

- **TDT Holder:** incentive to get the complete signer bond awarded to cover the losses from the BTC transfer.
- **Signing Group:** counter incentive. Loses signer bond. favors `notifyFraudFundingTimeout`.
- **Others/Malicious:** no incentive to spend gas. Not awarded any funds for reporting fraud.

Note: The BTC funding proof requires accumulated work to pass (x times BTC block time). TDT holder must ensure to call this method before `notifyFraudFundingTimeout` passes or otherwise signer group could at least recover half of the signer bond.

Note: `provideFraudBTCFundingProof` does not verify the block timestamp of the BTC transaction. Funding can also be provided after ECDSA fraud was called to receive the full signer bond. This does not make a lot of sense as the TDT holder does not profit from this scenario unless she also colludes in the signing group and initially committed fraud.

Funding: provideBTCFundingProof

Deposit funding with collateral was proven. Deposit is in active state.

- **TDT Holder:** strong incentive to move to active state to redeem tBTC for TDT.
- **Signing Group:** incentive to get deposit to active state (signer fee). incentive to commit fraud after BTC deposit was made (backoff time for submission depending on configured accumulated difficulty) and report fraud on the active deposit to be set as `liquidationInitiator` which would not be possible in the funding flow.
- **Others/Malicious:** no incentive to spend gas.

Note: Can be called even if funding timeout passed but not called out.

Active: notifyCourtesyCall

Deposit is undercollateralized.

- **TDT Holder:** counter incentive to call out under-collateralization for own deposit.
- **Signing Group:** no incentive to spend gas.
- **Others/Malicious:** no incentive to spend gas other than security the system.

Note: Can be called even if the deposit term is reached.

Note: Undercollateralization can be due to oracle price slippage. (Oracle Risk)

Active: notifyDepositExpiryCourtesyCall

Deposit is reaching end-of-term.

- **TDT Holder:** counter incentive to call out under-collateralization for own deposit.
- **Signing Group:** no incentive to spend gas.
- **Others/Malicious:** no incentive to spend gas other than security the system.

Note: Can be called even if the deposit term is reached.

Note: This transition should be removed.

Active: exitCourtesyCall

Exit from courtesy call if deposit term is not yet reached.

- **TDT Holder:** incentive to set deposit to active.
- **Signing Group:** no incentive to spend gas.
- **Others/Malicious:** no incentive to spend gas.

Note: Courtesy call a can be exit in the same block if someone calls `notifyDepositExpiryCourtesyCall` and `_d.fundedAt + TBTCConstants.getDepositTerm() == block.timestamp`.

Active: notifyUndercollateralizedLiquidation, notifyCourtesyTimeout, notifySignatureTimeout, notifyRedemptionProofTimeout

Liquidate the deposit due to it being severely undercollateralized.

- **TDT Holder:** no incentive to spend gas.
- **Signing Group:** incentive to set themselves as liquidationInitiator and recover the bond at the auction. Allows one member of the signing group to purchase the bond and the signing group may receive half of the remainder after the auction to the group.
- **Others/Malicious:** gets rewarded for calling out undercollateralized deposits (`liquidationInitiator`).

Note: Calling out undercollateralized deposits can be front-run.

Note: Undercollateralization can be due to oracle price slippage. (Oracle Risk)

Active: provideECDSAFraudProof, provideSPVFraudProof

Provide proof of signer fraud for an active deposit.

- **TDT Holder:** incentive to report fraud to be rewarded as `liquidationInitiator`.
- **Signing Group:** counter incentive to call out fraud on themselves and incentive to report fraud to be rewarded as `liquidationInitiator` and recover part of the bond.
- **Others/Malicious:** incentive to report fraud to be rewarded as `liquidationInitiator`.

Redemption: provideECDSAFraudProof, provideSPVFraudProof

Provide proof of signer fraud in the redemption flow.

- **Redeemer:** Can be set to any address when requesting redemption. Incentive to call the transition in order to receive the full signer bond.
- **Signing Group:** counter incentive to call out fraud on themselves.
- **Others/Malicious:** no incentive to spend gas for not being rewarded.

Liquidation: purchaseSignerBondsAtAuction

Anyone can purchase the signer bond for a deposit in liquidation. The auction is settled in TBTC. The party purchasing the bond receives 90-100% of the seized bond depending on how long the auction is active already.

The longer the auction is active the more percent of the bond is awarded to the buyer. There is an incentive for the buyer to wait until the end of the auction to receive all of the signer bond. The auction can be front-run by observing that someone places a bid. The bids price is static `lotSizeTbtc`.

Only the leftover contract balance (which can be zero at this time if the bidder waited until the end of the auction period) the liquidationInitiator (someone calling out fraud or undercollateralized deposits or timeouts) is compensated. - In the case of fraud the `liquidationInitiator` gets all the leftover contract balance - In the case of a timeout or undercollateralized event the leftover contract balance is split between the signing group and the one calling out the event

- **TDT Holder:** weak incentive to purchase bonds to make sure deposit is compensated with TBTC.
- **Signing Group:**
  - Fraud: incentivized to bid at the latest time possible to maximize the reward and avoid compensating the party calling out fraud.
  - Abort: incentivized to bid at the latest time possible to maximize the reward and avoid compensating the party calling out the abort with more than half of the remainder after the auction value.
- **Others/Malicious:**
  - incentivized to bid at the latest time possible to maximize the reward and avoid compensating the party calling out fraud/abort.
- **LiquidationInitiator:**
  - Fraud: is incentivized to maximize the reward by purchasing the bond at the earliest time possible.

- Abort: is incentivized to maximize the reward by purchasing the bond at the latest time possible.

Note: Bidding on the auction can be front-run to maximize rewards by bidding at the latest time possible.

Note: Can be front-run: observing if someone purchases the signer bond and then front-run it if lucrative.

Active: transferAndRequestRedemption, requestRedemption

Transfer TDT token ownership to a new recipient, request signer group to sign wpkhSpendSighash to initiate redemption. Minimum redemption fee is set and can only be adjusted to max. 5 times the initial redemption fee in cycles every 4 hrs with `increaseRedemptionFee`).

- **TDT Holder:** The method is supposed to be called by the current TDT holder (or VendingMachine).
- **Redeemer:** no incentive.
- **Signing Group:** no incentive.
- **Others/Malicious:** no incentive.

Note: Can be called after the deposit term is reached to close the deposit. However, the deposit might still fall severely undercollateralized while waiting in the redemption flow (cycling with `increaseRedemptionFee` for at most 5 * 4 hours) and it will not be possible to call that out.

Redemption: provideRedemptionSignature

Signers provide the signature for the most recent wpkhSpendSighash digest.

- **TDT Holder:** no incentive.
- **Redeemer:** no incentive.
- **Signing Group:** provides signature to continue redemption flow and be awarded the signer fee.
- **Others/Malicious:** no incentive.

Note: Bails if signature for different digest ist provided.

Redemption: provideRedemptionProof

Anyone can provide proof that the BTC transaction was sent terminating the deposit.

- **TDT Holder:** no incentive to spend gas.
- **Redeemer:** no incentive to spend gas. This will reward the signers (and fee rebate to the former depositor).
- **Signing Group:** incentive to receive the signer fee.
- **Others/Malicious:** no incentive to spend gas.

Note: Does not explicitly verify tx signature. accepts previously signed transactions after increasing the fee.

Redemption: IncreaseRedemptionFee

Signers can increase the redemption fee to cover BTC transaction costs.

- **TDT Holder:** no incentive to spend gas.
- **Redeemer:** no incentive to spend gas.
- **Signing Group:** incentive to adjust fee.
- **Others/Malicious:** no incentive to spend gas.

Note: If BTC network stays congested this might always require a few cycles of `provideRedemptionSignature` and `increaseRedemptionFee` being called every 4 hrs.

Note: Can only be increased every 4 hrs to x times initial fee chosen by redeemer.

Note: Fee can be increased up to 5 * initialRedemptionFee. initialRedemptionFee can be set when requesting redemption (must be >= system minimum). There is no incentive for TDT holder or others to not increase the fee. Fee is paid from the tBTC owned by the contract.

Note: Leftover tBTC assigned to the deposit contract that is >= signerFee is sent to the rebateTokenHolder for the deposit. Values < signerFee are lost?

Note: In `awaiting_withdrawal_proof` a signed btc tx can be constructed to actually redeem the deposit. Assuming someone sends the transaction redeeming BTC late and `increaseRedemptionFee` becomes available before being able to `provideRedemptionProof` (delayed due to required accumulated work), someone could increase the fee after the redemption has been made. `provideRedemptionProof` can then afterward still be called from `awaiting_withdrawal_proof`.

Note: When increasing the signer fee someone could attempt to just feed in a btc transaction with the lowest signer fee as all of the signatures for the different amounts are valid.

5 Issues

Each issue has an assigned severity:

- **Minor** issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- **Medium** issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- **Major** issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- **Critical** issues are directly exploitable security vulnerabilities that need to be fixed.

5.1 `TokenStaking.recoverStake` allows instant stake undelegation

Critical **✓ Addressed**

Resolution
Addressed with keep-network/keep-core#1521 by adding a non-zero check for the undelegation block.

Description

`TokenStaking.recoverStake` is used to recover stake that has been designated to be undelegated. It contains a single check to ensure that the undelegation period has passed:

keep-core/contracts/solidity/contracts/TokenStaking.sol:L182-L187

```
function recoverStake(address _operator) public {
    uint256 operatorParams = operators[_operator].packedParams;
    require(
```

```
        block.number > operatorParams.getUndelegationBlock().add(undelegationP
        "Can not recover stake before undelegation period is over."
    );
```

However, if an undelegation period is never set, this will always return true, allowing any operator to instantly undelegate stake at any time.

Recommendation

Require that the undelegation period is nonzero before allowing an operator to recover stake.

5.2 Improper length validation in BLS signature library allows RNG manipulation Critical ✓ Addressed

Resolution
Addressed with keep-network/keep-core#1523 by adding input length checks to `g2Decompress`, `g2Unmarshal` and `g1Unmarshal`.

Description

`KeepRandomBeaconOperator.relayEntry(bytes memory _signature)` is used to submit random beacon results:

keep-core/contracts/solidity/contracts/KeepRandomBeaconOperator.sol:L418-L433

```
function relayEntry(bytes memory _groupSignature) public nonReentrant {
    require(isEntryInProgress(), "Entry was submitted");
    require(!hasEntryTimedOut(), "Entry timed out");

    bytes memory groupPubKey = groups.getGroupPublicKey(signingRequest.groupIn

    require(
        BLS.verify(
            groupPubKey,
            signingRequest.previousEntry,
```

```
            _groupSignature
        ),
        "Invalid signature"
    );

    emit RelayEntrySubmitted();
```

The function calls `BLS.verify`, which validates that the submitted signature correctly signs the previous recorded random beacon entry. `BLS.verify` calls `AltBn128.g1Unmarshal(signature)`:

keep-core/contracts/solidity/contracts/cryptography/BLS.sol:L31-L37

```
function verify(
    bytes memory publicKey,
    bytes memory message,
    bytes memory signature
) public view returns (bool) {

    AltBn128.G1Point memory _signature = AltBn128.g1Unmarshal(signature);
```

`AltBn128.g1Unmarshal(signature)` reads directly from memory without making any length checks:

keep-core/contracts/solidity/contracts/cryptography/AltBn128.sol:L214-L228

```
/**
 * @dev Unmarshals a point on G1 from bytes in an uncompressed form.
 */
function g1Unmarshal(bytes memory m) internal pure returns(G1Point memory) {
    bytes32 x;
    bytes32 y;

    /* solium-disable-next-line */
    assembly {
        x := mload(add(m, 0x20))
        y := mload(add(m, 0x40))
    }
```

```
    return G1Point(uint256(x), uint256(y));
}
```

There are two potential issues with this:

1. `g1Unmarshal` may be reading out-of-bounds of the signature from dirty memory.
2. `g1Unmarshal` may not be reading all of the signature. If more than 64 bytes are supplied, they are ignored for the purposes of signature validation.

These issues are important because the hash of the signature is the "random number" supplied to user contracts:

keep-core/contracts/solidity/contracts/KeepRandomBeaconOperator.sol:L435-L448

```
// Spend no more than groupSelectionGasEstimate + 40000 gas max
// This will prevent relayEntry failure in case the service contract is compr
signingRequest.serviceContract.call.gas(groupSelectionGasEstimate.add(40000))(
    abi.encodeWithSignature(
        "entryCreated(uint256,bytes,address)",
        signingRequest.relayRequestId,
        _groupSignature,
        msg.sender
    )
);

if (signingRequest.callbackFee > 0) {
    executeCallback(signingRequest, uint256(keccak256(_groupSignature)));
}
```

An attacker can use this behavior to game random number generation by frontrunning a valid signature submission with additional byte padding.

Recommendation

Ensure each function in `BLS.sol` properly validates input lengths for all parameters; the same length validation issue exists in `BLS.verifyBytes`.

5.3 tbtc - the tecdsa keep is never closed, signer bonds are not released Critical ✓ Addressed

Resolution

Addressed with https://github.com/keep-network/tbtc/issues/473, https://github.com/keep-network/tbtc/issues/490, keep-network/tbtc#534, and keep-network/tbtc#520.

- failed_setup:
  - notifySignerSetupFailure ✔ closed by seizing funds with https://github.com/Consensys/thesis-tbtc-audit-2020-01/issues/520
  - notifyFundingTimeout ✔ closed with keep-network/tbtc#534
  - provideFundingECDSAFraudProof, ✔ slashes stake, distributes signer bonds to funder (push payment -> should be pull or funder may block), closes keep.
  - provideFraudBTCFundingProof ✔ removed with keep-network/tbtc#534
  - notifyFraudFundingTimeout ✔ removed with keep-network/tbtc#534
- liquidated:
  - provideSPVFraudProof ✔ removed
  - purchaseSignerBondsAtAuction ✔ via startSignerAbortLiquidation, ✔ via startSignerFraudLiquidation (implicitly via seizebonds)
- redeemed:
  - provideRedemptionProof ✔

Description

At the end of the TBTC deposit lifecycle happy path, the deposit is supposed to close the keep in order to release the signer bonds. However, there is no call to `closeKeep` in any of the code-bases under audit.

Recommendation

Close the keep releasing the signer bonds.

5.4 tbtc - No access control in `TBTCSystem.requestNewKeep` Critical ✓ Addressed

Resolution
Issue addressed in keep-network/tbtc#514. Each call to `requestNewKeep` makes a check that `uint(msg.sender)` is an existing `TBTCDepositToken`. Because these tokens are only minted in `DepositFactory`, `msg.sender` would have to be one of the cloned deposit contracts.

Description

`TBTCSystem.requestNewKeep` is used by each new `Deposit` contract on creation. It calls `BondedECDSAKeepFactory.openKeep`, which sets the `Deposit` contract as the "owner," a permissioned role within the created keep. `openKeep` also automatically allocates bonds from members registered to the application. The "application" from which member bonds are allocated is the tbtc system itself.

Because `requestNewKeep` has no access controls, anyone can request that a keep be opened with `msg.sender` as the "owner," and arbitrary signing threshold values:

tbtc/implementation/contracts/system/TBTCSystem.sol:L231-L243

```
/// @notice Request a new keep opening.
/// @param _m Minimum number of honest keep members required to sign.
/// @param _n Number of members in the keep.
/// @return Address of a new keep.
function requestNewKeep(uint256 _m, uint256 _n, uint256 _bond)
    external
    payable
    returns (address)
{
    IBondedECDSAKeepVendor _keepVendor = IBondedECDSAKeepVendor(keepVendor);
    IBondedECDSAKeepFactory _keepFactory = IBondedECDSAKeepFactory(_keepVendor
    return _keepFactory.openKeep.value(msg.value)(_n, _m, msg.sender, _bond);
}
```

Given that the owner of a keep is able to seize signer bonds, close the keep, and more, having control of this role could be detrimental to group members.

Recommendation

Add access control to `requestNewKeep`, so that it can only be called as a part of the `Deposit` creation and initialization process.

5.5 Unpredictable behavior due to front running or general bad timing Major ✓ Addressed

Resolution

This issue has been addressed with https://github.com/keep-network/tbtc/issues/493 and the following set of PRs:

- https://github.com/keep-network/tbtc/issues/493
- https://github.com/keep-network/keep-tecdsa/issues/296 - note: `initializeImplementation` should be done in `completeUpgrade` otherwise this could be used as a backdoor.
  - fixed by keep-network/keep-ecdsa#327 - fixed: initialization moved to complete upgrade step
- https://github.com/keep-network/keep-core/issues/1423 - note: initializeImplementation `should be done in` completeUpgrade` otherwise this could be used as a backdoor.
  - fixed by keep-network/keep-core#1517 - fixed: initialization moved to complete upgrade step

The client also provided the following statements:

> *In general, our current stance on frontrunning proofs that lead to rewards is that as long as it doesn't significantly compromise an incentive on the primary actors of the system, we're comfortable with having it present. In particular, frontrunnable actions that include rewards in several cases have additional incentives—for tBTC deposit owners, for example, claiming bonds in case of misbehavior; for signers, reclaiming bonds in case of deposit owner absence or other misbehavior. We consider signer reclamation of bonds to be a strong*

> *incentive, as bond value is expected to be large enough that there is ongoing expected value to having the bond value liquid rather than bonded.*
>
> *Some of the frontrunning cases (e.g. around beacon signing) did not have this additional incentive, and in those cases we've taken up the recommendations in the audit.*

Description

In a number of cases, administrators of contracts can update or upgrade things in the system without warning. This has the potential to violate a security goal of the system.

Specifically, privileged roles could use front running to make malicious changes just ahead of incoming transactions, or purely accidental negative effects could occur due to unfortunate timing of changes.

Some instances of this are more important than others, but in general users of the system should have assurances about the behavior of the action they're about to take.

Examples

System Parameters

The owner of the `TBTCSystem` contract can change system parameters at any time with changes taking effect immediately.

- `setSignerFeeDivisor` - stored in the deposit contract when creating a new deposit. emits an event.
- `setLotSizes` - stored in the deposit contract when creating a new deposit. emits an event.
- `setCollateralizationThresholds` - stored in the deposit contract when creating a new deposit. emits an event.

This also opens up an opportunity for malicious owner to:

- interfere with other participants deposit creation attempts (front-running transactions)
- craft a series of transactions that allow the owner to set parameters that are more beneficial to them, then create a deposit and reset the parameters to the systems' initial settings.

tbtc/implementation/contracts/system/TBTCSystem.sol:L113-L121

```
/// @notice Set the system signer fee divisor.
/// @param _signerFeeDivisor The signer fee divisor.
function setSignerFeeDivisor(uint256 _signerFeeDivisor)
    external onlyOwner
{
    require(_signerFeeDivisor > 9, "Signer fee divisor must be greater than 9,
    signerFeeDivisor = _signerFeeDivisor;
    emit SignerFeeDivisorUpdated(_signerFeeDivisor);
}
```

Upgradables

The proxy pattern used in many places throughout the system allows the operator to set a new implementation which takes effect immediately.

keep-core/contracts/solidity/contracts/KeepRandomBeaconService.sol:L67-L80

```
/**
 * @dev Upgrade current implementation.
 * @param _implementation Address of the new implementation contract.
 */
function upgradeTo(address _implementation)
    public
    onlyOwner
{
    address currentImplementation = implementation();
    require(_implementation != address(0), "Implementation address can't be ze
    require(_implementation != currentImplementation, "Implementation address
    setImplementation(_implementation);
    emit Upgraded(_implementation);
}
```

keep-tecdsa/solidity/contracts/BondedECDSAKeepVendor.sol:L57-L71

```
/// @notice Upgrades the current vendor implementation.
/// @param _implementation Address of the new vendor implementation contract.
```

```
function upgradeTo(address _implementation) public onlyOwner {
    address currentImplementation = implementation();
    require(
        _implementation != address(0),
        "Implementation address can't be zero."
    );
    require(
        _implementation != currentImplementation,
        "Implementation address must be different from the current one."
    );
    setImplementation(_implementation);
    emit Upgraded(_implementation);
}
```

Registry

keep-tecdsa/solidity/contracts/BondedECDSAKeepVendorImplV1.sol:L43-L50

```
function registerFactory(address payable _factory) external onlyOperatorContra
    require(_factory != address(0), "Incorrect factory address");
    require(
        registry.isApprovedOperatorContract(_factory),
        "Factory contract is not approved"
    );
    keepFactory = _factory;
}
```

Recommendation

The underlying issue is that users of the system can't be sure what the behavior of a function call will be, and this is because the behavior can change at any time.

We recommend giving the user advance notice of changes with a time lock. For example, make all upgrades require two steps with a mandatory time window between them. The first step merely broadcasts to users that a particular change is coming, and the second step commits that change after a suitable waiting period.

5.6 keep-core - reportRelayEntryTimeout creates an incentive for nodes to race for rewards potentially wasting gas and it creates an opportunity for front-running Major ✓ Addressed

Resolution
Following the discussion at https://github.com/keep-network/keep-core/issues/1404 it was verified that the method throws as early as possible in an attempt to safe gas in case many nodes call out the timeout in the same block. The client is currently comfortable with this tradeoff. We would like to note that this issue cannot easily be addressed (e.g. allowing nodes to disable calling out timeouts impacts the security of the system; a commit/reveal proxy adds overhead and is unlikely to make the situation better as nodes are programmed to call out timeouts) and we therefore recommend to monitor the network for this scenario.

Description

The incentive on `reportRelayEntryTimeout` for being rewarded with 5% of the seized amount creates an incentive to call the method but might also kick off a race for front-running this call. This method is being called from the keep node which is unlikely to adjust the gasPrice and might always lose the race against a front-running bot collecting rewards for all timeouts and fraud proofs (https://github.com/Consensys/thesis-tbtc-audit-2020-01/issues/41)

Examples

keep-core/contracts/solidity/contracts/KeepRandomBeaconOperator.sol:L600-L626

```
/**
 * @dev Function used to inform about the fact the currently ongoing
 * new relay entry generation operation timed out. As a result, the group
 * which was supposed to produce a new relay entry is immediately
 * terminated and a new group is selected to produce a new relay entry.
 * All members of the group are punished by seizing minimum stake of
 * their tokens. The submitter of the transaction is rewarded with a
 * tattletale reward which is limited to min(1, 20 / group_size) of the
 * maximum tattletale reward.
 */
```

```
function reportRelayEntryTimeout() public {
    require(hasEntryTimedOut(), "Entry did not time out");
    groups.reportRelayEntryTimeout(signingRequest.groupIndex, groupSize, minim

    // We could terminate the last active group. If that's the case,
    // do not try to execute signing again because there is no group
    // which can handle it.
    if (numberOfGroups() > 0) {
        signRelayEntry(
            signingRequest.relayRequestId,
            signingRequest.previousEntry,
            signingRequest.serviceContract,
            signingRequest.entryVerificationAndProfitFee,
            signingRequest.callbackFee
        );
    }
}
```

Recommendation

Make sure that `reportRelayEntryTimeout` throws as early as possible if the group was previously terminated (`isGroupTerminated`) to avoid that keep-nodes spend gas on a call that will fail. Depending on the reward for calling out the timeout this might create a front-running opportunity that cannot be resolved.

5.7 keep-core - reportUnauthorizedSigning fraud proof is not bound to reporter and can be front-run Major ✓ Addressed

Resolution
Addressed with https://github.com/keep-network/keep-core/issues/1405 by binding the proof to `msg.sender` .

Description

An attacker can monitor `reportUnauthorizedSigning()` for fraud reports and attempt to front-run the original call in an effort to be the first one reporting the fraud and be rewarded

5% of the total seized amount.

Examples

keep-core/contracts/solidity/contracts/KeepRandomBeaconOperator.sol:L742-L755

```
/**
 * @dev Reports unauthorized signing for the provided group. Must provide
 * a valid signature of the group address as a message. Successful signature
 * verification means the private key has been leaked and all group members
 * should be punished by seizing their tokens. The submitter of this proof is
 * rewarded with 5% of the total seized amount scaled by the reward adjustment
 * parameter and the rest 95% is burned.
 */
function reportUnauthorizedSigning(
    uint256 groupIndex,
    bytes memory signedGroupPubKey
) public {
    groups.reportUnauthorizedSigning(groupIndex, signedGroupPubKey, minimumSta
}
```

Recommendation

Require the reporter to include `msg.sender` in the signature proving the fraud or implement a two-step commit/reveal scheme to counter front-running opportunities by forcing a reporter to secretly commit the fraud parameters in one block and reveal them in another.

5.8 keep-core - operator contracts disabled via panic button can be re-enabled by RegistryKeeper Major ✓ Addressed

Resolution

Addressed by https://github.com/keep-network/keep-core/issues/1406 with changes from https://github.com/keep-network/keep-core/pull/1463:

- the contract is now using enums instead of int literals
- only new operator contracts can be approved

- only approved contracts can be disabled
- disabled contracts cannot be re-enabled
- disabling an operator contract does not yield an event
- changes take effect immediately

Description

The Registry contract defines three administrative accounts: `Governance`, `registryKeeper`, and `panicButton`. All permissions are initially assigned to the deployer when the contract is created. The account acting like a super-admin, being allowed to re-assign administrative accounts - is `Governance`. `registryKeeper` is a lower privileged account maintaining the registry and `panicButton` is an emergency account that can disable operator contracts.

The keep specification states the following:

> *Panic Button The Panic Button can disable malicious or malfunctioning contracts that have been previously approved by the Registry Keeper. When a contract is disabled by the Panic Button, its status on the registry changes to reflect this, and it becomes ineligible to penalize operators. Contracts disabled by the Panic Button can not be reactivated. The Panic Button can be rekeyed by Governance.*

It is assumed that the permissions are `Governance` > `panicButton` > `registryKeeper`, meaning that `panicButton` should be able to overrule `registryKeeper`, while `registryKeeper` cannot overrule `panicButton`.

With the current implementation of the Registry the `registryKeeper` account can re-enable an operator contract that has previously been disabled by the `panicButton` account.

We would also like to note the following:

- The contract should use enums instead of integer literals when working with contract states.
- Changes to the contract take effect immediately, allowing an administrative account to selectively front-run calls to the Registry ACL and interfere with user activity.
- The operator contract state can be set to the current value without raising an error.
- The panic button can be called for operator contracts that are not yet active.

Examples

keep-core/contracts/solidity/contracts/Registry.sol:L67-L75

```
function approveOperatorContract(address operatorContract) public onlyRegistry
    operatorContracts[operatorContract] = 1;
}

function disableOperatorContract(address operatorContract) public onlyPanicBut
    operatorContracts[operatorContract] = 2;
}
```

Recommendation

The keep specification states:

> *The Panic Button can be used to set the status of an APPROVED contract to DISABLED. Operator Contracts disabled with the Panic Button cannot be re-enabled, and disabled contracts may not punish operators nor be selected by service contracts to perform work.*

All three accounts are typically trusted. We recommend requiring the `Governance` or `paniceButton` accounts to reset the contract operator state before `registryKeeper` can change the state or disallow re-enabling of disabled operator contracts as stated in the specification.

5.9 tbtc - State transitions are not always enforced Major ✓ Addressed

Resolution

This issue was addressed with https://github.com/keep-network/tbtc/issues/494 and accepted by the client with the following statement. Deposits that are timed out can still be pushed to an active state.

> *For 5.7 around state transitions, our stance (specifically for the upcoming release) is that a skipped state is acceptable as long as it does not result in data loss or incentive skew. Taken in turn, the listed examples:*

- *'A TDT holder can choose not to call out notifySignerSetupFailure hoping that the signing group still forms after the signer setup timeout passes.' -> we consider this fine. If the TDT holder wishes to hold out hope, it is their choice. Signers should be incentivized to call* `notifySignerSetupFailure` *in case of actual failure to release their bond.*

- *'The deposit can be pushed to active state even after notifySignerSetupFailure, notifyFundingTimeout have passed but nobody called it out.' -> again, we consider this fine. A deposit that is funded and proven past its timeout is still a valid deposit, since the two players in question (the depositor and the signing group) were willing to wait longer to complete the flow. The timeouts in question are largely a matter of allowing signers to release their bond in case there is an issue setting up the deposit.*

- *'Members of the signing group might decide to call notifyFraudFundingTimeout in a race to avoid late submissions for provideFraudBTCFundingProof to succeed in order to contain funds lost due to fraud.' -> We are intending to change the mechanic here so that signers lose their whole bond in either case.*

- *'A malicious signing group observes BTC funding on the bitcoin chain in an attempt to commit fraud at the time the provideBTCFundingProof transition becomes available to front-run provideFundingECDSAFraudProof forcing the deposit into active state.' -> this one is tough, and we're working on changing the liquidation initiator reward so it is no longer a useful attack. In particular, we're looking at the suggestion in 2.4 for this.*

- *'If oracle price slippage occurs for one block (flash-crash type of event) someone could call an undercollateralization transition.' -> We are still investigating this possibility.*

- *'A deposit term expiration courtesy call can be exit in the rare case where _d.fundedAt + TBTCConstants.getDepositTerm() == block.timestamp' -> Deposit term expiration courtsey calls should no longer apply; see keep-network/tbtc@* `6344892` *. Courtesy call after deposit term is identical to courtsey call pre-term.*

Description

A deposit follows a complex state-machine that makes sure it is correctly funded before `TBTC` Tokens are minted. The deposit lifecycle starts with a set of states modeling a **funding** flow that - if successful - ultimately leads to the deposit being **active**, meaning that corresponding `TBTC` tokens exist for the deposits. A **redemption** flow allows to redeem `TBTC` for `BTC` and a **liquidation** flow handles fraud and abort conditions. Fraud cases in the **funding** flow are handled separately.

State transitions from one deposit state to another require someone calling the corresponding transition method on the deposit and actually spend gas on it. The incentive to call a transition varies and is analyzed in more detail in the **security-specification section** of this report.

This issue assumes that participants are not always pushing forward through the state machine as soon as a new state becomes available, opening up the possibility of having multiple state transitions being a valid option for a deposit (e.g. pushing a deposit to active state even though a timeout should have been called on it).

Examples

A TDT holder can choose not to call out `notifySignerSetupFailure` hoping that the signing group still forms after the signer setup timeout passes.

- there is no incentive for the TDT holder to terminate its own deposit after a timeout.
- the deposit might end up never being in a final error state.
- there is no incentive for the signing group to terminate the deposit.

This affects all states that can time out.

The deposit can be pushed to active state even after `notifySignerSetupFailure`, `notifyFundingTimeout` have passed but nobody called it out.

There is no timeout check in `retrieveSignerPubkey`, `provideBTCFundingProof`.

tbtc/implementation/contracts/deposit/DepositFunding.sol:L108-L117

```
/// @notice                 we poll the Keep contract to retrieve our pubkey
/// @dev                    We store the pubkey as 2 bytestrings, X and Y.
```

```
/// @param  _d          deposit storage pointer
/// @return             True if successful, otherwise revert
function retrieveSignerPubkey(DepositUtils.Deposit storage _d) public {
    require(_d.inAwaitingSignerSetup(), "Not currently awaiting signer setup")

    bytes memory _publicKey = IBondedECDSAKeep(_d.keepAddress).getPublicKey();
    require(_publicKey.length == 64, "public key not set or not 64-bytes long"
```

tbtc/implementation/contracts/deposit/DepositFunding.sol:L263-L278

```
function provideBTCFundingProof(
    DepositUtils.Deposit storage _d,
    bytes4 _txVersion,
    bytes memory _txInputVector,
    bytes memory _txOutputVector,
    bytes4 _txLocktime,
    uint8 _fundingOutputIndex,
    bytes memory _merkleProof,
    uint256 _txIndexInBlock,
    bytes memory _bitcoinHeaders
) public returns (bool) {

    require(_d.inAwaitingBTCFundingProof(), "Not awaiting funding");

    bytes8 _valueBytes;
    bytes memory  _utxoOutpoint;
```

Members of the signing group might decide to call `notifyFraudFundingTimeout` in a race to avoid late submissions for `provideFraudBTCFundingProof` to succeed in order to contain funds lost due to fraud.

It should be noted that even after the fraud funding timeout passed the TDT holder could `provideFraudBTCFundingProof` as it does not check for the timeout.

A malicious signing group observes BTC funding on the bitcoin chain in an attempt to commit fraud at the time the `provideBTCFundingProof` transition becomes available to front-run `provideFundingECDSAFraudProof` forcing the deposit into active state.

- The malicious users of the signing group can then try to report fraud, set themselves as `liquidationInitiator` to be awarded part of the signer bond (in addition to

taking control of the BTC collateral).

- The TDT holders fraud-proof can be front-run, see https://github.com/Consensys/thesis-tbtc-audit-2020-01/issues/12

If oracle price slippage occurs for one block (flash-crash type of event) someone could call an undercollateralization transition.

- For severe oracle errors deposits might be liquidated by calling `notifyUndercollateralizedLiquidation`. The TDT holder cannot exit liquidation in this case.
- For non-severe under collateralization someone could call `notifyCourtesyCall` to impose extra effort on TDT holders to `exitCourtesyCall` deposits.

A deposit term expiration courtesy call can be exit in the rare case where `_d.fundedAt + TBTCConstants.getDepositTerm() == block.timestamp`

tbtc/implementation/contracts/deposit/DepositLiquidation.sol:L289-L298

```
/// @notice     Goes from courtesy call to active
/// @dev        Only callable if collateral is sufficient and the deposit is n
/// @param  _d  deposit storage pointer
function exitCourtesyCall(DepositUtils.Deposit storage _d) public {
    require(_d.inCourtesyCall(), "Not currently in courtesy call");
    require(block.timestamp <= _d.fundedAt + TBTCConstants.getDepositTerm(), "
    require(getCollateralizationPercentage(_d) >= _d.undercollateralizedThresh
    _d.setActive();
    _d.logExitedCourtesyCall();
}
```

tbtc/implementation/contracts/deposit/DepositLiquidation.sol:L318-L327

```
/// @notice     Notifies the contract that its term limit has been reached
/// @dev        This initiates a courtesy call
/// @param  _d  deposit storage pointer
function notifyDepositExpiryCourtesyCall(DepositUtils.Deposit storage _d) publ
    require(_d.inActive(), "Deposit is not active");
    require(block.timestamp >= _d.fundedAt + TBTCConstants.getDepositTerm(), "
    _d.setCourtesyCall();
    _d.logCourtesyCalled();
```

```
    _d.courtesyCallInitiated = block.timestamp;
}
```

Allow exiting the courtesy call only if the deposit is not expired: `block.timestamp < _d.fundedAt + TBTCConstants.getDepositTerm()`

Recommendation

Ensure that there are no competing interests between participants of the system to favor one transition over the other, causing race conditions, front-running opportunities or stale deposits that are not pushed to end-states.

Note: Please find an analysis of incentives to call state transitions in the security section of this document.

5.10 tbtc - Funder loses payment to keep if signing group is not established in time Major Pending

Resolution
This issue was addressed with https://github.com/keep-network/tbtc/issues/495 by refunding the cost of creating a new keep. We recommend using the pull instead of a push payment pattern to avoid that the funder can block the call.

Description

To create a new deposit, the funder has to pay for the creation of a keep. If establishing the keep does not succeed in time, fails or the signing group decides not to return a public key when `retrieveSignerPubkey` is called to transition from `awaiting_signer_setup` to `awaiting_btc_funding_proof` the signer setup fails. After a timeout of 3 hrs, anyone can force the deposit to transition from `awaiting_signer_setup` to `failed_setup` by calling `notifySignerSetupFailure`.

The funder had to provide payment for the keep but the signing group failed to establish. Payment for the keep is not returned even though one could assume that the signing group tried to play unfairly. The signing group might intentionally try to cause this scenario to interfere with the system.

Examples

- `retrieveSignerPubkey` fails if keep provided pubkey is empty or of an unexpected length

tbtc/implementation/contracts/deposit/DepositFunding.sol:L108-L127

```
/// @notice             we poll the Keep contract to retrieve our pubkey
/// @dev                We store the pubkey as 2 bytestrings, X and Y.
/// @param  _d          deposit storage pointer
/// @return             True if successful, otherwise revert
function retrieveSignerPubkey(DepositUtils.Deposit storage _d) public {
    require(_d.inAwaitingSignerSetup(), "Not currently awaiting signer setup"

    bytes memory _publicKey = IBondedECDSAKeep(_d.keepAddress).getPublicKey();
    require(_publicKey.length == 64, "public key not set or not 64-bytes long"

    _d.signingGroupPubkeyX = _publicKey.slice(0, 32).toBytes32();
    _d.signingGroupPubkeyY = _publicKey.slice(32, 32).toBytes32();
    require(_d.signingGroupPubkeyY != bytes32(0) && _d.signingGroupPubkeyX !=
    _d.fundingProofTimerStart = block.timestamp;

    _d.setAwaitingBTCFundingProof();
    _d.logRegisteredPubkey(
        _d.signingGroupPubkeyX,
        _d.signingGroupPubkeyY);
}
```

- `notifySignerSetupFailure` can be called by anyone after a timeout of 3hrs

tbtc/implementation/contracts/deposit/DepositFunding.sol:L93-L106

```
/// @notice     Anyone may notify the contract that signing group setup has t
/// @dev        We rely on the keep system punishes the signers in this case
/// @param  _d  deposit storage pointer
function notifySignerSetupFailure(DepositUtils.Deposit storage _d) public {
    require(_d.inAwaitingSignerSetup(), "Not awaiting setup");
    require(
        block.timestamp > _d.signingGroupRequestedAt + TBTCConstants.getSignir
```

```
            "Signing group formation timeout not yet elapsed"
        );
        _d.setFailedSetup();
        _d.logSetupFailed();

        fundingTeardown(_d);
    }
```

Recommendation

It should be ensured that a keep group always establishes or otherwise the funder is refunded the fee for the keep.

5.11 tbtc - Ethereum block gas limit imposes a fundamental limitation on SPV proofs Major ✓ Addressed

Resolution
SPV fraud proofs were removed in keep-network/tbtc#521. Remember to continue exploring this limitation of the EVM with benchmarking and gas estimates in the tBTC UI.

Description

Several components of the tBTC system rely on SPV proofs to prove the existence of transactions on Bitcoin. Because an SPV proof must provide the entire Bitcoin transaction to the proving smart contract, the Ethereum block gas limit imposes an upper bound on the size of the transaction in question. Although an exact upper bound is subject to several variables, reasonable estimates show that even a moderately-sized Bitcoin transaction may not be able to be successfully validated on Ethereum.

This limitation is significant for two reasons:

1. Depositors may deposit BTC to the signers by way of a legitimate Bitcoin transaction, only to find that this transaction is unable to be verified on Ethereum. Although the depositor in question was not acting maliciously, they may lose their deposit entirely.

2. In case signers collude to spend a depositor's BTC unprompted, the system allows depositors to prove a fraudulent spend occurred by way of SPV fraud proof. Given that signers can easily spend BTC with a transaction that is too large to validate by way of SPV proof, this method of fraud proof is unreliable at best. Deposit owners should instead prove fraud by using an ECDSA fraud proof, which operates on a hash of the signed message.

Recommendation

It's important that prospective depositors are able to guarantee that their deposit transaction will be verified successfully. To that end, efforts should be made to provide a deposit UI that checks whether or not a given transaction will be verified successfully before it is submitted. Several variables can affect transaction verification:

- Current Ethereum block gas limits
- Number of zero-bytes in the Bitcoin transaction in question
- Size of the merkle proof needed to prove the transaction's existence

Given that not all of these can be calculated before the transaction is submitted to the Bitcoin blockchain, calculations should attempt to provide a margin of error for the process. Additionally, users should be well-educated about the process, including how to perform a deposit with relatively low risk.

Understanding the relative limitations of the EVM will help this process significantly. Consider benchmarking the gas cost of verifying Bitcoin transactions of various sizes.

Finally, because SPV fraud proofs can be gamed by colluding signers, they should be removed from the system entirely. Deposit owners should always be directed towards ECDSA fraud proofs, as these require relatively fewer assumptions and stronger guarantees.

5.12 bitcoin-spv - SPV proofs do not support transactions with larger numbers of inputs and outputs Major Pending

Description

There is no explicit restriction on the number of inputs and outputs a Bitcoin transaction can have - as long as the transaction fits into a block. The number of inputs and outputs in a transaction is denoted by a leading "varint" - a variable length integer. In

`BTCUtils.validateVin` and `BTCUtils.validateVout`, the value of this varint is restricted to under `0xFD`, or 253:

bitcoin-spv/solidity/contracts/BTCUtils.sol:L404-L415

```
/// @notice      Checks that the vin passed up is properly formatted
/// @dev         Consider a vin with a valid vout in its scriptsig
/// @param _vin  Raw bytes length-prefixed input vector
/// @return      True if it represents a validly formatted vin
function validateVin(bytes memory _vin) internal pure returns (bool) {
    uint256 _offset = 1;
    uint8 _nIns = uint8(_vin.slice(0, 1)[0]);

    // Not valid if it says there are too many or no inputs
    if (_nIns >= 0xfd || _nIns == 0) {
        return false;
    }
```

Transactions that include more than 252 inputs or outputs will not pass this validation, leading to some legitimate deposits being rejected by the tBTC system.

Examples

The 252-item limit exists in a few forms throughout the system, outside of the aforementioned `BTCUtils.validateVin` and `BTCUtils.validateVout`:

1. `BTCUtils.determineOutputLength`:

bitcoin-spv/solidity/contracts/BTCUtils.sol:L294-L303

```
/// @notice          Determines the length of an output
/// @dev             5 types: WPKH, WSH, PKH, SH, and OP_RETURN
/// @param _output   The output
/// @return          The length indicated by the prefix, error if invalid leng
function determineOutputLength(bytes memory _output) internal pure returns (ui
    uint8 _len = uint8(_output.slice(8, 1)[0]);
    require(_len < 0xfd, "Multi-byte VarInts not supported");

    return _len + 8 + 1; // 8 byte value, 1 byte for _len itself
}
```

1. `DepositUtils.findAndParseFundingOutput`:

tbtc/implementation/contracts/deposit/DepositUtils.sol:L150-L154

```
function findAndParseFundingOutput(
    DepositUtils.Deposit storage _d,
    bytes memory _txOutputVector,
    uint8 _fundingOutputIndex
) public view returns (bytes8) {
```

1. `DepositUtils.validateAndParseFundingSPVProof`:

tbtc/implementation/contracts/deposit/DepositUtils.sol:L181-L191

```
function validateAndParseFundingSPVProof(
    DepositUtils.Deposit storage _d,
    bytes4 _txVersion,
    bytes memory _txInputVector,
    bytes memory _txOutputVector,
    bytes4 _txLocktime,
    uint8 _fundingOutputIndex,
    bytes memory _merkleProof,
    uint256 _txIndexInBlock,
    bytes memory _bitcoinHeaders
) public view returns (bytes8 _valueBytes, bytes memory _utxoOutpoint){
```

1. `DepositFunding.provideFraudBTCFundingProof`:

tbtc/implementation/contracts/deposit/DepositFunding.sol:L213-L223

```
function provideFraudBTCFundingProof(
    DepositUtils.Deposit storage _d,
    bytes4 _txVersion,
    bytes memory _txInputVector,
    bytes memory _txOutputVector,
    bytes4 _txLocktime,
    uint8 _fundingOutputIndex,
```

```
    bytes memory _merkleProof,
    uint256 _txIndexInBlock,
    bytes memory _bitcoinHeaders
) public returns (bool) {
```

1. `DepositFunding.provideBTCFundingProof`:

tbtc/implementation/contracts/deposit/DepositFunding.sol:L263-L273

```
function provideBTCFundingProof(
    DepositUtils.Deposit storage _d,
    bytes4 _txVersion,
    bytes memory _txInputVector,
    bytes memory _txOutputVector,
    bytes4 _txLocktime,
    uint8 _fundingOutputIndex,
    bytes memory _merkleProof,
    uint256 _txIndexInBlock,
    bytes memory _bitcoinHeaders
) public returns (bool) {
```

1. `DepositLiquidation.provideSPVFraudProof`:

tbtc/implementation/contracts/deposit/DepositLiquidation.sol:L150-L160

```
function provideSPVFraudProof(
    DepositUtils.Deposit storage _d,
    bytes4 _txVersion,
    bytes memory _txInputVector,
    bytes memory _txOutputVector,
    bytes4 _txLocktime,
    bytes memory _merkleProof,
    uint256 _txIndexInBlock,
    uint8 _targetInputIndex,
    bytes memory _bitcoinHeaders
) public {
```

Recommendation

Incorporate varint parsing in `BTCUtils.validateVin` and `BTCUtils.validateVout`. Ensure that other components of the system reflect the removal of the 252-item limit.

5.13 bitcoin-spv - multiple integer under-/overflows Major ✓ Addressed

Resolution

This was partially addressed in summa-tx/bitcoin-spv#118, summa-tx/bitcoin-spv#119, and summa-tx/bitcoin-spv#122.

- Summa opted not to fix the underflow in `extractTarget`.
- In summa-tx/bitcoin-spv#118, the `determineOutputLength` overflow was addressed by casting `_len` to a `uint256` before addition.
- In summa-tx/bitcoin-spv#119, the `extractHash` underflow was addressed by returning an empty `bytes` array if the extracted length would cause underflow. Note that an explicit error and transaction revert is favorable in these cases, in order to avoid returning unusable data to the calling function.
- Underflow and overflow in `BytesLib` was addressed in summa-tx/bitcoin-spv#122. Multiple requires were added to the mentioned functions, ensuring memory reads stayed in-bounds for each array. A later change in summa-tx/bitcoin-spv#128 added support for `slice` with a length of 0.

Description

The bitcoin-spv library allows for multiple integer under-/overflows while processing or converting potentially untrusted or user-provided data.

Examples

- `uint8` underflow `uint256(uint8(_e - 3))`

Note: `_header[75]` will throw consuming all gas if out of bounds while the majority of the library usually uses `slice(start, 1)` to handle this more gracefully.

bitcoin-spv/solidity/contracts/BTCUtils.sol:L483-L494

```
/// @dev             Target is a 256 bit number encoded as a 3-byte mantissa a
/// @param _header   The header
/// @return          The target threshold
function extractTarget(bytes memory _header) internal pure returns (uint256) {
    bytes memory _m = _header.slice(72, 3);
    uint8 _e = uint8(_header[75]);
    uint256 _mantissa = bytesToUint(reverseEndianness(_m));
    uint _exponent = _e - 3;

    return _mantissa * (256 ** _exponent);
}
```

- `uint8` overflow `uint256(uint8(_len + 8 + 1))`

Note: might allow a specially crafted output to return an invalid determineOutputLength <= 9.

Note: while type `VarInt` is implemented for inputs, it is not for the output length.

bitcoin-spv/solidity/contracts/BTCUtils.sol:L295-L304

```
/// @dev             5 types: WPKH, WSH, PKH, SH, and OP_RETURN
/// @param _output   The output
/// @return          The length indicated by the prefix, error if invalid leng
function determineOutputLength(bytes memory _output) internal pure returns (ui
    uint8 _len = uint8(_output.slice(8, 1)[0]);
    require(_len < 0xfd, "Multi-byte VarInts not supported");

    return _len + 8 + 1; // 8 byte value, 1 byte for _len itself
}
```

- `uint8` underflow `uint256(uint8(extractOutputScriptLen(_output)[0]) - 2)`

bitcoin-spv/solidity/contracts/BTCUtils.sol:L366-L378

```
/// @dev             Determines type by the length prefix and validates forma
/// @param _output   The output
```

```
/// @return          The hash committed to by the pk_script, or null for erro
function extractHash(bytes memory _output) internal pure returns (bytes memory
    if (uint8(_output.slice(9, 1)[0]) == 0) {
        uint256 _len = uint8(extractOutputScriptLen(_output)[0]) - 2;
        // Check for maliciously formatted witness outputs
        if (uint8(_output.slice(10, 1)[0]) != uint8(_len)) {
            return hex"";
        }
        return _output.slice(11, _len);
    } else {
        bytes32 _tag = _output.keccak256Slice(8, 3);
```

- `BytesLib` input validation multiple start+length overflow

Note: multiple occurrences. should check `start+length > start && bytes.length >= start+length`

bitcoin-spv/solidity/contracts/BytesLib.sol:L246-L248

```
function slice(bytes memory _bytes, uint _start, uint _length) internal  pure
    require(_bytes.length >= (_start + _length), "Slice out of bounds");
```

- `BytesLib` input validation multiple start overflow

bitcoin-spv/solidity/contracts/BytesLib.sol:L280-L281

```
function toUint(bytes memory _bytes, uint _start) internal  pure returns (uint
    require(_bytes.length >= (_start + 32), "Uint conversion out of bounds.");
```

bitcoin-spv/solidity/contracts/BytesLib.sol:L269-L270

```
function toAddress(bytes memory _bytes, uint _start) internal  pure returns (a
    require(_bytes.length >= (_start + 20), "Address conversion out of bounds.
```

bitcoin-spv/solidity/contracts/BytesLib.sol:L246-L248

```
function slice(bytes memory _bytes, uint _start, uint _length) internal  pure
    require(_bytes.length >= (_start + _length), "Slice out of bounds");
```

bitcoin-spv/solidity/contracts/BytesLib.sol:L410-L412

```
function keccak256Slice(bytes memory _bytes, uint _start, uint _length) pure i
    require(_bytes.length >= (_start + _length), "Slice out of bounds");
```

Recommendation

We believe that a general-purpose parsing and verification library for bitcoin payments should be very strict when processing untrusted user input. With strict we mean, that it should rigorously validate provided input data and only proceed with the processing of the data if it is within a safe-to-use range for the method to return valid results. Relying on the caller to provide pre-validate data can be unsafe especially if the caller assumes that proper input validation is performed by the library.

Given the risk profile for this library, we recommend a conservative approach that balances security instead of gas efficiency without relying on certain calls or instructions to throw on invalid input.

For this issue specifically, we recommend proper input validation and explicit type expansion where necessary to prevent values from wrapping or processing data for arguments that are not within a safe-to-use range.

5.14 tbtc - Unreachable state `LIQUIDATION_IN_PROGRESS` Major ✓ Addressed

Resolution
Addressed with https://github.com/keep-network/tbtc/issues/497 with commits from keep-network/tbtc#517 changing all non-fraud transitions to end up in `LIQUIDATION_IN_PROGRESS`.

Description

According to the specification (overview, states, version 2020-02-06), a deposit can be in one of two **liquidation_in_progress** states.

LIQUIDATION_IN_PROGRESS

> *LIQUIDATION_IN_PROGRESS Liquidation due to undercollateralization or an abort has started Automatic (on-chain) liquidation was unsuccessful*

FRAUD_LIQUIDATION_IN_PROGRESS

> *FRAUD_LIQUIDATION_IN_PROGRESS Liquidation due to fraud has started Automatic (on-chain) liquidation was unsuccessful*

However, `LIQUIDATION_IN_PROGRESS` is unreachable and instead, `FRAUD_LIQUIDATION_IN_PROGRESS` is always called. This means that all non-fraud state transitions end up in the fraud liquidation path and will perform actions as if fraud was detected even though it might be caused by an undercollateralized notification or courtesy timeout.

Examples

- `startSignerAbortLiquidation` transitions to `FRAUD_LIQUIDATION_IN_PROGRESS` on non-fraud events `notifyUndercollateralizedLiquidation` and `notifyCourtesyTimeout`

tbtc/implementation/contracts/deposit/DepositLiquidation.sol:L96-L108

```
/// @notice         Starts signer liquidation due to abort or undercollateral
/// @dev            We first attempt to liquidate on chain, then by auction
/// @param  _d      deposit storage pointer
function startSignerAbortLiquidation(DepositUtils.Deposit storage _d) internal
    _d.logStartedLiquidation(false);
    // Reclaim used state for gas savings
    _d.redemptionTeardown();
    _d.seizeSignerBonds();

    _d.liquidationInitiated = block.timestamp;  // Store the timestamp for auc
    _d.liquidationInitiator = msg.sender;
    _d.setFraudLiquidationInProgress();
}
```

Recommendation

Verify state transitions and either remove `LIQUIDATION_IN_PROGRESS` if it is redundant or fix the state transitions for non-fraud liquidations.

Note that Deposit states can be simplified by removing redundant states by setting a flag (e.g. fraudLiquidation) in the deposit instead of adding a state to track the fraud liquidation path.

According to the specification, we assume the following state transitions are desired:

`LIQUIDATION_IN_PROGRESS` > In case of liquidation due to undercollateralization or abort, the remaining bond value is split 50-50 between the account which triggered the liquidation and the signers.

`FRAUD_LIQUIDATION_IN_PROGRESS` > In case of liquidation due to fraud, the remaining bond value in full goes to the account which triggered the liquidation by proving fraud.

5.15 tbtc - various deposit state transitions can be front-run (e.g. fraud proofs, timeouts) Major Won't Fix

Resolution

Addressed with the discussion at https://github.com/keep-network/tbtc/issues/498 where the client accepts that a malicious entity may always be able to front-run certain fraud proofs as long as fraud is being called out. The client also accepts that certain timeouts may be front-run which could end up in the client implementation always being front-run by a malicious actor.

> *we're comfortable with this tradeoff*

Description

An entity that can provide proof for fraudulent ECDSA signatures or SPV proofs in the liquidation flow is rewarded with part of the deposit contract ETH value.

> *Specification: Liquidation Any signer bond left over after the deposit owner is compensated is distributed to the account responsible for reporting the*

> *misbehavior (for fraud) or between the signers and the account that triggered liquidation (for collateralization issues).*

However, the methods under which proof is provided are not protected from front-running allowing anyone to observe transactions to `provideECDSAFraudProof` / `provideSPVFraudProof` and submit the same proofs with providing a higher gas value.

Please note that a similar issue exists for timeout states providing rewards for calling them out (i.e. they set the `liquidationInitiator` address).

Examples

- `provideECDSAFraudProof` verifies the fraudulent proof

`r,s,v,signedDigest` appear to be the fraudulent signature. `_preimage` is the correct value.

tbtc/implementation/contracts/deposit/DepositLiquidation.sol:L117-L137

```
/// @param _preimage        The sha256 preimage of the digest
function provideECDSAFraudProof(
    DepositUtils.Deposit storage _d,
    uint8 _v,
    bytes32 _r,
    bytes32 _s,
    bytes32 _signedDigest,
    bytes memory _preimage
) public {
    require(
        !_d.inFunding() && !_d.inFundingFailure(),
        "Use provideFundingECDSAFraudProof instead"
    );
    require(
        !_d.inSignerLiquidation(),
        "Signer liquidation already in progress"
    );
    require(!_d.inEndState(), "Contract has halted");
    require(submitSignatureFraud(_d, _v, _r, _s, _signedDigest, _preimage), "S
    startSignerFraudLiquidation(_d);
}
```

- `startSignerFraudLiquidation` sets the address that provides the proof as the beneficiary

tbtc/implementation/contracts/deposit/DepositFunding.sol:L153-L179

```
function provideFundingECDSAFraudProof(
    DepositUtils.Deposit storage _d,
    uint8 _v,
    bytes32 _r,
    bytes32 _s,
    bytes32 _signedDigest,
    bytes memory _preimage
) public {
    require(
        _d.inAwaitingBTCFundingProof(),
        "Signer fraud during funding flow only available while awaiting fundi
    );

    bool _isFraud = _d.submitSignatureFraud(_v, _r, _s, _signedDigest, _preima
    require(_isFraud, "Signature is not fraudulent");
    _d.logFraudDuringSetup();

    // If the funding timeout has elapsed, punish the funder too!
    if (block.timestamp > _d.fundingProofTimerStart + TBTCConstants.getFunding
        address(0).transfer(address(this).balance);  // Burn it all down (fir
        _d.setFailedSetup();
    } else {
        /* NB: This is reuse of the variable */
        _d.fundingProofTimerStart = block.timestamp;
        _d.setFraudAwaitingBTCFundingProof();
    }
}
```

- `purchaseSignerBondsAtAuction` pays out the funds

tbtc/implementation/contracts/deposit/DepositLiquidation.sol:L260-L276

```
    uint256 contractEthBalance = address(this).balance;
    address payable initiator = _d.liquidationInitiator;
```

```
    if (initiator == address(0)){
        initiator = address(0xdead);
    }
    if (contractEthBalance > 1) {
        if (_wasFraud) {
            initiator.transfer(contractEthBalance);
        } else {
            // There will always be a liquidation initiator.
            uint256 split = contractEthBalance.div(2);
            _d.pushFundsToKeepGroup(split);
            initiator.transfer(split);
        }
    }
}
```

Recommendation

For fraud proofs, it should be required that the reporter uses a commit/reveal scheme to lock in a proof in one block, and reveal the details in another.

5.16 tbtc - Anyone can emit log events due to missing access control Major ✓ Addressed

Resolution

Addressed with https://github.com/keep-network/tbtc/issues/477, keep-network/tbtc#467 and keep-network/tbtc#537 by restricting log calls to known `TBTCDepositToken`. `tbtcDepositToken` was moved to `DepositLog` which is not ideal.

Description

Access control for `DepositLog` is not implemented. `DepositLog` is inherited by `TBTCSystem` and its functionality is usually consumed by `Deposit` contracts to emit log events on `TBTCSystem`. Due to the missing access control, anyone can emit log events on

`TBTCSystem`. Users, client-software or other components that rely on these events might be tricked into performing actions that were not authorized by the system.

Examples

tbtc/implementation/contracts/DepositLog.sol:L95-L99

```
function approvedToLog(address _caller) public pure returns (bool) {
    /* TODO: auth via system */
    _caller;
    return true;
}
```

Recommendation

Log events are typically initiated by the Deposit contract. Make sure only Deposit contracts deployed by an approved factory can emit logs on TBTCSystem.

5.17 `DKGResultVerification.verify` unsafe packing in signed data

Medium **✓ Addressed**

Resolution
Addressed with keep-network/keep-core#1525 by adding additional checks for `groupPubKey` size, the number of signatures provided and the length of the provided misbehaved group indices. No salt was added to separate the fields.

Description

`DKGResultVerification.verify` allows the sender to arbitrarily move bytes between `groupPubKey` and `misbehaved`:

keep-core/contracts/solidity/contracts/libraries/operator/DKGResultVerification.sol:L80

```
bytes32 resultHash = keccak256(abi.encodePacked(groupPubKey, misbehaved));
```

Recommendation

Validate the expected length of both and add a salt between the two.

5.18 keep-core - Service contract callbacks can be abused to call into other contracts Medium ✓ Addressed

Resolution

Addressed with keep-network/keep-core#1532 by hardcoding the callback method signature and the following statement:

> *We still allow specifying an address of the callback contract. This could be beneficial in a situations where one contract pays for a random number for another contract.*

A subsequent change in keep-network/keep-ecdsa#339 updated `keep-tecdsa` to use the new, hardcoded callback function: `__beaconCallback(uint256)`.

Description

`KeepRandomBeaconServiceImplV1` allows senders to specify an arbitrary method and contract that will receive a callback once the beacon generates a relay entry:

keep-core/contracts/solidity/contracts/KeepRandomBeaconServiceImplV1.sol:L228-L245

```
/**
 * @dev Creates a request to generate a new relay entry, which will include
 * a random number (by signing the previous entry's random number).
 * @param callbackContract Callback contract address. Callback is called once
 * @param callbackMethod Callback contract method signature. String representa
 * uint256 input parameter i.e. "relayEntryCallback(uint256)".
 * @param callbackGas Gas required for the callback.
 * The customer needs to ensure they provide a sufficient callback gas
 * to cover the gas fee of executing the callback. Any surplus is returned
 * to the customer. If the callback gas amount turns to be not enough to
```

```
     * execute the callback, callback execution is skipped.
     * @return An uint256 representing uniquely generated relay request ID. It is
     */
    function requestRelayEntry(
        address callbackContract,
        string memory callbackMethod,
        uint256 callbackGas
    ) public nonReentrant payable returns (uint256) {
```

Once an operator contract receives the relay entry, it calls `executeCallback`:

keep-core/contracts/solidity/contracts/KeepRandomBeaconServiceImplV1.sol:L314-L335

```
    /**
     * @dev Executes customer specified callback for the relay entry request.
     * @param requestId Request id tracked internally by this contract.
     * @param entry The generated random number.
     * @return Address to receive callback surplus.
     */
    function executeCallback(uint256 requestId, uint256 entry) public returns (ad
        require(
            _operatorContracts.contains(msg.sender),
            "Only authorized operator contract can call execute callback."
        );

        require(
            _callbacks[requestId].callbackContract != address(0),
            "Callback contract not found"
        );

        _callbacks[requestId].callbackContract.call(abi.encodeWithSignature(_callb

        surplusRecipient = _callbacks[requestId].surplusRecipient;
        delete _callbacks[requestId];
    }
```

Arbitrary callbacks can be used to force the service contract to execute many functions within the keep contract system. Currently, the `KeepRandomBeaconOperator` includes an

`onlyServiceContract` modifier:

keep-core/contracts/solidity/contracts/KeepRandomBeaconOperator.sol:L150-L159

```
/**
 * @dev Checks if sender is authorized.
 */
modifier onlyServiceContract() {
    require(
        serviceContracts.contains(msg.sender),
        "Caller is not an authorized contract"
    );
    _;
}
```

The functions it protects cannot be targeted by the aforementioned service contract callbacks due to Solidity's `CALLDATASIZE` checking. However, the presence of the modifier suggests that the service contract is expected to be a permissioned actor within some contracts.

Recommendation

1. Stick to a constant callback method signature, rather than allowing users to submit an arbitrary string. An example is `__beaconCallback__(uint256)`.
2. Consider disallowing arbitrary callback destinations. Instead, rely on contracts making requests directly, and default the callback destination to `msg.sender`. Ensure the sender is not an EOA.

5.19 tbtc - Disallow signatures with high-s values in `DepositRedemption.provideRedemptionSignature` Medium ✓ Addressed

Resolution
Issue addressed in keep-network/tbtc#518

Description

`DepositRedemption.provideRedemptionSignature` is used by signers to publish a signature that can be used to redeem a deposit on Bitcoin. The function accepts a signature s value in the upper half of the secp256k1 curve:

tbtc/implementation/contracts/deposit/DepositRedemption.sol:L183-L202

```
function provideRedemptionSignature(
    DepositUtils.Deposit storage _d,
    uint8 _v,
    bytes32 _r,
    bytes32 _s
) public {
    require(_d.inAwaitingWithdrawalSignature(), "Not currently awaiting a sig

    // If we're outside of the signature window, we COULD punish signers here
    // Instead, we consider this a no-harm-no-foul situation.
    // The signers have not stolen funds. Most likely they've just inconvenie

    // The signature must be valid on the pubkey
    require(
        _d.signerPubkey().checkSig(
            _d.lastRequestedDigest,
            _v, _r, _s
        ),
        "Invalid signature"
    );
```

Although `ecrecover` accepts signatures with these s values, they are no longer used in Bitcoin. As such, the signature will appear to be valid to the Ethereum smart contract, but will likely not be accepted on Bitcoin. If no users watching malleate the signature, the redemption process will likely enter a fee increase loop, incurring a cost on the deposit owner.

Recommendation

Ensure the passed-in s value is restricted to the lower half of the secp256k1 curve, as done in `BondedECDSAKeep`:

keep-tecdsa/solidity/contracts/BondedECDSAKeep.sol:L333-L340

```
// Validate `s` value for a malleability concern described in EIP-2.
// Only signatures with `s` value in the lower half of the secp256k1
// curve's order are considered valid.
require(
    uint256(_s) <=
        0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0,
    "Malleable signature - s should be in the low half of secp256k1 curve's or
);
```

5.20 Consistent use of `SafeERC20` for external tokens Medium ✓ Addressed

Resolution
Addressed with https://github.com/keep-network/keep-core/issues/1407 and https://github.com/keep-network/keep-tecdsa/issues/272.

Description

Use `SafeERC20` features to interact with potentially broken tokens used in the system. E.g. `TokenGrant.receiveApproval()` is using `safeTransferFrom` while other contracts aren't.

Examples

- `TokenGrant.receiveApproval` using `safeTransferFrom`

keep-core/contracts/solidity/contracts/TokenGrant.sol:L200-L200

```
token.safeTransferFrom(_from, address(this), _amount);
```

- `TokenStaking.receiveApproval` not using `safeTransferFrom` while `safeTransfer` is being used.

keep-core/contracts/solidity/contracts/TokenStaking.sol:L75-L75

```
token.transferFrom(_from, address(this), _value);
```

keep-core/contracts/solidity/contracts/TokenStaking.sol:L103-L103

```
token.safeTransfer(owner, amount);
```

keep-core/contracts/solidity/contracts/TokenStaking.sol:L193-L193

```
token.transfer(tattletale, tattletaleReward);
```

- `distributeERC20ToMembers` not using `safeTransferFrom`

keep-tecdsa/solidity/contracts/BondedECDSAKeep.sol:L459-L463

```
token.transferFrom(
    msg.sender,
    tokenStaking.magpieOf(members[i]),
    dividend
);
```

Recommendation

Consistently use `SafeERC20` to support potentially broken tokens external to the system.

5.21 Initialize implementations for proxy contracts and protect initialization methods Medium ✓ Addressed

Resolution
This issue is addressed with the following changesets that ensure that the logic contracts cannot be used by other parties by initializing them in the constructor: https://github.com/keep-network/keep-tecdsa/issues/297, https://github.com/keep-network/keep-core/issues/1424, and https://github.com/keep-network/tbtc/issues/500.

Description

It should be avoided that the implementation for proxy contracts can be initialized by third parties. This can be the case if the `initialize` function is unprotected. Since the implementation contract is not meant to be used directly without a proxy delegate-calling it is recommended to protect the initialization method of the implementation by initializing on deployment.

Changing the proxies implementation (`upgradeTo()`) to a version that does not protect the initialization method may allow someone to front-run and initialize the contract if it is not done within the same transaction.

Examples

- `KeepVendor` delegates to `KeepVendorImplV1` . The implementations initialization method is unprotected.

keep-tecdsa/solidity/contracts/BondedECDSAKeepVendorImplV1.sol:L22-L32

```
/// @notice Initializes Keep Vendor contract implementation.
/// @param registryAddress Keep registry contract linked to this contract.
function initialize(
    address registryAddress
)
    public
{
    require(!initialized(), "Contract is already initialized.");
    _initialized["BondedECDSAKeepVendorImplV1"] = true;
    registry = Registry(registryAddress);
}
```

- `KeepRandomBeaconServiceImplV1` and `KeepRandomBeaconServiceUpgradeExample`

keep-core/contracts/solidity/contracts/KeepRandomBeaconServiceImplV1.sol:L118-L137

```
function initialize(
    uint256 priceFeedEstimate,
    uint256 fluctuationMargin,
    uint256 dkgContributionMargin,
```

```
    uint256 withdrawalDelay,
    address registry
)
    public
{
    require(!initialized(), "Contract is already initialized.");
    _initialized["KeepRandomBeaconServiceImplV1"] = true;
    _priceFeedEstimate = priceFeedEstimate;
    _fluctuationMargin = fluctuationMargin;
    _dkgContributionMargin = dkgContributionMargin;
    _withdrawalDelay = withdrawalDelay;
    _pendingWithdrawal = 0;
    _previousEntry = _beaconSeed;
    _registry = registry;
    _baseCallbackGas = 18845;
}
```

- `Deposit` is deployed via `cloneFactory` delegating to a `masterDepositAddress` in `DepositFactory`. The `masterDepositAddress` (`Deposit`) might be left uninitialized.

tbtc/implementation/contracts/system/DepositFactoryAuthority.sol:L3-L14

```
contract DepositFactoryAuthority {

    bool internal _initialized = false;
    address internal _depositFactory;

    /// @notice Set the address of the System contract on contract initializa
    function initialize(address _factory) public {
        require(! _initialized, "Factory can only be initialized once.");

        _depositFactory = _factory;
        _initialized = true;
    }
```

Recommendation

Initialize unprotected implementation contracts in the implementation's constructor. Protect initialization methods from being called by unauthorized parties or ensure that

deployment of the proxy and initialization is performed in the same transaction.

5.22 keep-tecdsa - If caller sends more than is contained in the signer subsidy pool, the value is burned Medium ✓ Addressed

Resolution
Issue addressed in keep-network/keep-ecdsa#306. The `subsidyPool` was removed in favor of a `reseedPool`, which is filled by the beacon by surplus sent to `requestRelayEntry`.

Description

The signer subsidy pool in `BondedECDSAKeepFactory` tracks funds sent to the contract. Each time a keep is opened, the subsidy pool is intended to be distributed to the members of the new keep:

keep-tecdsa/solidity/contracts/BondedECDSAKeepFactory.sol:L312-L320

```
// If subsidy pool is non-empty, distribute the value to signers but
// never distribute more than the payment for opening a keep.
uint256 signerSubsidy = subsidyPool < msg.value
    ? subsidyPool
    : msg.value;
if (signerSubsidy > 0) {
    subsidyPool -= signerSubsidy;
    keep.distributeETHToMembers.value(signerSubsidy)();
}
```

The tracking around subsidy pool increases is inconsistent, and can lead to sent value being burned. In the case that `subsidyPool` contains less Ether than is sent in `msg.value`, `msg.value` is unused and remains in the contract. It may or may not be added to `subsidyPool`, depending on the return status of the random beacon:

keep-tecdsa/solidity/contracts/BondedECDSAKeepFactory.sol:L347-L357

```
(bool success, ) = address(randomBeacon).call.gas(400000).value(msg.value)(
    abi.encodeWithSignature(
        "requestRelayEntry(address,string,uint256)",
        address(this),
        "setGroupSelectionSeed(uint256)",
        callbackGas
    )
);
if (!success) {
    subsidyPool += msg.value; // beacon is busy
}
```

Recommendation

Rather than tracking the `subsidyPool` individually, simply distribute `this.balance` to each new keep's members.

5.23 keep-core - TokenGrant and TokenStaking allow staking zero amount of tokens and front-running Medium ✓ Addressed

Resolution

Addressed with https://github.com/keep-network/keep-core/issues/1425 and keep-network/keep-core#1461 by requiring a hardcoded minimum amount of tokens to be staked.

Description

Tokens are staked via the callback `receiveApproval()` which is normally invoked when calling `approveAndCall()`. The method is not restricting who can initiate the staking of tokens and relies on the fact that the token transfer to the `TokenStaking` contract is pre-approved by the owner, otherwise, the call would revert.

However, `receiveApproval()` allows the staking of a zero amount of tokens. The only check performed on the number of tokens transferred is, that the token holders balance covers the amount to be transferred. This check is both relatively weak - having enough

balance does not imply that tokens are approved for transfer - and does not cover the fact that someone can call the method with a zero amount of tokens.

This way someone could create an arbitrary number of operators staking no tokens at all. This passes the token balance check, `token.transferFrom()` will succeed and an operator struct with a zero stake and arbitrary values for `operator, from, magpie, authorizer` can be set. Finally, an event is emitted for a zero stake.

An attacker could front-run calls to `receiveApproval` to block staking of a legitimate operator by creating a zero stake entry for the operator before she is able to. This vector might allow someone to permanently inconvenience an operator's address. To recover from this situation one could be forced to `cancelStake` terminating the zero stake struct in order to call the contract with the correct stake again.

The same issue exists for `TokenGrant`.

Examples

keep-core/contracts/solidity/contracts/TokenStaking.sol:L54-L81

```
/**
 * @notice Receives approval of token transfer and stakes the approved amount.
 * @dev Makes sure provided token contract is the same one linked to this cont
 * @param _from The owner of the tokens who approved them to transfer.
 * @param _value Approved amount for the transfer and stake.
 * @param _token Token contract address.
 * @param _extraData Data for stake delegation. This byte array must have the
 * following values concatenated: Magpie address (20 bytes) where the rewards
 * are sent, operator's (20 bytes) address, authorizer (20 bytes) address.
 */
function receiveApproval(address _from, uint256 _value, address _token, bytes
    require(ERC20Burnable(_token) == token, "Token contract must be the same c
    require(_value <= token.balanceOf(_from), "Sender must have enough tokens.
    require(_extraData.length == 60, "Stake delegation data must be provided."

    address payable magpie = address(uint160(_extraData.toAddress(0)));
    address operator = _extraData.toAddress(20);
    require(operators[operator].owner == address(0), "Operator address is alre
    address authorizer = _extraData.toAddress(40);
```

```
    // Transfer tokens to this contract.
    token.transferFrom(_from, address(this), _value);

    operators[operator] = Operator(_value, block.number, 0, _from, magpie, aut
    ownerOperators[_from].push(operator);

    emit Staked(operator, _value);
}
```

Recommendation

Require tokens to be staked and explicitly disallow the zero amount of tokens case. The balance check can be removed.

Note: Consider checking the calls return value or calling the contract via `SafeERC20` to support potentially broken tokens that do not revert in error cases (`token.transferFrom`).

5.24 tbtc - Inconsistency between `increaseRedemptionFee` and `provideRedemptionProof` may create un-provable redemptions

Medium **✓ Addressed**

Resolution
Issue addressed in keep-network/tbtc#522

Description

`DepositRedemption.increaseRedemptionFee` is used by signers to approve a signable bitcoin transaction with a higher fee, in case the network is congested and miners are not approving the lower-fee transaction.

Fee increases can be performed every 4 hours:

tbtc/implementation/contracts/deposit/DepositRedemption.sol:L225

```
require(block.timestamp >= _d.withdrawalRequestTime + TBTCConstants.getIncreas
```

In addition, each increase must increment the fee by exactly the initial proposed fee:

tbtc/implementation/contracts/deposit/DepositRedemption.sol:L260-L263

```
// Check that we're incrementing the fee by exactly the redeemer's initial fe
uint256 _previousOutputValue = DepositUtils.bytes8LEToUint(_previousOutputValu
_newOutputValue = DepositUtils.bytes8LEToUint(_newOutputValueBytes);
require(_previousOutputValue.sub(_newOutputValue) == _d.initialRedemptionFee,
```

Outside of these two restrictions, there is no limit to the number of times `increaseRedemptionFee` can be called. Over a 20-hour period, for example, `increaseRedemptionFee` could be called 5 times, increasing the fee to `initialRedemptionFee * 5`. Over a 24-hour period, `increaseRedemptionFee` could be called 6 times, increasing the fee to `initialRedemptionFee * 6`.

Eventually, it is expected that a transaction will be submitted and mined. At this point, anyone can call `DepositRedemption.provideRedemptionProof`, finalizing the redemption process and rewarding the signers. However, `provideRedemptionProof` will fail if the transaction fee is too high:

tbtc/implementation/contracts/deposit/DepositRedemption.sol:L308

```
require((_d.utxoSize().sub(_fundingOutputValue)) <= _d.initialRedemptionFee *
```

In the case that `increaseRedemptionFee` is called 6 times and the signers provide a signature for this transaction, the transaction can be submitted and mined but `provideRedemptionProof` for this will always fail. Eventually, a redemption proof timeout will trigger the deposit into liquidation and the signers will be punished.

Recommendation

Because it is difficult to say with certainty that a 5x fee increase will always ensure a transaction's redeemability, the upper bound on fee bumps should be removed from `provideRedemptionProof`.

This should be implemented in tandem with https://github.com/Consensys/thesis-tbtc-audit-2020-01/issues/38, so that signers cannot provide a proof that bypasses `increaseRedemptionFee` flow to spend the highest fee possible.

5.25 keep-tecdsa - keep cannot be closed if a members bond was seized or fully reassigned Medium ✓ Addressed

Description

A keep cannot be closed if the bonds have been completely reassigned or seized before, leaving at least one member with zero `lockedBonds`. In this case `closeKeep()` will throw in `freeMembersBonds()` because the requirement in `keepBonding.freeBond` is not satisfied anymore (`lockedBonds[bondID] > 0`). As a result of this, none of the potentially remaining bonds (reassign) are freed, the keep stays active even though it should be closed.

Examples

keep-tecdsa/solidity/contracts/BondedECDSAKeep.sol:L373-L396

```
/// @notice Closes keep when owner decides that they no longer need it.
/// Releases bonds to the keep members. Keep can be closed only when
/// there is no signing in progress or requested signing process has timed ou
/// @dev The function can be called by the owner of the keep and only is the
/// keep has not been closed already.
function closeKeep() external onlyOwner onlyWhenActive {
    require(
        !isSigningInProgress() || hasSigningTimedOut(),
        "Requested signing has not timed out yet"
    );

    isActive = false;

    freeMembersBonds();

    emit KeepClosed();
}

/// @notice Returns bonds to the keep members.
function freeMembersBonds() internal {
    for (uint256 i = 0; i < members.length; i++) {
        keepBonding.freeBond(members[i], uint256(address(this)));
    }
}
```

keep-tecdsa/solidity/contracts/KeepBonding.sol:L173-L190

```
/// @notice Releases the bond and moves the bond value to the operator's
/// unbounded value pool.
/// @dev Function requires that caller is the holder of the bond which is
/// being released.
/// @param operator Address of the bonded operator.
/// @param referenceID Reference ID of the bond.
function freeBond(address operator, uint256 referenceID) public {
    address holder = msg.sender;
    bytes32 bondID = keccak256(
        abi.encodePacked(operator, holder, referenceID)
    );

    require(lockedBonds[bondID] > 0, "Bond not found");

    uint256 amount = lockedBonds[bondID];
    lockedBonds[bondID] = 0;
    unbondedValue[operator] = amount;
}
```

Recommendation

Make sure the keep can be set to an end-state (closed/inactive) indicating its end-of-life even if the bond has been seized before. Avoid throwing an exception when freeing member bonds to avoid blocking the unlocking of bonds.

5.26 tbtc - `provideFundingECDSAFraudProof` attempts to burn non-existent funds Medium ✓ Addressed

Resolution
Addressed as https://github.com/keep-network/tbtc/issues/502 and fixed with keep-network/tbtc#523.

Description

The funding flow was recently changed from requiring the funder to provide a bond that stays in the Deposit contract to forwarding the funds to the keep, paying for the keep setup.

> *So at a high level, the funding bond was designed to ensure that funders had some minimum skin in the game, so that DoSing signers/the system was expensive. The upside was that we could refund it in happy paths. Now that we've realized that opening the keep itself will cost enough to prevent DoS, the concept of refunding goes away entirely. We definitely missed cleaning up the funder handling in provideFundingECDSAFraudProof though.*

Examples

tbtc/implementation/contracts/deposit/DepositFunding.sol:L170-L173

```
// If the funding timeout has elapsed, punish the funder too!
if (block.timestamp > _d.fundingProofTimerStart + TBTCConstants.getFundingTime
    address(0).transfer(address(this).balance);  // Burn it all down (fire em
    _d.setFailedSetup();
```

Recommendation

Remove the line that attempts to punish the funder by burning the Deposit contract balance which is zero due to recent changes in how the payment provided with `createNewDeposit` is handled.

5.27 bitcoin-spv - Bitcoin output script length is not checked in `wpkhSpendSighash` Medium Won't Fix

Resolution
Summa opted not to make this change. See https://github.com/summa-tx/bitcoin-spv/issues/112 for details.

Description

`CheckBitcoinSigs.wpkhSpendSighash` calculates the sighash of a Bitcoin transaction. Among its parameters, it accepts `bytes memory _outpoint`, which is a 36-byte UTXO id

consisting of a 32-byte transaction hash and a 4-byte output index.

The function in question should not accept an `_outpoint` that is not 36-bytes, but no length check is made:

bitcoin-spv/solidity/contracts/CheckBitcoinSigs.sol:L130-L159

```
function wpkhSpendSighash(
    bytes memory _outpoint,  // 36 byte UTXO id
    bytes20 _inputPKH,       // 20 byte hash160
    bytes8 _inputValue,      // 8-byte LE
    bytes8 _outputValue,     // 8-byte LE
    bytes memory _outputScript    // lenght-prefixed output script
) internal pure returns (bytes32) {
    // Fixes elements to easily make a 1-in 1-out sighash digest
    // Does not support timelocks
    bytes memory _scriptCode = abi.encodePacked(
        hex"1976a914",  // length, dup, hash160, pkh_length
        _inputPKH,
        hex"88ac");  // equal, checksig
    bytes32 _hashOutputs = abi.encodePacked(
        _outputValue,  // 8-byte LE
        _outputScript).hash256();
    bytes memory _sighashPreimage = abi.encodePacked(
        hex"01000000",  // version
        _outpoint.hash256(),  // hashPrevouts
        hex"8cb9012517c817fead650287d61bdd9c68803b6bf9c64133dcab3e65b5a50cb9",
        _outpoint,  // outpoint
        _scriptCode,  // p2wpkh script code
        _inputValue,  // value of the input in 8-byte LE
        hex"00000000",  // input nSequence
        _hashOutputs,  // hash of the single output
        hex"00000000",  // nLockTime
        hex"01000000"  // SIGHASH_ALL
    );
    return _sighashPreimage.hash256();
}
```

Recommendation

Check that `_outpoint.length` is 36.

5.28 tbtc - liquidationInitiator can block purchaseSignerBondsAtAuction indefinitely Medium ✓ Addressed

Resolution
Addressed with https://github.com/keep-network/tbtc/issues/503 and commits from keep-network/tbtc#524 switching from `transfer` to `send`.

Description

When reporting a fraudulent proof the deposits `liquidationInitiator` is set to the entity reporting and proofing the fraud. The deposit that is in a `*_liquidation_in_progress` state can be bought by anyone at an auction calling `purchaseSignerBondsAtAuction`.

Instead of receiving a share of the funds the `liquidationInitiator` can decide to intentionally reject the funds by raising an exception causing `initiator.transfer(contractEthBalance)` to throw, blocking the auction and forcing the liquidation to fail. The deposit will stay in one of the `*_liquidation_in_progress` states.

Examples

tbtc/implementation/contracts/deposit/DepositLiquidation.sol:L224-L276

```
/// @notice     Closes an auction and purchases the signer bonds. Payout to bu
/// @dev        For interface, reading auctionValue will give a past value. th
/// @param  _d  deposit storage pointer
function purchaseSignerBondsAtAuction(DepositUtils.Deposit storage _d) public
    bool _wasFraud = _d.inFraudLiquidationInProgress();
    require(_d.inSignerLiquidation(), "No active auction");

    _d.setLiquidated();
    _d.logLiquidated();

    // send the TBTC to the TDT holder. If the TDT holder is the Vending Machi
    address tdtHolder = _d.depositOwner();
```

```
    TBTCToken _tbtcToken = TBTCToken(_d.TBTCToken);

    uint256 lotSizeTbtc = _d.lotSizeTbtc();
    require(_tbtcToken.balanceOf(msg.sender) >= lotSizeTbtc, "Not enough TBTC

    if(tdtHolder == _d.VendingMachine){
        _tbtcToken.burnFrom(msg.sender, lotSizeTbtc);  // burn minimal amount
    }
    else{
        _tbtcToken.transferFrom(msg.sender, tdtHolder, lotSizeTbtc);
    }

    // Distribute funds to auction buyer
    uint256 _valueToDistribute = _d.auctionValue();
    msg.sender.transfer(_valueToDistribute);

    // Send any TBTC left to the Fee Rebate Token holder
    _d.distributeFeeRebate();

    // For fraud, pay remainder to the liquidation initiator.
    // For non-fraud, split 50-50 between initiator and signers. if the transf
    // division will yield a 0 value which causes a revert; instead,
    // we simply ignore such a tiny amount and leave some wei dust in escrow
    uint256 contractEthBalance = address(this).balance;
    address payable initiator = _d.liquidationInitiator;

    if (initiator == address(0)){
        initiator = address(0xdead);
    }
    if (contractEthBalance > 1) {
        if (_wasFraud) {
            initiator.transfer(contractEthBalance);
        } else {
            // There will always be a liquidation initiator.
            uint256 split = contractEthBalance.div(2);
            _d.pushFundsToKeepGroup(split);
            initiator.transfer(split);
        }
```

```
    }
}
```

Recommendation

Use a pull vs push funds pattern or use `address.send` instead of `address.transfer` which might leave some funds locked in the contract if it fails.

5.29 bitcoin-spv - `verifyHash256Merkle` allows existence proofs for the same leaf in multiple locations in the tree Medium Won't Fix

Resolution
Summa opted not to make this change, citing inconsistencies in Bitcoin's merkle implementation. See https://github.com/summa-tx/bitcoin-spv/issues/108 for details.

Description

`BTCUtils.verifyHash256Merkle` is used by `ValidateSPV.prove` to validate a transaction's existence in a Bitcoin block. The function accepts as input a `_proof` and an `_index`. The `_proof` consists of, in order: the transaction hash, a list of intermediate nodes, and the merkle root.

The proof is performed iteratively, and uses the `_index` to determine whether the next proof element represents a "left branch" or a "right branch:"

bitcoin-spv/solidity/contracts/BTCUtils.sol:L574-L586

```
uint _idx = _index;
bytes32 _root = _proof.slice(_proof.length - 32, 32).toBytes32();
bytes32 _current = _proof.slice(0, 32).toBytes32();

for (uint i = 1; i < (_proof.length.div(32)) - 1; i++) {
    if (_idx % 2 == 1) {
        _current = _hash256MerkleStep(_proof.slice(i * 32, 32), abi.encodePack
    } else {
        _current = _hash256MerkleStep(abi.encodePacked(_current), _proof.slice
```

```
    }
    _idx = _idx >> 1;
  }
  return _current == _root;
```

If `_idx` is even, the computed hash is placed before the next proof element. If `_idx` is odd, the computed hash is placed after the next proof element. After each iteration, `_idx` is decremented by `_idx /= 2`.

Because `verifyHash256Merkle` makes no requirements on the size of `_proof` relative to `_index`, it is possible to pass in invalid values for `_index` that prove a transaction's existence in multiple locations in the tree.

Examples

By modifying existing tests, we showed that any transaction can be proven to exist at least one alternate index. This alternate index is calculated as `(2 ** treeHeight) + prevIndex` - though other alternate indices are possible. The modified test is below:

```
it('verifies a bitcoin merkle root', async () => {
 for (let i = 0; i < verifyHash256Merkle.length; i += 1) {
   const res = await instance.verifyHash256Merkle(
     verifyHash256Merkle[i].input.proof,
     verifyHash256Merkle[i].input.index
   ); // 0-indexed
   assert.strictEqual(res, verifyHash256Merkle[i].output);

   // Now, attempt to use the same proof to verify the same leaf at
   // a different index in the tree:
   let pLen = verifyHash256Merkle[i].input.proof.length;
   let height = ((pLen - 2) / 64) - 2;

   // Only attempt to verify roots that are meant to be verified
   if (verifyHash256Merkle[i].output && height >= 1) {
     let altIdx = (2 ** height) + verifyHash256Merkle[i].input.index;

     const resNext = await instance.verifyHash256Merkle(
       verifyHash256Merkle[i].input.proof,
       altIdx
```

```
      );

      assert.strictEqual(resNext, verifyHash256Merkle[i].output);

      console.log('Verified transaction twice!');
    }
  }
});
```

Recommendation

Use the length of `_proof` to determine the maximum allowed `_index`. `_index` should satisfy the following criterion: `_index < 2 ** (_proof.length.div(32) - 2)`.

Note that subtraction by 2 accounts for the transaction hash and merkle root, which are assumed to be encoded in the proof along with the intermediate nodes.

5.30 keep-core - stake operator should not be eligible if `undelegatedAt` is set Minor ✓ Addressed

Resolution

Addressed with https://github.com/keep-network/keep-core/issues/1433 by enforcing that stake must be canceled in initialization period.

> *undelegatedAt is intended to support undelegation in advance at any given time. Whether we do < or <= is not actually significant, as transaction reordering also means ability to include/not include transactions arbitrarily, but changing the check to operator.UndelegatedAt == 0 would ruin e.g. the use-case where Alice wants to delegate to Bob for 12 months. If we don't currently need that use-case, the check can be simplified to == 0.*

Description

An operator's stake should not be eligible if they stake an amount and immediately call `undelegate` in an attempt to indicate that they are going to recover their stake soon.

Examples

keep-core/contracts/solidity/contracts/TokenStaking.sol:L232-L236

```
bool notUndelegated = block.number <= operator.undelegatedAt || operator.undel

if (isAuthorized && isActive && notUndelegated) {
    balance = operator.amount;
}
```

Recommendation

A stake that is entering undelegation is indicated by `operator.undelegatedAt` being non-zero. Change the `notUndelegated` check `block.number <= operator.undelegatedAt || operator.undelegatedAt == 0` to `operator.undelegatedAT == 0` as any value being set indicates that undelegation is in progress.

Enforce that within the initialization period stake is canceled instead of being undelegated.

5.31 keep-core - Specification inconsistency: TokenStaking amount to be slashed/seized Minor ✓ Addressed

Resolution
Partially addressed with https://github.com/keep-network/keep-core/issues/1428 by ensuring that at least some stack is slashed. As noted in the issue, the case where less than the minimum stake was slashed from an operator is left unhandled with this fix.

Description

The keep specification states that `slash` and `seize` affect at least the amount specified or the remaining stake of a member.

> *Slash each operator in the list misbehavers by the specified amount (or their remaining stake, whichever is lower).*
>
> *Punish each operator in the list misbehavers by the specified amount or their remaining stake.*

The implementation, however, bails if one of the accounts does not have enough stake to be slashed or seized because of the use of `SafeMath.sub()`. This behavior is inconsistent with the specification which states that `min(amount, misbehaver.stake)` stake should be affected. The call to slash/seize will revert and no stakes are affected. At max, the staked amount of the lowest staker can be slashed/seized from every staker.

Implementing this method as stated in the specification using `min(amount, misbehaver.stake)` will cover the fact that slashing/seizing was only partially successful. If `misbehaver.stake` is zero no error might be emitted even though no stake was slashed/seized.

Examples

keep-core/contracts/solidity/contracts/TokenStaking.sol:L151-L195

```
/**
 * @dev Slash provided token amount from every member in the misbehaved
 * operators array and burn 100% of all the tokens.
 * @param amount Token amount to slash from every misbehaved operator.
 * @param misbehavedOperators Array of addresses to seize the tokens from.
 */
function slash(uint256 amount, address[] memory misbehavedOperators)
    public
    onlyApprovedOperatorContract(msg.sender) {
    for (uint i = 0; i < misbehavedOperators.length; i++) {
        address operator = misbehavedOperators[i];
        require(authorizations[msg.sender][operator], "Not authorized");
        operators[operator].amount = operators[operator].amount.sub(amount);
    }

    token.burn(misbehavedOperators.length.mul(amount));
}

/**
 * @dev Seize provided token amount from every member in the misbehaved
 * operators array. The tattletale is rewarded with 5% of the total seized
 * amount scaled by the reward adjustment parameter and the rest 95% is burned
 * @param amount Token amount to seize from every misbehaved operator.
 * @param rewardMultiplier Reward adjustment in percentage. Min 1% and 100% ma
 * @param tattletale Address to receive the 5% reward.
```

```
 * @param misbehavedOperators Array of addresses to seize the tokens from.
 */
function seize(
    uint256 amount,
    uint256 rewardMultiplier,
    address tattletale,
    address[] memory misbehavedOperators
) public onlyApprovedOperatorContract(msg.sender) {
    for (uint i = 0; i < misbehavedOperators.length; i++) {
        address operator = misbehavedOperators[i];
        require(authorizations[msg.sender][operator], "Not authorized");
        operators[operator].amount = operators[operator].amount.sub(amount);
    }

    uint256 total = misbehavedOperators.length.mul(amount);
    uint256 tattletaleReward = (total.mul(5).div(100)).mul(rewardMultiplier).d

    token.transfer(tattletale, tattletaleReward);
    token.burn(total.sub(tattletaleReward));
}
```

Recommendation

Require that `minimumStake` has been provided and can be seized/slashed. Update the documentation to reflect the fact that the solution always seizes/slashes `minimumStake`. Ensure that stakers cannot cancel their stake while they are actively participating in the network.

5.32 keep-tecdsa - Change state-mutability of `checkSignatureFraud` to view Minor ✓ Addressed

Resolution
Addressed as part of https://github.com/keep-network/keep-tecdsa/issues/254 with commits from keep-network/keep-tecdsa#283 splitting the method into two parts: `checkSignatureFraud` declared `view-only` and `submitSignatureFraud` which initiates slashing of signer stakes.

Description

`BondedECDSAKeep.sol.submitSignatureFraud` is not state-changing and should, therefore, be declared with the function state-mutability `view`.

Examples

keep-tecdsa/solidity/contracts/BondedECDSAKeep.sol:L265-L290

```
function submitSignatureFraud(
    uint8 _v,
    bytes32 _r,
    bytes32 _s,
    bytes32 _signedDigest,
    bytes calldata _preimage
) external returns (bool _isFraud) {
    require(publicKey.length != 0, "Public key was not set yet");

    bytes32 calculatedDigest = sha256(_preimage);
    require(
        _signedDigest == calculatedDigest,
        "Signed digest does not match double sha256 hash of the preimage"
    );

    bool isSignatureValid = publicKeyToAddress(publicKey) ==
        ecrecover(_signedDigest, _v, _r, _s);

    // Check if the signature is valid but was not requested.
    require(
        isSignatureValid && !digests[_signedDigest],
        "Signature is not fraudulent"
    );

    return true;
}
```

Recommendation

Declare method as `view`. Consider renaming `submitSignatureFraud` to e.g. `checkSignatureFraud` to emphasize that it is only checking the signature and not actually

changing state.

5.33 keep-core - Specification inconsistency: `TokenStaking.slash()` is never called Minor ✓ Addressed

Resolution
Addressed with https://github.com/keep-network/keep-tecdsa/issues/254 and changesets from keep-network/keep-tecdsa#283 by slashing the signer stakes when signature fraud is proven.

Description

According to the keep specification stake should be slashed if a staker violates the protocol:

> *Slashing If a staker violates the protocol of an operation in a way which can be proven on-chain, they will be penalized by having their stakes slashed.*

While this functionality can only be called by the approved operator contract, it is not being used throughout the system. In contrast `seize()` is being called when reporting unauthorized signing or relay entry timeout.

Examples

keep-core/contracts/solidity/contracts/TokenStaking.sol:L151-L167

```
/**
 * @dev Slash provided token amount from every member in the misbehaved
 * operators array and burn 100% of all the tokens.
 * @param amount Token amount to slash from every misbehaved operator.
 * @param misbehavedOperators Array of addresses to seize the tokens from.
 */
function slash(uint256 amount, address[] memory misbehavedOperators)
    public
    onlyApprovedOperatorContract(msg.sender) {
    for (uint i = 0; i < misbehavedOperators.length; i++) {
        address operator = misbehavedOperators[i];
        require(authorizations[msg.sender][operator], "Not authorized");
```

```
            operators[operator].amount = operators[operator].amount.sub(amount);
        }

        token.burn(misbehavedOperators.length.mul(amount));
    }
```

Recommendation

Implement slashing according to the specification.

5.34 tbtc - Remove `notifyDepositExpiryCourtesyCall` and allow `exitCourtesyCall` exiting the courtesy call at term Minor ✓ Addressed

Resolution
Addressed with keep-network/tbtc#476 following the recommendation.

Description

Following a deep dive into state transitions with the client it was agreed that `notifyDepositExpiryCourtesyCall` should be removed from the system as it is a left-over of a previous version of the deposit contract.

Additionally, `exitCourtesyCall` should be callable at any time.

Examples

tbtc/implementation/contracts/deposit/DepositLiquidation.sol:L289-L298

```
    /// @notice     Goes from courtesy call to active
    /// @dev        Only callable if collateral is sufficient and the deposit is
    /// @param  _d  deposit storage pointer
    function exitCourtesyCall(DepositUtils.Deposit storage _d) public {
        require(_d.inCourtesyCall(), "Not currently in courtesy call");
        require(block.timestamp <= _d.fundedAt + TBTCConstants.getDepositTerm(), "
        require(getCollateralizationPercentage(_d) >= _d.undercollateralizedThresh
```

```
    _d.setActive();
    _d.logExitedCourtesyCall();
}
```

Recommendation

Remove the `notifyDepositExpiryCourtesyCall` state transition and remove the requirement on `exitCourtesyCall` being callable only before the deposit expires.

5.35 keep-tecdsa - withdraw should check for zero value transfer

Minor ✓ Addressed

Resolution
Addressed with https://github.com/keep-network/keep-tecdsa/issues/280 by denying zero value withdrawals.

Description

Requesting the withdrawal of zero `ETH` in `KeepBonding.withdraw` should fail as this would allow the method to succeed, calling the user-provided destination even though the sender has no unbonded value.

Examples

keep-tecdsa/solidity/contracts/KeepBonding.sol:L78-L88

```
function withdraw(uint256 amount, address payable destination) public {
    require(
        unbondedValue[msg.sender] >= amount,
        "Insufficient unbonded value"
    );

    unbondedValue[msg.sender] -= amount;

    (bool success, ) = destination.call.value(amount)("");
```

```
    require(success, "Transfer failed");
}
```

And a similar instance in `BondedECDSAKeep`:

keep-tecdsa/solidity/contracts/BondedECDSAKeep.sol:L487-L498

```
/// @notice Withdraws amount of ether hold in the keep for the member.
/// The value is sent to the beneficiary of the specific member.
/// @param _member Keep member address.
function withdraw(address _member) external {
    uint256 value = memberETHBalances[_member];
    memberETHBalances[_member] = 0;

    /* solium-disable-next-line security/no-call-value */
    (bool success, ) = tokenStaking.magpieOf(_member).call.value(value)("");

    require(success, "Transfer failed");
}
```

Recommendation

Require that the amount to be withdrawn is greater than zero.

5.36 keep-core - TokenStaking owner should be protected from `slash()` and `seize()` during initializationPeriod Minor ✓ Addressed

Resolution
Addressed by https://github.com/keep-network/keep-core/issues/1426 and fixed with keep-network/keep-core#1453.

Description

From the specification:

> *Slashing If a staker violates the protocol of an operation in a way which can be proven on-chain, they will be penalized by having their stakes slashed.*

The initialization period is a backoff time during which operator stakes are not active nor eligible to receive work. Since they cannot misbehave they should be protected from having their stake slashed or seized.

It should also be noted that `slash()` and `seize()` can be front-run during the initializationPeriod by having the operator owner cancel the deposit before it is being slashed or seized.

Recommendation

Require deposits to be in active state for being slashed or seized.

5.37 tbtc - Signer collusion may bypass `increaseRedemptionFee` flow Minor ✓ Addressed

Resolution
Issue addressed in keep-network/tbtc#522

Description

DepositRedemption.increaseRedemptionFee is used by signers to approve a signable bitcoin transaction with a higher fee, in case the network is congested and miners are not approving the lower-fee transaction.

Fee increases can be performed every 4 hours:

tbtc/implementation/contracts/deposit/DepositRedemption.sol:L225

```
require(block.timestamp >= _d.withdrawalRequestTime + TBTCConstants.getIncreas
```

In addition, each increase must increment the fee by exactly the initial proposed fee:

tbtc/implementation/contracts/deposit/DepositRedemption.sol:L260-L263

```
// Check that we're incrementing the fee by exactly the redeemer's initial fe
uint256 _previousOutputValue = DepositUtils.bytes8LEToUint(_previousOutputValu
_newOutputValue = DepositUtils.bytes8LEToUint(_newOutputValueBytes);
require(_previousOutputValue.sub(_newOutputValue) == _d.initialRedemptionFee,
```

Outside of these two restrictions, there is no limit to the number of times `increaseRedemptionFee` can be called. Over a 20-hour period, for example, `increaseRedemptionFee` could be called 5 times, increasing the fee to `initialRedemptionFee * 5`.

Rather than calling `increaseRedemptionFee` 5 times over 20 hours, colluding signers may immediately create and sign a transaction with a fee of `initialRedemptionFee * 5`, wait for it to be mined, then submit it to `provideRedemptionProof`. Because `provideRedemptionProof` does not check that a transaction signature signs an approved digest, interested parties would need to monitor the bitcoin blockchain, notice the spend, and provide an ECDSA fraud proof before `provideRedemptionProof` is called.

Recommendation

Track the latest approved fee, and ensure the transaction in `provideRedemptionProof` does not include a higher fee.

5.38 tbtc - liquidating a deposit does not send the complete remainder of the contract balance to recipients Minor ✓ Addressed

Resolution

Addressed with https://github.com/keep-network/tbtc/issues/504 and commits from keep-network/tbtc#524, transferring the remaining balance of the contract to the initiator and switching from `transfer` which might block the auction to `send`. We'd like to note that in case the `send` fails funds might be locked in the contract.

Description

`purchaseSignerBondsAtAuction` might leave a wei in the contract if:

- there is only one wei remaining in the contract

- there is more than one wei remaining but the contract balance is odd.

Examples

- contract balances must be > 1 wei otherwise no transfer is attempted
- the division at line 271 floors the result if dividing an odd balance. The contract is sending `floor(contract.balance / 2)` to the keep group and liquidationInitiator leaving one 1 in the contract.

tbtc/implementation/contracts/deposit/DepositLiquidation.sol:L266-L275

```
if (contractEthBalance > 1) {
    if (_wasFraud) {
        initiator.transfer(contractEthBalance);
    } else {
        // There will always be a liquidation initiator.
        uint256 split = contractEthBalance.div(2);
        _d.pushFundsToKeepGroup(split);
        initiator.transfer(split);
    }
}
```

Recommendation

Define a reasonable minimum amount when awarding the fraud reporter or liquidation initiator. Alternatively, always transfer the contract balance. When splitting the amount use the contract balance after the first transfer as the value being sent to the second recipient. Use the presence of locked funds in a contract as an error indicator unless funds were sent forcefully to the contract.

5.39 tbtc - `approveAndCall` unused return parameter Minor ✓ Addressed

Resolution
Addressed with https://github.com/keep-network/tbtc/issues/505 by returning `true` instead of `false`.

Description

`approveAndCall` always returns false because the return value `bool success` is never set.

Examples

tbtc/implementation/contracts/system/TBTCDepositToken.sol:L42-L54

```
/// @notice           Set allowance for other address and notify.
///                   Allows `_spender` to transfer the specified TDT
///                   on your behalf and then ping the contract about it.
/// @dev              The `_spender` should implement the `tokenRecipient` in
///                   to receive approval notifications.
/// @param _spender   Address of contract authorized to spend.
/// @param _tdtId     The TDT they can spend.
/// @param _extraData Extra information to send to the approved contract.
function approveAndCall(address _spender, uint256 _tdtId, bytes memory _extraD
    tokenRecipient spender = tokenRecipient(_spender);
    approve(_spender, _tdtId);
    spender.receiveApproval(msg.sender, _tdtId, address(this), _extraData);
}
```

Recommendation

Return the correct success state.

5.40 bitcoin-spv - Unnecessary memory allocation in `BTCUtils`

Minor Pending

Resolution
The client provided feedback that this issue is not scheduled to be addressed.

Description

`BTCUtils` makes liberal use of `BytesLib.slice`, which returns a freshly-allocated slice of an existing bytes array. In many cases, the desired behavior is simply to read a 32-byte slice

of a byte array. As a result, the typical pattern used is: `bytesVar.slice(start, start + 32).toBytes32()`.

This pattern introduces unnecessary complexity and memory allocation in a critically important library: cloning a portion of the array, storing that clone in memory, and then reading it from memory. A simpler alternative would be to implement `BytesLib.readBytes32(bytes _b, uint _idx)` and other "memory-read" functions.

Rather than moving the free memory pointer and redundantly reading, storing, then re-reading memory, `readBytes32` and similar functions would perform a simple length check and `mload` directly from the desired index in the array.

Examples

`extractInputTxIdLE`:

bitcoin-spv/solidity/contracts/BTCUtils.sol:L254-L260

```
/// @notice          Extracts the outpoint tx id from an input
/// @dev             32 byte tx id
/// @param _input    The input
/// @return          The tx id (little-endian bytes)
function extractInputTxIdLE(bytes memory _input) internal pure returns (bytes3
    return _input.slice(0, 32).toBytes32();
}
```

`verifyHash256Merkle`:

bitcoin-spv/solidity/contracts/BTCUtils.sol:L574-L586

```
uint _idx = _index;
bytes32 _root = _proof.slice(_proof.length - 32, 32).toBytes32();
bytes32 _current = _proof.slice(0, 32).toBytes32();

for (uint i = 1; i < (_proof.length.div(32)) - 1; i++) {
    if (_idx % 2 == 1) {
        _current = _hash256MerkleStep(_proof.slice(i * 32, 32), abi.encodePack
    } else {
        _current = _hash256MerkleStep(abi.encodePacked(_current), _proof.slice
    }
```

```
    _idx = _idx >> 1;
}
return _current == _root;
```

Recommendation

Implement `BytesLib.readBytes32` and favor its use over the `bytesVar.slice(start, start + 32).toBytes32()` pattern. Implement other memory-read functions where possible, and avoid the use of `slice`.

Note, too, that implementing this change in `verifyHash256Merkle` would allow `_hash256MerkleStep` to accept 2 `bytes32` inputs (rather than `bytes`), removing additional unnecessary casting and memory allocation.

5.41 bitcoin-spv - `ValidateSPV.validateHeaderChain` does not completely validate input Minor Won't Fix

Resolution

Summa opted not to make this change. See https://github.com/summa-tx/bitcoin-spv/issues/111

Description

`ValidateSPV.validateHeaderChain` takes as input a sequence of Bitcoin headers and calculates the total accumulated difficulty across the entire sequence. The input headers are checked to ensure they are relatively well-formed:

bitcoin-spv/solidity/contracts/ValidateSPV.sol:L173-L174

```
// Check header chain length
if (_headers.length % 80 != 0) {return ERR_BAD_LENGTH;}
```

However, the function lacks a check for nonzero length of `_headers`. Although the total difficulty returned would be zero, an explicit check would make this more clear.

Recommendation

If `headers.length` is zero, return `ERR_BAD_LENGTH`

5.42 bitcoin-spv - unnecessary intermediate cast Minor ✓ Addressed

Resolution
Issue addressed in summa-tx/bitcoin-spv#123

Description

`CheckBitcoinSigs.accountFromPubkey()` casts the `bytes32` keccack256 hash of the `pubkey` to `uint256`, then `uint160` and then finally to `address` while the intermediate cast is not required.

Examples

bitcoin-spv/solidity/contracts/CheckBitcoinSigs.sol:L15-L25

```
/// @notice          Derives an Ethereum Account address from a pubkey
/// @dev             The address is the last 20 bytes of the keccak256 of the
/// @param _pubkey   The public key X & Y. Unprefixed, as a 64-byte array
/// @return          The account address
function accountFromPubkey(bytes memory _pubkey) internal pure returns (addres
    require(_pubkey.length == 64, "Pubkey must be 64-byte raw, uncompressed ke

    // keccak hash of uncompressed unprefixed pubkey
    bytes32 _digest = keccak256(_pubkey);
    return address(uint160(uint256(_digest)));
}
```

Recommendation

The intermediate cast from `uint256` to `uint160` can be omitted. Refactor to `return address(uint256(_digest))` instead.

5.43 bitcoin-spv - unnecessary logic in `BytesLib.toBytes32()`

Minor ✓ Addressed

Resolution
Issue addressed in summa-tx/bitcoin-spv#125

Description

The heavily used library function `BytesLib.toBytes32()` unnecessarily casts `_source` to `bytes` (same type) and creates a copy of the dynamic byte array to check it's length, while this can be done directly on the user-provided `bytes _source`.

Examples

bitcoin-spv/solidity/contracts/BytesLib.sol:L399-L408

```
function toBytes32(bytes memory _source) pure internal returns (bytes32 result
    bytes memory tempEmptyStringTest = bytes(_source);
    if (tempEmptyStringTest.length == 0) {
        return 0x0;
    }

    assembly {
        result := mload(add(_source, 32))
    }
}
```

Recommendation

```
function toBytes32(bytes memory _source) pure internal returns (bytes32 result
        if (_source.length == 0) {
            return 0x0;
        }

        assembly {
            result := mload(add(_source, 32))
```

```
        }
    }
```

5.44 bitcoin-spv - redundant functionality Minor Won't Fix

Resolution
Summa opted not to make this change. See https://github.com/summa-tx/bitcoin-spv/issues/116 for details.

Description

The library exposes redundant implementations of bitcoins double `sha256`.

Examples

- solidity native implementation with an overzealous type correction https://github.com/Consensys/thesis-tbtc-audit-2020-01/issues/16

bitcoin-spv/solidity/contracts/BTCUtils.sol:L110-L116

```
/// @notice          Implements bitcoin's hash256 (double sha2)
/// @dev             abi.encodePacked changes the return to bytes instead of b
/// @param _b        The pre-image
/// @return          The digest
function hash256(bytes memory _b) internal pure returns (bytes32) {
    return abi.encodePacked(sha256(abi.encodePacked(sha256(_b)))).toBytes32();
}
```

- assembly implementation

Note this implementation does not handle errors when staticcall'ing the precompiled `sha256` contract (private chains).

bitcoin-spv/solidity/contracts/BTCUtils.sol:L118-L129

```
/// @notice          Implements bitcoin's hash256 (double sha2)
/// @dev             sha2 is precompiled smart contract located at address(2)
/// @param _b        The pre-image
/// @return          The digest
function hash256View(bytes memory _b) internal view returns (bytes32 res) {
    assembly {
        let ptr := mload(0x40)
        pop(staticcall(gas, 2, add(_b, 32), mload(_b), ptr, 32))
        pop(staticcall(gas, 2, ptr, 32, ptr, 32))
        res := mload(ptr)
    }
}
```

Recommendation

We recommend providing only one implementation for calculating the double `sha256` as maintaining two interfaces for the same functionality is not desirable. Furthermore, even though the assembly implementation is saving gas, we recommend keeping the language provided implementation.

5.45 bitcoin-spv - unnecessary type correction Minor ✓ Addressed

Resolution
Issue addressed in summa-tx/bitcoin-spv#126

Description

The type correction `encodePacked().toBytes32()` is not needed as `sha256` already returns `bytes32` .

Examples

bitcoin-spv/solidity/contracts/BTCUtils.sol:L114-L117

```
function hash256(bytes memory _b) internal pure returns (bytes32) {
    return abi.encodePacked(sha256(abi.encodePacked(sha256(_b)))).toBytes32();
```

```
}
```

Recommendation

Refactor to `return sha256(abi.encodePacked(sha256(_b)));` to save gas.

5.46 tbtc - Restrict access to fallback function in `Deposit.sol`

Minor ✓ Addressed

Resolution
Issue addressed in keep-network/tbtc#526

Description

`Deposit.sol` has an empty, `payable` fallback function. It is unused except when seizing signer bonds from `BondedECDSAKeep`.

Recommendation

So that Ether is not accidentally sent to a `Deposit`, have the fallback revert if the sender is not the `BondedECDSAKeep`.

5.47 tbtc - Where possible, a specific contract type should be used rather than `address`

Minor ✓ Addressed

Resolution
This issue has been addressed with https://github.com/keep-network/tbtc/issues/507 and keep-network/tbtc#542.

Description

Rather than storing `address`es and then casting to the known contract type, it's better to use the best type available so the compiler can check for type safety.

Examples

`TBTCSystem.priceFeed` is of type `address`, but it could be type `IBTCETHPriceFeed` instead. Not only would this give a little more type safety when deploying new modules, but it would avoid repeated casts throughout the codebase of the form `IBTCETHPriceFeed(priceFeed)`, `IRelay(relay)`, `TBTCSystem()`, and others.

tbtc/implementation/contracts/deposit/DepositUtils.sol:L25-L37

```
struct Deposit {

    // SET DURING CONSTRUCTION
    address TBTCSystem;
    address TBTCToken;
    address TBTCDepositToken;
    address FeeRebateToken;
    address VendingMachine;
    uint256 lotSizeSatoshis;
    uint8 currentState;
    uint256 signerFeeDivisor;
    uint128 undercollateralizedThresholdPercent;
    uint128 severelyUndercollateralizedThresholdPercent;
```

tbtc/implementation/contracts/proxy/DepositFactory.sol:L16-L28

```
contract DepositFactory is CloneFactory, TBTCSystemAuthority{

    // Holds the address of the deposit contract
    // which will be used as a master contract for cloning.
    address public masterDepositAddress;
    address public tbtcSystem;
    address public tbtcToken;
    address public tbtcDepositToken;
    address public feeRebateToken;
    address public vendingMachine;
    uint256 public keepThreshold;
    uint256 public keepSize;
```

Remediation

Where possible, use more specific types instead of `address`. This goes for parameter types as well as state variable types.

5.48 tbtc - Variable shadowing in `DepositFactory` Minor ✓ Addressed

Resolution
Issue addressed in keep-network/tbtc#512

Description

`DepositFactory` inherits from `TBTCSystemAuthority`. Both contracts declare a state variable with the same name, `tbtcSystem`.

tbtc/implementation/contracts/proxy/DepositFactory.sol:L21

```
address public tbtcSystem;
```

Recommendation

Remove the shadowed variable.

5.49 tbtc - Values may contain dirty lower-order bits Minor Pending

Description

`FundingScript` and `RedemptionScript` use `mload` to cast the first bytes of a byte array to `bytes4`. Because `mload` deals with 32-byte chunks, the resulting `bytes4` value may contain dirty lower-order bits.

Examples

`FundingScript.receiveApproval`:

tbtc/implementation/contracts/scripts/FundingScript.sol:L38-L44

```
// Verify _extraData is a call to unqualifiedDepositToTbtc.
bytes4 functionSignature;
assembly { functionSignature := mload(add(_extraData, 0x20)) }
require(
    functionSignature == vendingMachine.unqualifiedDepositToTbtc.selector,
    "Bad _extraData signature. Call must be to unqualifiedDepositToTbtc."
);
```

`RedemptionScript.receiveApproval`:

tbtc/implementation/contracts/scripts/RedemptionScript.sol:L39-L45

```
// Verify _extraData is a call to tbtcToBtc.
bytes4 functionSignature;
assembly { functionSignature := mload(add(_extraData, 0x20)) }
require(
    functionSignature == vendingMachine.tbtcToBtc.selector,
    "Bad _extraData signature. Call must be to tbtcToBtc."
);
```

Recommendation

Solidity truncates these unneeded bytes in the subsequent comparison operations, so there is no action required. However, this is good to keep in mind if these values are ever used for anything outside of strict comparison.

5.50 tbtc - Revert error string may be malformed Minor Pending

Resolution
This issue is being tracked as https://github.com/keep-network/tbtc/issues/509.

Description

`FundingScript` handles an error from a call to `VendingMachine` like so.

tbtc/implementation/contracts/scripts/FundingScript.sol:L46-L52

```
// Call the VendingMachine.
// We could explictly encode the call to vending machine, but this would
// involve manually parsing _extraData and allocating variables.
(bool success, bytes memory returnData) = address(vendingMachine).call(
    _extraData
);
require(success, string(returnData));
```

On a high-level revert, `returnData` will already include the typical "error selector". As `FundingScript` propagates this error message, it will add another error selector, which may make it difficult to read the error message.

The same issue is present in `RedemptionScript`:

tbtc/implementation/contracts/scripts/RedemptionScript.sol:L47-L52

```
(bool success, bytes memory returnData) = address(vendingMachine).call(_extraD
// By default, `address.call` will catch any revert messages.
// Converting the `returnData` to a string will effectively forward any rever
// https://ethereum.stackexchange.com/questions/69133/forward-revert-message-
// TODO: there's some noisy couple bytes at the beginning of the converted st
require(success, string(returnData));
```

Recommendation

Rather than adding an assembly-level revert to the affected contracts, ensure nested error selectors are handled in external libraries.

5.51 tbtc - Where possible, use `constant` rather than state variables

Minor ✓ Addressed

Resolution
Issue addressed in keep-network/tbtc#513

Description

`TBTCSystem` uses a state variable for `pausedDuration`, but this value is never changed.

tbtc/implementation/contracts/system/TBTCSystem.sol:L34

```
uint256 pausedDuration = 10 days;
```

Recommendation

Consider using the `constant` keyword.

5.52 tbtc - Variable shadowing in `TBTCDepositToken` constructor

Minor ✓ Addressed

Resolution
Issue addressed in keep-network/tbtc#512

Description

`TBTCDepositToken` inherits from `DepositFactoryAuthority`, which has a single state variable, `_depositFactory`. This variable is shadowed in the `TBTCDepositToken` constructor.

tbtc/implementation/contracts/system/TBTCDepositToken.sol:L21-L26

```
constructor(address _depositFactory)
    ERC721Metadata("tBTC Deopsit Token", "TDT")
    DepositFactoryAuthority(_depositFactory)
public {
    // solium-disable-previous-line no-empty-blocks
}
```

Recommendation

Rename the parameter or state variable.

Appendix 1 - Code Quality Recommendations

A.1.1 Possible faulty initialization process in `KeepRandomBeaconOperator`

`KeepRandomBeaconOperator.genesis()` may be callable multiple times if `numberOfGroups` returns zero:

```
    function genesis() public payable {
        require(numberOfGroups() == 0, "Groups exist");
        // Set latest added service contract as a group selection starter to
        groupSelectionStarterContract = ServiceContract(serviceContracts[servi
        startGroupSelection(_genesisGroupSeed, msg.value);
    }
```

Consider switching to a boolean `initialized` variable, instead.

A.1.2 Incomplete/Outdated comment and TODO's

Comments in the codebase suggest that the project is still undergoing heavy development. Check comments for accuracy and review TODO's.

- Inaccurate natspec for duplicate `@param _m`. Other params are undocumented.

```
// THIS IS THE INIT FUNCTION
/// @notice         The system can spin up a new deposit
/// @dev            This should be called by an approved contract, not a c
/// @param _m       m for m-of-n
/// @param _m       n for m-of-n
/// @return         True if successful, otherwise revert
function createNewDeposit(
    address _TBTCSystem,
    address _TBTCToken,
    address _TBTCDepositToken,
    address _FeeRebateToken,
    address _VendingMachine,
    uint256 _m,
```

```
    uint256 _n,
    uint256 _lotSize
) public onlyFactory payable returns (bool) {
    self.TBTCSystem = _TBTCSystem;
    self.TBTCToken = _TBTCToken;
    self.TBTCDepositToken = _TBTCDepositToken;
    self.FeeRebateToken = _FeeRebateToken;
    self.VendingMachine = _VendingMachine;
    self.createNewDeposit(_m, _n, _lotSize);
    return true;
}
```

- TODO's

```
function approvedToLog(address _caller) public pure returns (bool) {
/* TODO: auth via system */
_caller;
return true;
}
```

```
/* TODO: make this better than 6 */
require(
_observedDiff >= _reqDiff.mul(TBTCConstants.getTxProofDifficultyFactor()),
"Insufficient accumulated difficulty in header chain"
);
```

while the difficulty factor is actually set to 1.

```
uint256 public constant TX_PROOF_DIFFICULTY_FACTOR = 1; // TODO: decreased
```

A.1.3 Code duplication

Duplicated or logically equivalent code can be hard to maintain. We therefore recommend to avoid code duplication when feasible.

For example, in `tBTC` the contracts `TBTCSystemAuthority` and `VendingMachineAuthority` are logically equivalent. Both variants implement a subset of the functionality of openzeppelin's `Ownable`. Instead of having to maintain both variants it is recommended to create an abstracted version that fits both use-cases. This also applies to `DepositFactoryAuthority` which could be abstracted as an `Ownable` variant for proxies.

As another example, `CloneFactory.sol` lives as a copy in `keep-tecdsa/solidity/contracts` and `tbtc/implementation/contracts/proxy`.

A.1.4 Variable naming

It is good practice to follow the solidity style guidelines and naming conventions.

For example, the state variable `VendingMachine` might be mistaken as a contract type due to the non-conformant variable naming. Note that `VendingMachine` is also the name of a contract in the system.

```
contract VendingMachineAuthority {
    address internal VendingMachine;

    constructor(address _vendingMachine) public {
        VendingMachine = _vendingMachine;
    }
```

A.1.5 Share interface definitions instead of re-defining them

Both `tbtc/TBTCToken.sol` and `tbtc/TBTCDepositToken.sol` declare the same interface `tokenRecipient`. Code duplications can be hard to maintain. We, therefore, suggest avoiding code duplications when possible.

```
/**
 @dev Interface of recipient contract for approveAndCall pattern.
*/
interface tokenRecipient { function receiveApproval(address _from, uint256 _va
```

A.1.6 Visually distinguish internal from public API

Methods and Functions usually live in one of two worlds:

- public API - methods declared with visibility `public` or `external` exposed for interaction by other parties
- internal API - methods declared with visibility `internal`, `private` that are not exposed for interaction by other parties

It is good practice to visually distinguish and internal functions from public API by following commonly accepted naming convention e.g. by prefixing internal functions with an underscore (`_doSomething` vs. `doSomething`) or adding the keyword `unsafe` to unsafe functions that are not performing checks and may have a dramatic effect to the system (`_unsafePayout` vs. `RequestPayout`). Some development teams also prefer to separate publicly accessible methods (contract API) from internal methods by keeping all public methods grouped together (e.g. at the beginning of the contract).

A.1.7 Pin Solidity Version

Most of the files use a floating pragma statement `pragma solidity ^0.5.10;` . We recommend settling on the most recent version of Solidity 0.5.x.

A.1.8 Use of general-purpose third-party libraries (e.g. SafeMath)

Make sure to use only use security audited versions of third-party libraries with your codebase. Declare third-party libraries with the project's dependencies instead of copying them into your project. Copies of general purpose libraries may easily get outdated and often end up never being updated. This might leave the project vulnerable to security issues that are fixed in the upstream version already and should avoid that the codebase is using two different or modified versions of the same general-purpose library.

e.g. for SafeMath consider importing it from the openzeppelin-solidity contract package. Avoid importing a copied version of SafeMath from another third-party library (`@summa-tx/bitcoin-spv-sol/contracts/SafeMath.sol`) in favor of importing it from the original source (`openzeppelin-solidity/contracts/math/SafeMath.sol`).

A.1.9 Use enums when referencing a predefined list of contextual information

Increase compile-time checking and avoid errors from passing in invalid constants, as well as document which values are available by defining enumerations of allowed values.

For example, `keep-core/Registry.sol` defines three statis an operator contracts can be in: `DEFAULT`, `APPROVED`, and `DISABLED`. Even though mentioned as a comment they are being referred to by their integer literal instead of an enum.

```
// The registry of operator contracts
// 0 - NULL (default), 1 - APPROVED, 2 - DISABLED
mapping(address => uint256) public operatorContracts;
```

```
function approveOperatorContract(address operatorContract) public onlyRegistry
    operatorContracts[operatorContract] = 1;
}

function disableOperatorContract(address operatorContract) public onlyPanicBut
    operatorContracts[operatorContract] = 2;
}
```

A.1.10 Unused return values

Ignoring a method's return value can lead to unexpected states or conditions being overlooked. It is therefore recommended to always check a method's return value. In many cases, however, API is defined as returning a static success code (`true`) while throwing in any error condition. Since it can be assumed that the method succeeded if it does not throw, returning a success code can be omitted.

The following example of `tbtc/DepositFactory.sol` shows an instance of this issue. `deposit.createNewDeposit()` throws on error, otherwise always returns success. The return value, in this case, can be omitted.

```
  function createDeposit (uint256 _lotSize) public payable returns(address) {
       address cloneAddress = createClone(masterDepositAddress);
```

```
        Deposit deposit = Deposit(address(uint160(cloneAddress)));
        deposit.initialize(address(this));
        deposit.createNewDeposit.value(msg.value)(  //@audit - unchecked retu
                tbtcSystem,
                tbtcToken,
                tbtcDepositToken,
                feeRebateToken,
                vendingMachine,
                keepThreshold,
                keepSize,
                _lotSize
            );
```

`tbtc/Deposit.sol` and `tbtc/DepositFunding.sol`

```
    // THIS IS THE INIT FUNCTION
    /// @notice         The system can spin up a new deposit
    /// @dev            This should be called by an approved contract, not a c
    /// @param _m       m for m-of-n
    /// @param _m       n for m-of-n
    /// @return         True if successful, otherwise revert
    function createNewDeposit(
        address _TBTCSystem,
        address _TBTCToken,
        address _TBTCDepositToken,
        address _FeeRebateToken,
        address _VendingMachine,
        uint256 _m,
        uint256 _n,
        uint256 _lotSize
    ) public onlyFactory payable returns (bool) {
        self.TBTCSystem = _TBTCSystem;
        self.TBTCToken = _TBTCToken;
        self.TBTCDepositToken = _TBTCDepositToken;
        self.FeeRebateToken = _FeeRebateToken;
        self.VendingMachine = _VendingMachine;
        self.createNewDeposit(_m, _n, _lotSize);  //@audit - throws
        return true;
    }
```

Appendix 2 - Files in Scope

Our review covered the following files at the outset:

bitcoin-spv

File	SHA-1 hash
bitcoin-spv/solidity/contracts/BTCUtils.sol	c35c9ea329cc87ff74f1c5ce0c300a0d7db3
bitcoin-spv/solidity/contracts/BytesLib.sol	2178fa49f897c2afe236478a9f4559408ac8
bitcoin-spv/solidity/contracts/SafeMath.sol	7462e2ec469c36913b6fc47bafef1749f29b
bitcoin-spv/solidity/contracts/BTCUtilsDelegate.sol	ea3bc8ef148ef4fb8daff8c4c260c24ff747e4
bitcoin-spv/solidity/contracts/CheckBitcoinSigs.sol	e9624d00af1fbd377229fe767032eceec856
bitcoin-spv/solidity/contracts/CheckBitcoinSigsDelegate.sol	53c0a185f9c778df4c184921a3bec6f0c6c5
bitcoin-spv/solidity/contracts/ValidateSPV.sol	1a5fcca4dfe7b2c6ec41603044522690563
bitcoin-spv/solidity/contracts/ValidateSPVDelegate.sol	1c0bfe67ec7d9c20192e1e940a8101c0ac7

tBTC

File	SHA-1
tbtc/implementation/contracts/DepositLog.sol	0b4097f3400f2b6bfd1783
tbtc/implementation/contracts/deposit/DepositFunding.sol	c77af1cd7eb7422bc1365e
tbtc/implementation/contracts/system/TBTCToken.sol	91a9c9663212800c7b1fb
tbtc/implementation/contracts/system/VendingMachineAuthority.sol	5e63aae00f82cd5c6c7823
tbtc/implementation/contracts/system/TBTCSystem.sol	2171736428af6abd9c31fc
tbtc/implementation/contracts/system/VendingMachine.sol	17f16b793f5c0378f88680
tbtc/implementation/contracts/system/TBTCDepositToken.sol	2e926a39620647d72dbfd
tbtc/implementation/contracts/system/DepositFactoryAuthority.sol	188311a48e8b7e4491d2b
tbtc/implementation/contracts/system/FeeRebateToken.sol	0e977f37fca62daeed737e
tbtc/implementation/contracts/deposit/TBTCConstants.sol	5b0fc693173bd612cba1cb

File	SHA-1
tbtc/implementation/contracts/deposit/DepositUtils.sol	7308079022c02b2e14646
tbtc/implementation/contracts/deposit/DepositStates.sol	5ebaa3a0c9f708a98f6536
tbtc/implementation/contracts/interfaces/ITBTCSystem.sol	97a6241eea43fd6f319def
tbtc/implementation/contracts/deposit/Deposit.sol	0449315750be89b5a74a0
tbtc/implementation/contracts/deposit/DepositLiquidation.sol	613be100e9f79a8964746
tbtc/implementation/contracts/deposit/OutsourceDepositLogging.sol	790c605150564a8963be5
tbtc/implementation/contracts/deposit/DepositRedemption.sol	7ee02dd144011e257f246
tbtc/implementation/contracts/system/TBTCSystemAuthority.sol	7924969f054ee6740de374
tbtc/implementation/contracts/proxy/DepositFactory.sol	26a280871b518490022b5
tbtc/implementation/contracts/proxy/CloneFactory.sol	9044bc020f1d0132f5d408
tbtc/implementation/contracts/interfaces/IBTCETHPriceFeed.sol	d9d24818569427dbc4d64
tbtc/implementation/contracts/external/IMedianizer.sol	957d66ee5fc768bf9ff7c47
tbtc/implementation/contracts/price-feed/BTCETHPriceFeed.sol	3658670d0d66b155cdf56

keep-tecdsa

File Name	SHA-1 Hash
contracts/BondedECDSAKeep.sol	bc89cc51280d6c424fa76ac70afaca59794bf8ce
contracts/BondedECDSAKeepFactory.sol	23d428253b1f70f12e98e791ff39547edac898ad
contracts/BondedECDSAKeepVendor.sol	6397c7bac818add006ec5add72f72f8ca77dee0
contracts/BondedECDSAKeepVendorImplV1.sol	4314a3c1f5aff333db73426d35da9b545e46834
contracts/CloneFactory.sol	7408e755f2f9eb6699c04b45a8c28446041a3f73
contracts/KeepBonding.sol	a3b01f99c4fde8652f050a45fe2b4a30c6fa4b9e
contracts/api/IBondedECDSAKeep.sol	02624cb967aade2c5290cb13c9740825e905b4
contracts/api/IBondedECDSAKeepFactory.sol	30d55d502d4ef0f5aadb812ab553c6221cc1d63
contracts/api/IBondedECDSAKeepVendor.sol	764019742ba132a75ddf1272cdeb0e8a7ccb7f1

sortition-pools

File Name	SHA-1 Hash
contracts/AbstractSortitionPool.sol	7a4b163dcf5fd3ea8a9c74c5c219aadfc6c007b9
contracts/BondedSortitionPool.sol	3cde74fa4b63e4e9979dafc6418aa57ac90ec798
contracts/BondedSortitionPoolFactory.sol	49706b318ace886b3b8bd0725d546ece329958b9
contracts/Branch.sol	2571e8c19fe3f4764aa9feac8b37808f595bb407
contracts/DynamicArray.sol	ab6b782ce938cf958cc56e2c6b2a0f2334715d18
contracts/GasStation.sol	790159120d85a0dbdbfe57f729b5ada572ebbaef
contracts/Interval.sol	1fab3c416d8261f42d35d53d37c77b644fa1e3c0
contracts/Leaf.sol	22b7bee520b77214b1f81b75e352f44ad059ffc8
contracts/Position.sol	36cf18478fae2c9e22124d3ac52b5a050c7fe78b
contracts/RNG.sol	dc7862e02c56b9b033cc1db67fe19153a1e38ba7
contracts/SortitionPool.sol	e8896237641128599842d0951f8721632cfd061e
contracts/SortitionPoolFactory.sol	56bcc990f6a8cbfbd877b06ca0df43a7da21dd38
contracts/SortitionTree.sol	7d4d0fac5e8d8d1bea709280c442576751f18b33
contracts/StackLib.sol	e91cfb78f3b90ca8b3a18f701356c565a933e52e
contracts/api/IBondedSortitionPool.sol	d9fd422dc4a6ca6323a0ba536cb65f33e44c3e1b
contracts/api/IBonding.sol	71b96ff01a2efdb09e6d24b7432484b9a15a4a00
contracts/api/ISortitionPool.sol	709d56b46065c160042dcac8c2cb9a42a1ea201c
contracts/api/IStaking.sol	9412ade9ccf9f0672875d1c94b49d230dbbe4be1

keep-core

File Name	
keep-core/contracts/solidity/contracts/cryptography/AltBn128.sol	0af848f5bdf3bc54
keep-core/contracts/solidity/contracts/cryptography/BLS.sol	95f316615a6177e
keep-core/contracts/solidity/contracts/DelayedWithdrawal.sol	ad8109961339eaf
keep-core/contracts/solidity/contracts/KeepRandomBeaconOperator.sol	206cb9399c1d4c7
keep-core/contracts/solidity/contracts/KeepRandomBeaconService.sol	280a810f174100a

File Name	
keep-core/contracts/solidity/contracts/KeepRandomBeaconServiceImplV1.sol	8d23f4ef32aea55e
keep-core/contracts/solidity/contracts/KeepToken.sol	91f2bb61583f741
keep-core/contracts/solidity/contracts/Registry.sol	e1b58dd981a5baa
keep-core/contracts/solidity/contracts/StakeDelegatable.sol	0e469a07df4bb72
keep-core/contracts/solidity/contracts/TokenGrant.sol	cf6b6befe786cfc1
keep-core/contracts/solidity/contracts/TokenStaking.sol	02c0446475d84aa
keep-core/contracts/solidity/contracts/libraries/operator/DKGResultVerification.sol	132d1a7aa9c6d6c
keep-core/contracts/solidity/contracts/libraries/operator/GroupSelection.sol	8812a2027044f6a
keep-core/contracts/solidity/contracts/libraries/operator/Groups.sol	ba8c30b6340966b
keep-core/contracts/solidity/contracts/libraries/operator/Reimbursements.sol	285de769e1f56d8
keep-core/contracts/solidity/contracts/utils/AddressArrayUtils.sol	85d9bf08c8628ec
keep-core/contracts/solidity/contracts/utils/ModUtils.sol	ebf6ebc9647c6b6
keep-core/contracts/solidity/contracts/utils/ThrowProxy.sol	fa012ba7589dc8b
keep-core/contracts/solidity/contracts/utils/UintArrayUtils.sol	5d1210befba8fc72

Appendix 3 - Artifacts

This section contains some of the artifacts generated during our review by automated tools, the test suite, etc. If any issues or recommendations were identified by the output presented here, they have been addressed in the appropriate section above.

A.3.1 MythX

MythX is a security analysis API for Ethereum smart contracts. It performs multiple types of analysis, including fuzzing and symbolic execution, to detect many common vulnerability types. The tool was used for automated vulnerability discovery for all audited contracts and libraries. More details on MythX can be found at mythx.io.

A.3.2 Ethlint

Ethlint is an open source project for linting Solidity code. Only security-related issues were reviewed by the audit team.

Below is the raw output of the Ethlint vulnerability scan:

bitcoin-spv

```
solidity/contracts/BTCUtils.sol
  123:8    error    Avoid using Inline Assembly.    security/no-inline-assembl

solidity/contracts/BytesLib.sol
  41:8      error      Avoid using Inline Assembly.
  110:8     error      Avoid using Inline Assembly.
  249:8     error      Avoid using Inline Assembly.
  273:8     error      Avoid using Inline Assembly.
  284:8     error      Avoid using Inline Assembly.
  294:8     error      Avoid using Inline Assembly.
  337:8     error      Avoid using Inline Assembly.
  399:50    warning    Visibility modifier "internal" should come before other
  405:8     error      Avoid using Inline Assembly.
  410:81    warning    Visibility modifier "internal" should come before other
  413:8     error      Avoid using Inline Assembly.

solidity/contracts/CheckBitcoinSigs.sol
  177:10    error    Only use indent of 12 spaces.    indentation
  178:10    error    Only use indent of 12 spaces.    indentation
  179:10    error    Only use indent of 12 spaces.    indentation
  180:10    error    Only use indent of 12 spaces.    indentation
  181:10    error    Only use indent of 12 spaces.    indentation
  184:0     error    Only use indent of 8 spaces.     indentation
  196:6     error    Only use indent of 8 spaces.     indentation
  197:6     error    Only use indent of 8 spaces.     indentation
  198:6     error    Only use indent of 8 spaces.     indentation
  199:6     error    Only use indent of 8 spaces.     indentation
  200:6     error    Only use indent of 8 spaces.     indentation
  202:6     error    Only use indent of 8 spaces.     indentation

✖ 22 errors, 2 warnings found.
```

tBTC

```
contracts/DepositLog.sol
  114:12    warning    Avoid using 'block.timestamp'.    security/no-block-mem
  164:12    warning    Avoid using 'block.timestamp'.    security/no-block-mem
  181:12    warning    Avoid using 'block.timestamp'.    security/no-block-mem
  193:12    warning    Avoid using 'block.timestamp'.    security/no-block-mem
  205:12    warning    Avoid using 'block.timestamp'.    security/no-block-mem
  217:12    warning    Avoid using 'block.timestamp'.    security/no-block-mem
  229:12    warning    Avoid using 'block.timestamp'.    security/no-block-mem
  242:12    warning    Avoid using 'block.timestamp'.    security/no-block-mem
  255:12    warning    Avoid using 'block.timestamp'.    security/no-block-mem
  267:12    warning    Avoid using 'block.timestamp'.    security/no-block-mem
  279:12    warning    Avoid using 'block.timestamp'.    security/no-block-mem

contracts/deposit/Deposit.sol
  128:1    warning    Line contains trailing whitespace    no-trailing-whitesp
  130:7    warning    Line contains trailing whitespace    no-trailing-whitesp
  136:1    warning    Line contains trailing whitespace    no-trailing-whitesp

contracts/deposit/DepositFunding.sol
  69:37     warning    Avoid using 'block.timestamp'.    security/no-block-mem
  99:12     warning    Avoid using 'block.timestamp'.    security/no-block-mem
  121:36    warning    Avoid using 'block.timestamp'.    security/no-block-mem
  135:12    warning    Avoid using 'block.timestamp'.    security/no-block-mem
  171:12    warning    Avoid using 'block.timestamp'.    security/no-block-mem
  176:40    warning    Avoid using 'block.timestamp'.    security/no-block-mem
  190:12    warning    Avoid using 'block.timestamp'.    security/no-block-mem
  294:22    warning    Avoid using 'block.timestamp'.    security/no-block-mem

contracts/deposit/DepositLiquidation.sol
  90:34     warning    Avoid using 'block.timestamp'.       security/no-block-
  93:8      warning    Line contains trailing whitespace    no-trailing-whites
  105:34    warning    Avoid using 'block.timestamp'.       security/no-block-
  257:1     warning    Line contains trailing whitespace    no-trailing-whites
  258:1     warning    Line contains trailing whitespace    no-trailing-whites
  284:35    warning    Avoid using 'block.timestamp'.       security/no-block-
  294:16    warning    Avoid using 'block.timestamp'.       security/no-block-
```

```
  314:16    warning    Avoid using 'block.timestamp'.       security/no-block-
  323:16    warning    Avoid using 'block.timestamp'.       security/no-block-
  326:35    warning    Avoid using 'block.timestamp'.       security/no-block-

contracts/deposit/DepositRedemption.sol
  50:38     warning    Avoid using 'block.timestamp'.       security/no-block-
  127:35    warning    Avoid using 'block.timestamp'.       security/no-block-
  159:4     warning    Line contains trailing whitespace    no-trailing-whites
  163:1     warning    Line contains trailing whitespace    no-trailing-whites
  225:16    warning    Avoid using 'block.timestamp'.       security/no-block-
  238:35    warning    Avoid using 'block.timestamp'.       security/no-block-
  357:16    warning    Avoid using 'block.timestamp'.       security/no-block-
  366:16    warning    Avoid using 'block.timestamp'.       security/no-block-

contracts/deposit/DepositStates.sol
  39:65    warning    Operator "||" should be on the line where left side of t
  58:67    warning    Operator "||" should be on the line where left side of t
  69:73    warning    Operator "||" should be on the line where left side of t
  80:52    warning    Operator "||" should be on the line where left side of t
  81:54    warning    Operator "||" should be on the line where left side of t
  92:50    warning    Operator "||" should be on the line where left side of t

contracts/deposit/DepositUtils.sol
  214:11    warning    Avoid using 'block.timestamp'.       security/no-block-
  215:31    warning    Avoid using 'block.timestamp'.       security/no-block-
  225:27    warning    Avoid using 'block.timestamp'.       security/no-block-
  239:1     warning    Line contains trailing whitespace    no-trailing-whites
  240:1     warning    Line contains trailing whitespace    no-trailing-whites
  410:1     warning    Line contains trailing whitespace    no-trailing-whites
  429:1     warning    Line contains trailing whitespace    no-trailing-whites

contracts/price-feed/BTCETHPriceFeed.sol
  18:25    warning    Code contains empty block    no-empty-blocks

contracts/price-feed/MockMedianizer.sol
  11:25    warning    Code contains empty block    no-empty-blocks
  28:43    warning    Code contains empty block    no-empty-blocks
  31:43    warning    Code contains empty block    no-empty-blocks

contracts/proxy/DepositFactory.sol
```

```
  29:1    warning    Line contains trailing whitespace    no-trailing-whitespa

contracts/scripts/FundingScript.sol
  40:8     error      Avoid using Inline Assembly.              security/no-in
  49:74    warning    Avoid using low-level function 'call'.    security/no-lo

contracts/scripts/RedemptionScript.sol
  14:4     warning    Line contains trailing whitespace         no-trailing-wh
  41:8     error      Avoid using Inline Assembly.              security/no-in
  47:74    warning    Avoid using low-level function 'call'.    security/no-lo

contracts/system/TBTCDepositToken.sol
  21:1    warning    Line contains trailing whitespace    no-trailing-whitespa

contracts/system/TBTCSystem.sol
  89:1      warning    Line contains trailing whitespace    no-trailing-whites
  92:26     warning    Avoid using 'block.timestamp'.       security/no-block-
  101:16    warning    Avoid using 'block.timestamp'.       security/no-block-
  108:16    warning    Avoid using 'block.timestamp'.       security/no-block-
  110:31    warning    Avoid using 'block.timestamp'.       security/no-block-

contracts/system/VendingMachine.sol
  19:1    warning    Line contains trailing whitespace    no-trailing-whitespa

✖ 2 errors, 68 warnings found.
```

keep-tecdsa

```
contracts/BondedECDSAKeep.sol
  194:12    warning    Avoid using 'block.timestamp'.    security/no-block-mem

contracts/KeepBonding.sol
  63:10     error    Only use indent of 12 spaces.
  86:39     error    Consider using 'transfer' in place of 'call.value()'.
  220:39    error    Consider using 'transfer' in place of 'call.value()'.

contracts/api/IBondedECDSAKeep.sol
  55:0    error    Only use indent of 4 spaces.    indentation
```

```
✖ 4 errors, 1 warning found.
```

sortition-pools

```
contracts/AbstractSortitionPool.sol
  155:12    warning    Assignment operator must have exactly single space on b

✖ 1 warning found.
```

keep-core

```
contracts/DelayedWithdrawal.sol
  21:29    warning    Avoid using 'block.timestamp'.
  29:16    warning    Avoid using 'block.timestamp'.
  34:33    error      Consider using 'transfer' in place of 'call.value()'.

contracts/KeepRandomBeaconOperator.sol
  237:63    error      Consider using 'transfer' in place of 'call.value()'.
  377:19    error      Consider using 'transfer' in place of 'call.value()'.
  385:19    error      Consider using 'transfer' in place of 'call.value()'.
  486:45    error      Consider using 'transfer' in place of 'call.value()'.
  508:8     warning    Line exceeds the limit of 145 characters
  707:62    error      Consider using 'transfer' in place of 'call.value()'.

contracts/KeepRandomBeaconServiceImplV1.sol
  287:42    error      Consider using 'transfer' in place of 'call.value()'.
  331:47    warning    Avoid using low-level function 'call'.

contracts/TokenGrant.sol
  182:8     warning    Line contains trailing whitespace                  no-t
  238:12    warning    Avoid using 'now' (alias to 'block.timestamp').    secu
  240:19    warning    Avoid using 'now' (alias to 'block.timestamp').    secu
  243:31    warning    Avoid using 'now' (alias to 'block.timestamp').    secu

contracts/TokenStaking.sol
  157:1    warning    Line contains trailing whitespace    no-trailing-whitesp
```

```
contracts/cryptography/AltBn128.sol
  118:2    warning    Line contains trailing whitespace    no-trailing-whitesp
  120:8    warning    Line contains trailing whitespace    no-trailing-whitesp
  358:7    warning    Line contains trailing whitespace    no-trailing-whitesp

contracts/libraries/operator/Groups.sol
  405:8    error    Avoid using Inline Assembly.    security/no-inline-assembl

contracts/libraries/operator/Reimbursements.sol
  48:19    error    Consider using 'transfer' in place of 'call.value()'.    s
  58:19    error    Consider using 'transfer' in place of 'call.value()'.    s

contracts/utils/UintArrayUtils.sol
  6:1    warning    Line contains trailing whitespace    no-trailing-whitespac

✖ 10 errors, 13 warnings found.
```

A.3.3 Surya

Surya is a utility tool for smart contract systems. It provides a number of visual outputs and information about the structure of smart contracts. It also supports querying the function call graph in multiple ways to aid in the manual inspection and control flow analysis of contracts.

Below is a complete list of functions with their visibility and modifiers:

Contracts Description Table

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

bitcoin-spv

Contract	Type	Bases		
L	**Function Name**	**Visibility**	**Mutability**	**Modifiers**

Contract	Type	Bases		
BTCUtils	Library			
L	determineVarIntDataLength	Internal 🔒		
L	reverseEndianness	Internal 🔒		
L	reverseUint256	Internal 🔒		
L	bytesToUint	Internal 🔒		
L	lastBytes	Internal 🔒		
L	hash160	Internal 🔒		
L	hash256	Internal 🔒		
L	hash256View	Internal 🔒		
L	extractInputAtIndex	Internal 🔒		
L	isLegacyInput	Internal 🔒		
L	determineInputLength	Internal 🔒		
L	extractSequenceLELegacy	Internal 🔒		
L	extractSequenceLegacy	Internal 🔒		
L	extractScriptSig	Internal 🔒		
L	extractScriptSigLen	Internal 🔒		

Contract	Type	Bases		
L	extractSequenceLEWitness	Internal 🔒		
L	extractSequenceWitness	Internal 🔒		
L	extractOutpoint	Internal 🔒		
L	extractInputTxIdLE	Internal 🔒		
L	extractInputTxId	Internal 🔒		
L	extractTxIndexLE	Internal 🔒		
L	extractTxIndex	Internal 🔒		
L	determineOutputLength	Internal 🔒		
L	extractOutputAtIndex	Internal 🔒		
L	extractOutputScriptLen	Internal 🔒		
L	extractValueLE	Internal 🔒		
L	extractValue	Internal 🔒		
L	extractOpReturnData	Internal 🔒		
L	extractHash	Internal 🔒		
L	validateVin	Internal 🔒		
L	validateVout	Internal 🔒		

Contract	Type	Bases		
L	extractMerkleRootLE	Internal 🔒		
L	extractMerkleRootBE	Internal 🔒		
L	extractTarget	Internal 🔒		
L	calculateDifficulty	Internal 🔒		
L	extractPrevBlockLE	Internal 🔒		
L	extractPrevBlockBE	Internal 🔒		
L	extractTimestampLE	Internal 🔒		
L	extractTimestamp	Internal 🔒		
L	extractDifficulty	Internal 🔒		
L	_hash256MerkleStep	Internal 🔒		
L	verifyHash256Merkle	Internal 🔒		
L	retargetAlgorithm	Internal 🔒		
BytesLib	Library			
L	concat	Internal 🔒		
L	concatStorage	Internal 🔒	🛑	
L	slice	Internal 🔒		

Contract	Type	Bases		
L	toAddress	Internal 🔒		
L	toUint	Internal 🔒		
L	equal	Internal 🔒		
L	equalStorage	Internal 🔒		
L	toBytes32	Internal 🔒		
L	keccak256Slice	Internal 🔒		
SafeMath	Library			
L	mul	Internal 🔒		
L	div	Internal 🔒		
L	sub	Internal 🔒		
L	add	Internal 🔒		
BTCUtilsDelegate	Library			
L	determineVarIntDataLength	Public ❗		NO ❗
L	reverseEndianness	Public ❗		NO ❗
L	bytesToUint	Public ❗		NO ❗
L	lastBytes	Public ❗		NO ❗

Contract	Type	Bases		
L	hash160	Public ❗		NO ❗
L	hash256	Public ❗		NO ❗
L	extractInputAtIndex	Public ❗		NO ❗
L	isLegacyInput	Public ❗		NO ❗
L	determineInputLength	Public ❗		NO ❗
L	extractSequenceLELegacy	Public ❗		NO ❗
L	extractSequenceLegacy	Public ❗		NO ❗
L	extractScriptSig	Public ❗		NO ❗
L	extractScriptSigLen	Public ❗		NO ❗
L	extractSequenceLEWitness	Public ❗		NO ❗
L	extractSequenceWitness	Public ❗		NO ❗
L	extractOutpoint	Public ❗		NO ❗
L	extractInputTxIdLE	Public ❗		NO ❗
L	extractInputTxId	Public ❗		NO ❗
L	extractTxIndexLE	Public ❗		NO ❗
L	extractTxIndex	Public ❗		NO ❗

Contract	Type	Bases		
L	determineOutputLength	Public ❗		NO ❗
L	extractOutputAtIndex	Public ❗		NO ❗
L	extractOutputScriptLen	Public ❗		NO ❗
L	extractValueLE	Public ❗		NO ❗
L	extractValue	Public ❗		NO ❗
L	extractOpReturnData	Public ❗		NO ❗
L	extractHash	Public ❗		NO ❗
L	validateVin	Public ❗		NO ❗
L	validateVout	Public ❗		NO ❗
L	extractMerkleRootLE	Public ❗		NO ❗
L	extractMerkleRootBE	Public ❗		NO ❗
L	extractTarget	Public ❗		NO ❗
L	calculateDifficulty	Public ❗		NO ❗
L	extractPrevBlockLE	Public ❗		NO ❗
L	extractPrevBlockBE	Public ❗		NO ❗
L	extractTimestampLE	Public ❗		NO ❗

Contract	Type	Bases		
L	extractTimestamp	Public ❗		NO ❗
L	extractDifficulty	Public ❗		NO ❗
L	_hash256MerkleStep	Public ❗		NO ❗
L	verifyHash256Merkle	Public ❗		NO ❗
L	retargetAlgorithm	Public ❗		NO ❗
CheckBitcoinSigs	Library			
L	accountFromPubkey	Internal 🔒		
L	p2wpkhFromPubkey	Internal 🔒		
L	checkSig	Internal 🔒		
L	checkBitcoinSig	Internal 🔒		
L	isSha256Preimage	Internal 🔒		
L	isKeccak256Preimage	Internal 🔒		
L	wpkhSpendSighash	Internal 🔒		
L	wpkhToWpkhSighash	Internal 🔒		
L	oneInputOneOutputSighash	Internal 🔒		
CheckBitcoinSigsDelegate	Library			

Contract	Type	Bases		
L	accountFromPubkey	Public ❗		NO ❗
L	p2wpkhFromPubkey	Public ❗		NO ❗
L	checkSig	Public ❗		NO ❗
L	checkBitcoinSig	Public ❗		NO ❗
L	isSha256Preimage	Public ❗		NO ❗
L	isKeccak256Preimage	Public ❗		NO ❗
L	oneInputOneOutputSighash	Public ❗		NO ❗
ValidateSPV	Library			
L	getErrBadLength	Internal 🔒		
L	getErrInvalidChain	Internal 🔒		
L	getErrLowWork	Internal 🔒		
L	prove	Internal 🔒		
L	calculateTxId	Internal 🔒		
L	parseInput	Internal 🔒		
L	parseOutput	Internal 🔒		
L	parseHeader	Internal 🔒		

Contract	Type	Bases		
L	validateHeaderChain	Internal 🔒		
L	validateHeaderWork	Internal 🔒		
L	validateHeaderPrevHash	Internal 🔒		
ValidateSPVDelegate	Library			
L	getErrBadLength	Public ❗		NO ❗
L	getErrInvalidChain	Public ❗		NO ❗
L	getErrLowWork	Public ❗		NO ❗
L	prove	Public ❗		NO ❗
L	calculateTxId	Public ❗		NO ❗
L	parseInput	Public ❗		NO ❗
L	parseOutput	Public ❗		NO ❗
L	parseHeader	Public ❗		NO ❗
L	validateHeaderChain	Public ❗		NO ❗
L	validateHeaderWork	Public ❗		NO ❗
L	validateHeaderPrevHash	Public ❗		NO ❗

tBTC

Contract	Type	Bases

Contract	Type	Bases
L	**Function Name**	**Visibility**
DepositLog	Implementation	
L	approvedToLog	Public ❗
L	logCreated	Public ❗
L	logRedemptionRequested	Public ❗
L	logGotRedemptionSignature	Public ❗
L	logRegisteredPubkey	Public ❗
L	logSetupFailed	Public ❗
L	logFraudDuringSetup	Public ❗
L	logFunded	Public ❗
L	logCourtesyCalled	Public ❗
L	logStartedLiquidation	Public ❗
L	logRedeemed	Public ❗
L	logLiquidated	Public ❗
L	logExitedCourtesyCall	Public ❗
DepositFunding	Library	
L	fundingTeardown	Internal 🔒
L	fundingFraudTeardown	Internal 🔒
L	createNewDeposit	Public ❗
L	partiallySlashForFraudInFunding	Internal 🔒
L	distributeSignerBondsToFunder	Internal 🔒
L	notifySignerSetupFailure	Public ❗
L	retrieveSignerPubkey	Public ❗
L	notifyFundingTimeout	Public ❗
L	provideFundingECDSAFraudProof	Public ❗
L	notifyFraudFundingTimeout	Public ❗

Contract	Type	Bases
L	provideFraudBTCFundingProof	Public ❗
L	provideBTCFundingProof	Public ❗
tokenRecipient	Interface	
L	receiveApproval	External
TBTCToken	Implementation	ERC20Detailed, VendingMachine
L		Public ❗
L	mint	Public ❗
L	burnFrom	Public ❗
L	burn	Public ❗
L	approveAndCall	Public ❗
VendingMachineAuthority	Implementation	
L		Public ❗
TBTCSystem	Implementation	Ownable, ITBTC DepositLc
L		Public ❗
L	initialize	External
L	getAllowNewDeposits	External
L	emergencyPauseNewDeposits	External
L	resumeNewDeposits	Public ❗
L	getRemainingPauseTerm	Public ❗
L	setSignerFeeDivisor	External
L	getSignerFeeDivisor	External
L	setLotSizes	External
L	getAllowedLotSizes	External

Contract	Type	Bases
L	isAllowedLotSize	External
L	setCollateralizationThresholds	External
L	getUndercollateralizedThresholdPercent	External
L	getSeverelyUndercollateralizedThresholdPercent	External
L	getInitialCollateralizedPercent	External
L	fetchBitcoinPrice	External
L	fetchRelayCurrentDifficulty	External
L	fetchRelayPreviousDifficulty	External
L	createNewDepositFeeEstimate	External
L	requestNewKeep	External
VendingMachine	Implementation	TBTCSystemAu
L		Public ❗
L	setExternalAddresses	Public ❗
L	isQualified	Public ❗
L	tbtcToTdt	Public ❗
L	tdtToTbtc	Public ❗
L	unqualifiedDepositToTbtc	Public ❗
L	tbtcToBtc	Public ❗
TBTCDepositToken	Implementation	ERC721Meta DepositFactoryA
L		Public ❗
L	mint	Public ❗
L	exists	Public ❗
L	approveAndCall	Public ❗
tokenRecipient	Interface	
L	receiveApproval	External

Contract	Type	Bases
DepositFactoryAuthority	Implementation	
L	initialize	Public ❗
FeeRebateToken	Implementation	ERC721Meta VendingMachine
L		Public ❗
L	mint	Public ❗
L	exists	Public ❗
TBTCConstants	Library	
L	getBeneficiaryRewardDivisor	Public ❗
L	getSatoshiMultiplier	Public ❗
L	getFundingFraudPartialSlashDivisor	Public ❗
L	getDepositTerm	Public ❗
L	getTxProofDifficultyFactor	Public ❗
L	getSignatureTimeout	Public ❗
L	getIncreaseFeeTimer	Public ❗
L	getRedemptionProofTimeout	Public ❗
L	getMinimumRedemptionFee	Public ❗
L	getFundingTimeout	Public ❗
L	getSigningGroupFormationTimeout	Public ❗
L	getFraudFundingTimeout	Public ❗
L	getCourtesyCallTimeout	Public ❗
L	getAuctionDuration	Public ❗
L	getAuctionBasePercentage	Public ❗
L	getPermittedFeeBumps	Public ❗
DepositUtils	Library	

Contract	Type	Bases
L	currentBlockDifficulty	Public ❗
L	previousBlockDifficulty	Public ❗
L	evaluateProofDifficulty	Public ❗
L	checkProofFromTxId	Public ❗
L	findAndParseFundingOutput	Public ❗
L	validateAndParseFundingSPVProof	Public ❗
L	remainingTerm	Public ❗
L	auctionValue	Public ❗
L	lotSizeTbtc	Public ❗
L	signerFee	Public ❗
L	auctionTBTCAmount	Public ❗
L	determineCompressionPrefix	Public ❗
L	compressPubkey	Public ❗
L	signerPubkey	Public ❗
L	signerPKH	Public ❗
L	utxoSize	Public ❗
L	fetchBitcoinPrice	Public ❗
L	fetchBondAmount	Public ❗
L	bytes8LEToUint	Public ❗
L	wasDigestApprovedForSigning	Public ❗
L	feeRebateTokenHolder	Public ❗
L	depositOwner	Public ❗
L	redemptionTeardown	Public ❗
L	seizeSignerBonds	Internal 🔒
L	distributeFeeRebate	Internal 🔒
L	pushFundsToKeepGroup	Internal 🔒
L	getOwnerRedemptionTbtcRequirement	Internal 🔒

Contract	Type	Bases
L	getRedemptionTbtcRequirement	Internal
DepositStates	Library	
L	inFunding	Public
L	inFundingFailure	Public
L	inSignerLiquidation	Public
L	inRedemption	Public
L	inEndState	Public
L	inRedeemableState	Public
L	inStart	Public
L	inAwaitingSignerSetup	External
L	inAwaitingBTCFundingProof	External
L	inFraudAwaitingBTCFundingProof	External
L	inFailedSetup	External
L	inActive	External
L	inAwaitingWithdrawalSignature	External
L	inAwaitingWithdrawalProof	External
L	inRedeemed	External
L	inCourtesyCall	External
L	inFraudLiquidationInProgress	External
L	inLiquidationInProgress	External
L	inLiquidated	External
L	setAwaitingSignerSetup	External
L	setAwaitingBTCFundingProof	External
L	setFraudAwaitingBTCFundingProof	External
L	setFailedSetup	External
L	setActive	External

Contract	Type	Bases
L	setAwaitingWithdrawalSignature	External
L	setAwaitingWithdrawalProof	External
L	setRedeemed	External
L	setCourtesyCall	External
L	setFraudLiquidationInProgress	External
L	setLiquidationInProgress	External
L	setLiquidated	External
ITBTCSystem	Interface	
L	fetchBitcoinPrice	External
L	fetchRelayCurrentDifficulty	External
L	fetchRelayPreviousDifficulty	External
Deposit	Implementation	DepositFactoryA
L		Public ❗
L		External
L	getCurrentState	Public ❗
L	inActive	Public ❗
L	remainingTerm	Public ❗
L	signerFee	Public ❗
L	lotSizeSatoshis	Public ❗
L	lotSizeTbtc	Public ❗
L	utxoSize	Public ❗
L	createNewDeposit	Public ❗
L	requestRedemption	Public ❗
L	transferAndRequestRedemption	Public ❗
L	getRedemptionTbtcRequirement	Public ❗
L	getOwnerRedemptionTbtcRequirement	Public ❗

Contract	Type	Bases
L	provideRedemptionSignature	Public ❗
L	increaseRedemptionFee	Public ❗
L	provideRedemptionProof	Public ❗
L	notifySignatureTimeout	Public ❗
L	notifyRedemptionProofTimeout	Public ❗
L	notifySignerSetupFailure	Public ❗
L	retrieveSignerPubkey	Public ❗
L	notifyFundingTimeout	Public ❗
L	provideFundingECDSAFraudProof	Public ❗
L	notifyFraudFundingTimeout	Public ❗
L	provideFraudBTCFundingProof	Public ❗
L	provideBTCFundingProof	Public ❗
L	provideECDSAFraudProof	Public ❗
L	provideSPVFraudProof	Public ❗
L	purchaseSignerBondsAtAuction	Public ❗
L	notifyCourtesyCall	Public ❗
L	exitCourtesyCall	Public ❗
L	notifyUndercollateralizedLiquidation	Public ❗
L	notifyCourtesyTimeout	Public ❗
L	notifyDepositExpiryCourtesyCall	Public ❗
DepositLiquidation	Library	
L	submitSignatureFraud	Public ❗
L	getCollateralizationPercentage	Public ❗
L	startSignerFraudLiquidation	Internal 🔒
L	startSignerAbortLiquidation	Internal 🔒
L	provideECDSAFraudProof	Public ❗

Contract	Type	Bases
L	provideSPVFraudProof	Public ❗
L	validateRedeemerNotPaid	Internal 🔒
L	purchaseSignerBondsAtAuction	Public ❗
L	notifyCourtesyCall	Public ❗
L	exitCourtesyCall	Public ❗
L	notifyUndercollateralizedLiquidation	Public ❗
L	notifyCourtesyTimeout	Public ❗
L	notifyDepositExpiryCourtesyCall	Public ❗
OutsourceDepositLogging	Library	
L	logCreated	External
L	logRedemptionRequested	Public ❗
L	logGotRedemptionSignature	External
L	logRegisteredPubkey	External
L	logSetupFailed	External
L	logFraudDuringSetup	External
L	logFunded	External
L	logCourtesyCalled	External
L	logStartedLiquidation	External
L	logRedeemed	External
L	logLiquidated	External
L	logExitedCourtesyCall	External
DepositRedemption	Library	
L	distributeSignerFee	Internal 🔒
L	approveDigest	Internal 🔒
L	performRedemptionTBTCTransfers	Internal 🔒
L	_requestRedemption	Internal 🔒

Contract	Type	Bases
L	transferAndRequestRedemption	Public ❗
L	requestRedemption	Public ❗
L	provideRedemptionSignature	Public ❗
L	increaseRedemptionFee	Public ❗
L	checkRelationshipToPrevious	Public ❗
L	provideRedemptionProof	Public ❗
L	redemptionTransactionChecks	Public ❗
L	notifySignatureTimeout	Public ❗
L	notifyRedemptionProofTimeout	Public ❗
TBTCSystemAuthority	Implementation	
L		Public ❗
DepositFactory	Implementation	CloneFacto TBTCSystemAu
L		Public ❗
L	setExternalDependencies	Public ❗
L	createDeposit	Public ❗
CloneFactory	Implementation	
L	createClone	Internal 🔒
L	isClone	Internal 🔒
IBTCETHPriceFeed	Interface	
L	getPrice	External
IMedianizer	Interface	
L	read	External
BTCETHPriceFeed	Implementation	Ownable IBTCETHPrice

Contract	Type	Bases
L		Public ❗
L	initialize	External
L	getPrice	External

keep-tecdsa

Contract	Type	Bases
L	**Function Name**	**Visibility**
BondedECDSAKeep	Implementation	IBondedECDSAKeep
L	initialize	Public ❗
L	submitPublicKey	External ❗
L	hasKeyGenerationTimedOut	Internal 🔒
L	hasMemberSubmittedPublicKey	Internal 🔒
L	getPublicKey	External ❗
L	checkBondAmount	External ❗
L	seizeSignerBonds	External ❗
L	submitSignatureFraud	External ❗
L	sign	External ❗
L	isAwaitingSignature	External ❗
L	submitSignature	External ❗
L	isSigningInProgress	Internal 🔒
L	hasSigningTimedOut	Internal 🔒
L	closeKeep	External ❗
L	freeMembersBonds	Internal 🔒
L	publicKeyToAddress	Internal 🔒
L	distributeETHToMembers	External ❗
L	distributeERC20ToMembers	External ❗
L	getMemberETHBalance	External ❗

Contract	Type	Bases
L	withdraw	External ❗
BondedECDSAKeepFactory	Implementation	IBondedECDSAKeepFacto CloneFactory
L		Public ❗
L		External ❗
L	createSortitionPool	External ❗
L	getSortitionPool	External ❗
L	registerMemberCandidate	External ❗
L	isOperatorRegistered	Public ❗
L	isOperatorUpToDate	External ❗
L	updateOperatorStatus	External ❗
L	getSortitionPoolForOperator	Internal 🔒
L	openKeepFeeEstimate	Public ❗
L	openKeep	External ❗
L	newGroupSelectionSeed	Internal 🔒
L	setGroupSelectionSeed	External ❗
BondedECDSAKeepVendor	Implementation	Ownable
L		Public ❗
L	implementation	Public ❗
L	setImplementation	Internal 🔒
L		External ❗
L	upgradeTo	Public ❗
BondedECDSAKeepVendorImplV1	Implementation	IBondedECDSAKeepVend Ownable
L	initialize	Public ❗
L	initialized	Public ❗

Contract	Type	Bases
└	registerFactory	External ❗
└	selectFactory	Public ❗
CloneFactory	Implementation	
└	createClone	Internal 🔒
└	isClone	Internal 🔒
KeepBonding	Implementation	
└		Public ❗
└	availableUnbondedValue	Public ❗
└	deposit	External ❗
└	withdraw	Public ❗
└	createBond	Public ❗
└	bondAmount	Public ❗
└	reassignBond	Public ❗
└	freeBond	Public ❗
└	seizeBond	Public ❗
└	authorizeSortitionPoolContract	Public ❗
└	hasSecondaryAuthorization	Public ❗
Migrations	Implementation	
└		Public ❗
└	setCompleted	Public ❗
└	upgrade	Public ❗
IBondedECDSAKeep	Implementation	
└	getPublicKey	External ❗
└	checkBondAmount	External ❗
└	sign	External ❗

Contract	Type	Bases
L	distributeETHToMembers	External ❗
L	distributeERC20ToMembers	External ❗
L	seizeSignerBonds	External ❗
L	submitSignatureFraud	External ❗
IBondedECDSAKeepFactory	Interface	
L	openKeep	External ❗
L	openKeepFeeEstimate	External ❗
IBondedECDSAKeepVendor	Implementation	
L	selectFactory	Public ❗

sortition-pool

keep-core

Contract	Type	Bases
L	**Function Name**	**Visibility**
AltBn128	Library	
L	getP	Internal 🔒
L	g1	Internal 🔒
L	g2	Internal 🔒
L	twistB	Private 🔐
L	hexRoot	Private 🔐
L	g1YFromX	Internal 🔒
L	g2YFromX	Internal 🔒
L	g1HashToPoint	Internal 🔒
L	parity	Private 🔐
L	g1Compress	Internal 🔒
L	g2Compress	Internal 🔒

Contract	Type	Bases
└	g1Decompress	Internal 🔒
└	g1Unmarshal	Internal 🔒
└	g2Unmarshal	Internal 🔒
└	g2Decompress	Internal 🔒
└	g1Add	Internal 🔒
└	gfP2Add	Internal 🔒
└	gfP2Multiply	Internal 🔒
└	gfP2Pow	Internal 🔒
└	g2X2y	Internal 🔒
└	isG1PointOnCurve	Internal 🔒
└	isG2PointOnCurve	Internal 🔒
└	scalarMultiply	Internal 🔒
└	pairing	Internal 🔒
BLS	Library	
└	verify	Public ❗
DelayedWithdrawal	Implementation	Ownable
└	initiateWithdrawal	Public ❗
└	finishWithdrawal	Public ❗
ServiceContract	Interface	
└	entryCreated	External ❗
└	fundRequestSubsidyFeePool	External ❗
└	fundDkgFeePool	External ❗
KeepRandomBeaconOperator	Implementation	ReentrancyGuard
└	genesis	Public ❗
└		Public ❗

Contract	Type	Bases
L	addServiceContract	Public ❗
L	removeServiceContract	Public ❗
L	setPriceFeedEstimate	Public ❗
L	gasPriceWithFluctuationMargin	Internal 🔒
L	createGroup	Public ❗
L	startGroupSelection	Internal 🔒
L	isGroupSelectionPossible	Public ❗
L	submitTicket	Public ❗
L	ticketSubmissionTimeout	Public ❗
L	submittedTicketsCount	Public ❗
L	selectedParticipants	Public ❗
L	submitDkgResult	Public ❗
L	reimburseDkgSubmitter	Internal 🔒
L	setMinimumStake	Public ❗
L	sign	Public ❗
L	signRelayEntry	Internal 🔒
L	relayEntry	Public ❗
L	executeCallback	Internal 🔒
L	newEntryRewardsBreakdown	Internal 🔒
L	getDelayFactor	Internal 🔒
L	isEntryInProgress	Internal 🔒
L	hasEntryTimedOut	Internal 🔒
L	reportRelayEntryTimeout	Public ❗
L	groupProfitFee	Public ❗
L	hasMinimumStake	Public ❗
L	isGroupRegistered	Public ❗
L	isStaleGroup	Public ❗

Contract	Type	Bases
└	numberOfGroups	Public ❗
└	getGroupMemberRewards	Public ❗
└	getGroupMemberIndices	Public ❗
└	withdrawGroupMemberRewards	Public ❗
└	getFirstActiveGroupIndex	Public ❗
└	getGroupPublicKey	Public ❗
└	groupCreationGasEstimate	Public ❗
└	getGroupMembers	Public ❗
└	reportUnauthorizedSigning	Public ❗
KeepRandomBeaconService	Implementation	Ownable
└		Public ❗
└	implementation	Public ❗
└	setImplementation	Internal 🔒
└		External ❗
└	upgradeTo	Public ❗
OperatorContract	Interface	
└	entryVerificationGasEstimate	External ❗
└	groupCreationGasEstimate	External ❗
└	groupProfitFee	External ❗
└	sign	External ❗
└	numberOfGroups	External ❗
└	createGroup	External ❗
└	isGroupSelectionPossible	External ❗
KeepRandomBeaconServiceImplV1	Implementation	DelayedWithdrawal, ReentrancyGuard
└	initialize	Public ❗

Contract	Type	Bases
L	initialized	Public ❗
L	addOperatorContract	Public ❗
L	removeOperatorContract	Public ❗
L	fundDkgFeePool	Public ❗
L	fundRequestSubsidyFeePool	Public ❗
L	selectOperatorContract	Public ❗
L	requestRelayEntry	Public ❗
L	requestRelayEntry	Public ❗
L	entryCreated	Public ❗
L	executeCallback	Public ❗
L	createGroupIfApplicable	Internal 🔒
L	baseCallbackGas	Public ❗
L	setPriceFeedEstimate	Public ❗
L	priceFeedEstimate	Public ❗
L	gasPriceWithFluctuationMargin	Internal 🔒
L	callbackFee	Public ❗
L	entryFeeEstimate	Public ❗
L	entryFeeBreakdown	Public ❗
L	previousEntry	Public ❗
L	version	Public ❗
tokenRecipient	Interface	
L	receiveApproval	External ❗
KeepToken	Implementation	ERC20Burnable
L		Public ❗
L	approveAndCall	Public ❗
Migrations	Implementation	

Contract	Type	Bases
L		Public ❗
L	setCompleted	Public ❗
L	upgrade	Public ❗
Registry	Implementation	
L		Public ❗
L	setGovernance	Public ❗
L	setRegistryKeeper	Public ❗
L	setPanicButton	Public ❗
L	setOperatorContractUpgrader	Public ❗
L	approveOperatorContract	Public ❗
L	disableOperatorContract	Public ❗
L	isApprovedOperatorContract	Public ❗
L	operatorContractUpgraderFor	Public ❗
StakeDelegatable	Implementation	
L	balanceOf	Public ❗
L	operatorsOf	Public ❗
L	ownerOf	Public ❗
L	magpieOf	Public ❗
L	authorizerOf	Public ❗
tokenSender	Interface	
L	approveAndCall	External ❗
TokenGrant	Implementation	
L		Public ❗
L	balanceOf	Public ❗
L	stakeBalanceOf	Public ❗

Contract	Type	Bases
└	getGrant	Public ❗
└	getGrantVestingSchedule	Public ❗
└	getGrants	Public ❗
└	receiveApproval	Public ❗
└	withdraw	Public ❗
└	grantedAmount	Public ❗
└	withdrawable	Public ❗
└	revoke	Public ❗
└	stake	Public ❗
└	cancelStake	Public ❗
└	undelegate	Public ❗
└	recoverStake	Public ❗
TokenStaking	Implementation	StakeDelegatable
└		Public ❗
└	receiveApproval	Public ❗
└	cancelStake	Public ❗
└	undelegate	Public ❗
└	recoverStake	Public ❗
└	getUndelegation	Public ❗
└	slash	Public ❗
└	seize	Public ❗
└	authorizeOperatorContract	Public ❗
└	eligibleStake	Public ❗
└	activeStake	Public ❗
DKGResultVerification	Library	

Contract	Type	Bases
L	verify	Public ❗
GroupSelection	Library	
L	start	Public ❗
L	stop	Public ❗
L	submitTicket	Public ❗
L	submitTicket	Public ❗
L	isTicketValid	Internal 🔒
L	addTicket	Internal 🔒
L	findReplacementIndex	Internal 🔒
L	getTicketValueOrderedIndices	Internal 🔒
L	selectedParticipants	Public ❗
L	cleanupTickets	Internal 🔒
L	cleanupCandidates	Internal 🔒
Groups	Library	
L	addGroup	Internal 🔒
L	setGroupMembers	Internal 🔒
L	addGroupMemberReward	Internal 🔒
L	getGroupMemberRewards	Internal 🔒
L	getGroupPublicKey	Internal 🔒
L	getGroupMember	Internal 🔒
L	getGroupMemberIndices	Public ❗
L	terminateGroup	Internal 🔒
L	isGroupTerminated	Internal 🔒
L	isGroupRegistered	Internal 🔒
L	groupActiveTimeOf	Internal 🔒
L	groupStaleTime	Internal 🔒

Contract	Type	Bases
L	isStaleGroup	Public ❗
L	isStaleGroup	Public ❗
L	numberOfGroups	Internal 🔒
L	expireOldGroups	Internal 🔒
L	selectGroup	Public ❗
L	shiftByExpiredGroups	Internal 🔒
L	shiftByTerminatedGroups	Internal 🔒
L	withdrawFromGroup	Public ❗
L	membersOf	Public ❗
L	membersOf	Public ❗
L	reportUnauthorizedSigning	Public ❗
L	reportRelayEntryTimeout	Public ❗
L	getGroupMembers	Public ❗
Reimbursements	Library	
L	reimburseCallback	Public ❗
AddressArrayUtils	Library	
L	contains	Internal 🔒
L	removeAddress	Internal 🔒
ModUtils	Library	
L	modExp	Internal 🔒
L	modSqrt	Internal 🔒
L	legendre	Internal 🔒
ThrowProxy	Implementation	
L		Public ❗
L		External ❗

Contract	Type	Bases	
└	execute	Public ❗	
UintArrayUtils	Library		
└	removeValue	Internal 🔒	

sortition-pools

Contract	Type	Bases	
└	**Function Name**	**Visibility**	**Mutability**
AbstractSortitionPool	Implementation	SortitionTree, GasStation	
└	operatorInitBlocks	Public ❗	
└	isOperatorEligible	Public ❗	
└	isOperatorInPool	Public ❗	
└	isOperatorUpToDate	Public ❗	
└	getPoolWeight	Public ❗	
└	joinPool	Public ❗	🛑
└	updateOperatorStatus	Public ❗	🛑
└	generalizedSelectGroup	Internal 🔒	🛑
└	getEligibleWeight	Internal 🔒	
└	decideFate	Internal 🔒	
└	gasDepositSize	Internal 🔒	
BondedSortitionPool	Implementation	AbstractSortitionPool	
└		Public ❗	🛑
└	selectSetGroup	Public ❗	🛑
└	initializeSelectionParams	Internal 🔒	🛑
└	getEligibleWeight	Internal 🔒	
└	decideFate	Internal 🔒	

Contract	Type	Bases	
BondedSortitionPoolFactory	Implementation		
└	createSortitionPool	Public ❗	🛑
Branch	Library		
└	slotShift	Internal 🔒	
└	getSlot	Internal 🔒	
└	clearSlot	Internal 🔒	
└	setSlot	Internal 🔒	
└	sumWeight	Internal 🔒	
└	pickWeightedSlot	Internal 🔒	
DynamicArray	Library		
└	uintArray	Internal 🔒	
└	addressArray	Internal 🔒	
└	convert	Internal 🔒	
└	convert	Internal 🔒	
└	arrayPush	Internal 🔒	
└	arrayPush	Internal 🔒	
└	arrayPop	Internal 🔒	
└	arrayPop	Internal 🔒	
└	_allocateUints	Private 🔐	
└	_allocateAddresses	Private 🔐	
└	_copy	Private 🔐	
└	_copy	Private 🔐	
└	_push	Private 🔐	
└	_push	Private 🔐	
└	_pop	Private 🔐	
└	_pop	Private 🔐	

Contract	Type	Bases	
GasStation	Implementation		
└	depositGas	Internal 🔒	🛑
└	releaseGas	Internal 🔒	🛑
└	setDeposit	Internal 🔒	🛑
└	gasDepositSize	Internal 🔒	
Interval	Library		
└	make	Internal 🔒	
└	opWeight	Internal 🔒	
└	index	Internal 🔒	
└	setIndex	Internal 🔒	
└	insert	Internal 🔒	
└	skip	Internal 🔒	
└	remapIndices	Internal 🔒	
Leaf	Library		
└	make	Internal 🔒	
└	operator	Internal 🔒	
└	creationBlock	Internal 🔒	
└	weight	Internal 🔒	
└	setWeight	Internal 🔒	
Migrations	Implementation		
└		Public ❗	🛑
└	setCompleted	Public ❗	🛑
└	upgrade	Public ❗	🛑
Position	Library		
└	slot	Internal 🔒	

Contract	Type	Bases	
└	parent	Internal 🔒	
└	child	Internal 🔒	
└	setFlag	Internal 🔒	
└	unsetFlag	Internal 🔒	
RNG	Library		
└	initialize	Internal 🔒	
└	reseed	Internal 🔒	
└	retryIndex	Internal 🔒	
└	addSkippedInterval	Internal 🔒	
└	removeInterval	Internal 🔒	
└	generateNewIndex	Internal 🔒	
└	bitsRequired	Internal 🔒	
└	truncate	Internal 🔒	
└	getIndex	Internal 🔒	
└	getUniqueIndex	Internal 🔒	
SortitionPool	Implementation	AbstractSortitionPool	
└		Public ❗	🛑
└	selectGroup	Public ❗	🛑
└	initializeSelectionParams	Internal 🔒	
└	getEligibleWeight	Internal 🔒	
└	queryEligibleWeight	Internal 🔒	
└	decideFate	Internal 🔒	
SortitionPoolFactory	Implementation		
└	createSortitionPool	Public ❗	🛑
SortitionTree	Implementation		

Contract	Type	Bases	
└		Public ❗	🛑
└	isOperatorRegistered	Public ❗	
└	operatorsInPool	Public ❗	
└	insertOperator	Internal 🔒	🛑
└	removeOperator	Internal 🔒	🛑
└	updateOperator	Internal 🔒	🛑
└	removeOperatorLeaf	Internal 🔒	🛑
└	getFlaggedOperatorLeaf	Internal 🔒	
└	removeLeaf	Internal 🔒	🛑
└	updateLeaf	Internal 🔒	🛑
└	setLeaf	Internal 🔒	🛑
└	pickWeightedLeafWithIndex	Internal 🔒	
└	pickWeightedLeaf	Internal 🔒	
└	getEmptyLeaf	Internal 🔒	🛑
└	leavesInStack	Internal 🔒	
└	totalWeight	Internal 🔒	
StackLib	Library		
└	stackPeek	Internal 🔒	
└	stackPush	Public ❗	🛑
└	stackPop	Internal 🔒	🛑
└	getSize	Internal 🔒	
IBondedSortitionPool	Interface		
└	selectSetGroup	External ❗	🛑
└	isOperatorEligible	External ❗	
└	isOperatorInPool	External ❗	
└	isOperatorUpToDate	External ❗	

Contract	Type	Bases	
└	joinPool	External ❗	🛑
└	updateOperatorStatus	External ❗	🛑
IBonding	Interface		
└	availableUnbondedValue	External ❗	
ISortitionPool	Interface		
└	selectGroup	External ❗	🛑
└	isOperatorEligible	External ❗	
└	isOperatorInPool	External ❗	
└	isOperatorUpToDate	External ❗	
└	joinPool	External ❗	🛑
└	updateOperatorStatus	External ❗	🛑
IStaking	Interface		
└	eligibleStake	External ❗	

Appendix 4 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically,

for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.