Quantstamp Contract Security Certificate

April 10th 2020 — Quantstamp Verified

Everest

This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

Type	DAO
Auditors	Leonardo Passos, Senior Research Engineer Sebastian Banescu, Senior Research Engineer Ed Zulkoski, Senior Security Engineer
Timeline	2020-04-06 through 2020-04-10
EVM	Muir Glacier
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Everest Provided Specification
Source Code	Repository: everest — Commit: a102518
Changelog	• 2020-04-10 - Initial report
Overall Assessment	Overall, we find code quality to be good, as well as documentation and provided specification. However, we did find 13 issues, one of which is of high risk and two others are medium risk. Specifically, they concern the fact that: (1-high) data migration shall be required during contract upgrades; (2-medium) voting period is set incorrectly in the deployment scripts; and (3-medium) the DID registry value is currently undefined in the mainnet deployment flow. Moreover, the test suite currently has a branch coverage of 59%, which should be increased to as close as possible to 100%. Last, but not least, the test suite has two test cases that are failing and must be fixed to assure the tested flows work as expected.

Repository	Commit
everest	a102518

Total Issues	13	(0 Resolved)
High Risk Issues	1	(0 Resolved)
Medium Risk Issues	2	(0 Resolved)
Low Risk Issues	3	(0 Resolved)
Informational Risk Issues	4	(0 Resolved)
Undetermined Risk Issues	3	(0 Resolved)

13 Unresolved
0 Acknowledged
0 Resolved

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.

Summary of Findings

ID	Description	Severity	Status
QSP-1	Upgrading `Everest` requires data migration	^ High	Unresolved
QSP-2	`didAddress` is undefined for the `mainnet` deployment	^ Medium	Unresolved
QSP-3	`votingPeriodDuration` is set to `172800` sec (2 days)	^ Medium	Unresolved
QSP-4	Values for `charter` and `categories` must be updated for `mainnet`	∨ Low	Unresolved
QSP-5	Centralization of Power	∨ Low	Unresolved
QSP-6	Block Timestamp Manipulation	∨ Low	Unresolved
QSP-7	Unlocked Pragma	○ Informational	Unresolved
QSP-8	MockDAI token holder setup should only be done for testnet	○ Informational	Unresolved
QSP-9	Clone-and-Own	○ Informational	Unresolved
QSP-10	Race Conditions / Front-Running	○ Informational	Unresolved
QSP-11	`permit()` is given an infinite approval	? Undetermined	Unresolved
QSP-12	Challenger still gets challenge reward after exiting	? Undetermined	Unresolved
QSP-13	Missing validation on input `address` parameters	? Undetermined	Unresolved

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Truffle

- Ganache
- SolidityCoverage
- Mythril
- Securify
- Slither

Steps taken to run the tools:

1. Installed Truffle: `npm install -g truffle`
2. Installed Ganache: `npm install -g ganache-cli`
3. Installed the solidity-coverage tool (within the project's root directory): `npm install --save-dev solidity-coverage`
4. Ran the coverage tool from the project's root directory: `./node_modules/.bin/solidity-coverage`
5. Installed the Mythril tool from Pypi: `pip3 install mythril`
6. Ran the Mythril tool on each contract: `myth -x path/to/contract`
7. Ran the Securify tool: `java -Xmx6048m -jar securify-0.1.jar -fs contract.sol`
8. Installed the Slither tool: `pip install slither-analyzer`
9. Run Slither from the project directory `slither .`

Assessment

Findings

QSP-1 Upgrading `Everest` requires data migration

Severity: *High Risk*

Status: Unresolved

File(s) affected: `contract/Everest.sol`

Description: Many state variables are stored in `Everest`, as opposed to being stored in `Registry`. If `Everest` is upgraded, one must migrate the data from the old to the newly deployed `Everest` contract. Otherwise, several issues could occur, such as:

- If the `applicationFee` is changed in the upgraded contract, the payouts in `resolveChallenge()` may break.
- Challenge IDs may conflict if the upgraded contract does not use the correct starting `nonce`.

Even if one is able to successfully perform data migration to the new contract, performing it has drawbacks such as interruption of service and gas costs.

Recommendation: Move all storage values in `Everest.sol` to `Registry.sol`, providing getter/setters to manipulate them.

QSP-2 `didAddress` is undefined for the `mainnet` deployment

Severity: *Medium Risk*

Status: Unresolved

File(s) affected: `migrations/3_everest.js`

Description: `didAddress` is set for `development` and `ropsten`, but not for `mainnet`. Since `didAddress` is passed as the parameter for initializing `erc1056Registry` in `Everest`, and the latter cannot be changed after deployment, it is imperative that `didAddress` is correctly set for `mainnet`.

Recommendation: For the `mainnet` network, set the `didAddress` to an appropriate address. Do **NOT** let it `undefined`.

QSP-3 `votingPeriodDuration` is set to `172800` sec (2 days)

Severity: *Medium Risk*

Status: Unresolved

File(s) affected: `conf/config.js, L36`

Description: According to the specification provided to Quantstamp, the voting period was supposed to be set to three days.

Recommendation: The current setup does not match the specification we were given. Thus, we judge this issue to be of medium risk, requiring prompt attention from developers. As a recommendation, setting `votingPeriodDuration` to `259200` seconds (3 days) fixes the issue. Alternatively, set it as a constant in the `Everest.sol` contract, giving it `3 days` as its value.

QSP-4 Values for `charter` and `categories` must be updated for `mainnet`

Severity: Low Risk

Status: Unresolved

File(s) affected: `conf/config.js`

Description: Currently, the values of `charter` and `categories` are annotated with the following comments:

- L40 (charter): `// This points to the charter TODO - update mainnet`
- L42 (categories): `// Point to IPFS hash of categories. TODO - update mainnnet`

From the comments, it seems the values of `charter` and `categories` are not yet set for `mainnet` deployment.

Recommendation: Prior to deploying the contract on `mainnet`, update the values of `charter` and `categories` accordingly. Since `charter` and `categories` can be updated post-deployment, this issue poses less of a risk in comparison to QSP-2.

QSP-5 Centralization of Power

Severity: Low Risk

Status: Unresolved

File(s) affected: `contracts/ReserveBank.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. However, this centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

Exploit Scenario: The reserve bank is supposed to be owned by the everest contract, but the owner can transfer it to an arbitrary address at any time. If done maliciously, it would break the everest contract itself (e.g., `withdraw()` would now fail).

Recommendation: Explicitly document this issue so that users are made aware of its potential risks.

QSP-6 Block Timestamp Manipulation

Severity: Low Risk

Status: Unresolved

File(s) affected: `contracts/Everest.sol`, `contracts/Registry.sol`

Description: Projects may rely on block timestamps for various purposes. However, it's important to realize that miners individually set the timestamp of a block, and attackers may be able to manipulate timestamps for their own purposes. If a smart contract relies on a timestamp, it must take this into account.

Exploit Scenario: The voting period is determined by using block timestamps, which can be influenced by miners. Miners colluding with project owners or project delegates could either extend or reduce the voting time by 90 seconds, which could have a significant impact on the voting outcome.

Recommendation: Switch from using block timestamps to using block numbers. As a solution against a miner ending the voting time early, the interface could clearly indicate to the voters that they should vote at least 90 seconds before the official end of the voting period to avoid being denied the right to vote.

QSP-7 Unlocked Pragma

Severity: *Informational*

Status: Unresolved

File(s) affected: `contracts/*.sol`,

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.4.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked." For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version. Additionally, there are different versions of solidity in use, which need to be aligned. Different versions of Solidity are used in :

- ^0.5.8 (`lib/EthereumDIDRegistry.sol#17`)
- ^0.5.8 (`Everest.sol#11`)
- ^0.5.7 (`abdk-libraries-solidity/ABDKMath64x64.sol#13`)
- ^0.5.8 (`Registry.sol#1`)
- ^0.5.0 (`lib/Ownable.sol#2`)
- ^0.5.8 (`lib/dai.sol#22`)
- ^0.5.0 (`lib/Context.sol#2`)
- ^0.5.8 (`Migrations.sol#1`)
- ^0.5.8 (`lib/lib.sol#20`)
- ^0.5.0 (`lib/SafeMath.sol#2`)
- ^0.5.8 (`lib/AddressUtils.sol#4`)
- ^0.5.8 (`ReserveBank.sol#1`)

Recommendation: Lock a single version across all Solidity files.

QSP-8 MockDAI token holder setup should only be done for testnet

Severity: *Informational*

Status: Unresolved

File(s) affected: `migrations/2_dai.js, L24`

Description: Currently, `MockDAI` token holder setup is performed for any network that is not development, whereas it should only be done for `testnet`.

Recommendation: Change the `else` branch into `else if (network !== 'mainnet')`.

QSP-9 Clone-and-Own

Severity: *Informational*

Status: Unresolved

File(s) affected: `lib/Ownable.sol`, `lib/Context.sol`, `lib/SafeMath.sol`, `lib/AddressUtils.sol`, `abdk-libraries-solidity/ABDKMath64x64.sol`

Description: The clone-and-own approach involves copying and adjusting open source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries. Rather than the clone-and-own approach, a good industry practice is to use the Truffle framework for managing library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as, using libraries.

Recommendation: For cloned files for which an npm package exists (e.g., `Ownable` is part of OpenZeppelin), add a dependency and remove the cloned code. That allows one to benefit from recent bug fixes. For those not in the npm registry (e.g., `ABDKMath64x64.sol`), the cloning is justified.

QSP-10 Race Conditions / Front-Running

Severity: *Informational*

Status: Unresolved

File(s) affected: `contracts/Everest.sol`

Description: A block is an ordered collection of transactions from all around the network. It's possible for the ordering of these transactions to manipulate the end result of a block. A miner attacker can take advantage of this by generating and moving transactions in a way that benefits themselves.

Exploit Scenario: Suppose the following flow:

- Alice calls `resolveChallenge(...)` to receive the `resolverReward`.
- Bob wants to solve as many challenges as possible, and as such, he calls `resolveChallenge(...)` with higher gas than Alice.
- Bob' transaction is mined first; consequently, Alice call to `resolveReward(...)` fails.

Scenarios like the one above may cause certain users to concentrate most or all rewards associated to resolving challenges.

Recommendation: Unfortunately, exploits as the one mentioned are hard to mitigate in practice, unless if you limit the gas per transaction (which we do not advise). A general recommendation here is to be as transparent as possible, providing users with a clear documentation and potential issues concerning the existing implementation.

QSP-11 `permit()` is given an infinite approval

Severity: *Undetermined*

Status: Unresolved

Description: In the current implementation, `applySignedWithAttributeAndPermitInternal()` invokes `permit()`, giving Everest an infinite allowance.

Recommendation: We believe that an infinite allowance is simply to make an one-time setup such that the user never needs to approve Everest again in subsequent operations. While this comes with good intention, it is advisable to let users know that they will be giving Everest an infinite allowance. This could be achieved by externally-facing documentation. Alternatively, remove `applySignedWithAttributeAndPermitInternal()` from the contract and have users call `permit(...)` prior to executing any function in Everest for which a fee mut be paid.

QSP-12 Challenger still gets challenge reward after exiting

Severity: *Undetermined*

Status: Unresolved

File(s) affected: `contracts/Everest.sol`

Description: There is no check inside the `resolveChallenge` function whether the challenger address is still a member of the registry. Therefore, in case the challenge passes and it has more than one vote, the owner of a challenger who has exited after the challenge was resolved, still gets the challenger reward. Is this OK? Or would it be better if the challenger deposit would stay in the bank and the challenge would be cancelled?

Recommendation: Check that the `challenger` address is a member of the Everest Registry inside `resolveChallenge`; if not, then the challenger reward should just be kept in the `ReserveBank`.

QSP-13 Missing validation on input `address` parameters

Severity: *Undetermined*

Status: Unresolved

File(s) affected: `contracts/Registry.sol`, `contracts/Everest.sol`

Description: All input parameters of type `address` should be checked to be different from `0x0`. The following functions do not check this:

- `getChallengeID` on `L27` of `Registry.sol`
- `getMemberStartTime` on `L37` of `Registry.sol`
- `setMember` on `L51` of `Registry.sol`
- `editChallengeID` on `L69` of `Registry.sol`
- `deleteMember` on `L78` of `Registry.sol`
- `applySignedWithAttributeAndPermit` in `Everest.sol`
- `applySignedWithAttributeAndPermitInternal` in `Everest.sol`
- `applySignedWithAttribute` in `Everest.sol`
- `memberExit` in `Everest.sol`
- `isMember` in `Everest.sol`
- `memberChallengeExists` in `Everest.sol`
- `challenge` in `Everest.sol`
- `submitVote` in `Everest.sol` This effectively means that `0x0` can be a member of the Everest registry.

Recommendation: Given the functions pointed out in the description, add a `require` statement to check whether the input address is different from `0x0`.

Automated Analyses

Mythril

Nothing reported by Mythril.

Slither

Slither provided the following output (filtered by auditors to exclude any false positives):

- `AddressUtils.isContract(address)` (`lib/AddressUtils.sol#27-36`) is declared view but contains assembly code
- `i` in `Everest.submitVotes(uint256,Everest.VoteChoice[],address[])` (`Everest.sol#524`) is a local variable never explicitly initialized
- `Everest.applySignedWithAttributeAndPermit._owner (local variable @ Everest.sol#257)` shadows: `Ownable._owner (state variable @ lib/Ownable.sol#15)`
- `Everest.applySignedWithAttributeAndPermitInternal._owner (local variable @ Everest.sol#281)` shadows `Ownable._owner (state variable @ lib/Ownable.sol#15)`
- `Everest.applySignedWithAttribute._owner (local variable @ Everest.sol#327)` shadows `Ownable._owner (state variable @ lib/Ownable.sol#15)`
- `Everest.onlyMemberOwnerOrDelegate.owner (local variable @ Everest.sol#155)` shadows: `Ownable.owner (function @ lib/Ownable.sol#30-32)`
- `Everest.onlyMemberOwner.owner (local variable @ Everest.sol#174)` shadows: `Ownable.owner (function @ lib/Ownable.sol#30-32)`
- `Everest.delegateType should be constant (Everest.sol#54)`
- `Registry.getChallengeID(address) (Registry.sol#27-30)`should be declared `external`
- `Registry.getMemberStartTime(address) (Registry.sol#37-40)` should be declared `external`
- `Everest.submitVotes(uint256,Everest.VoteChoice[],address[]) (Everest.sol#515-527)` should be declared `external`
- `Everest.resolveChallenge(uint256) (Everest.sol#535-593)` should be declared `external`
- `Everest.withdraw(address,uint256) (Everest.sol#605-609)` should be declared `external`
- `Everest.transferOwnershipReserveBank(address) (Everest.sol#616-619)` should be declared `external`
- `Everest.transferOwnershipRegistry(address) (Everest.sol#626-629)` should be declared `external`
- `ReserveBank.withdraw(address,uint256) (ReserveBank.sol#37-44)` should be declared `external`

Adherence to Best Practices

- `Everest.sol`: in the constructor, `_challengeDeposit` and `_applicationFee` can be zero. Add a `require` statement to assure they cannot be zero.
- `dai.sol` and `lib.sol`: filenames do not adhere to the naming pattern of other files. Rename `dai.sol` to `Dai.sol` and `lib.sol` to `Lib.sol`

- to be consistent with the naming of other files. Furthermore, it is advisable to use a more descriptive name for `lib.sol`.

- `Everest.sol`: typo in comment at L286. Change `owners behalf` to `owner's behalf`.

- `Everest.sol`: `transferOwnershipReserveBank` and `transferOwnershipRegistry` always return `true` or revert otherwise. Same with `challengeCanBeResolved`. Since `false` is never returned and execution is reverted in case of error, there is no need to return any value at all. Another possibility is to change this function to a modifier.

- `Everest.sol`: `withdraw` does not check if `_amount` is greater than zero. Add a `require` statement checking that `_amount` is greater than zero.

- `Everest.sol`, L442: typo -- change `challengers vote` to `challengers' vote`.

- `Everest.sol`, L420, 491: there are increments performed without using SafeMath. In theory, the number of challengers and votes is bounded by the number of addresses in Ethereum, which is in theory, unbounded. In practice, overflow in lines L420 and 491 is unlikely to occur, but it is advisable to be defensive and use SafeMath instead of relying on Solidity's arithmetic operators. Same with line 417 and 438.

- `Everest.sol`, L550: same as previous issue; however, unlikely to occur in practice, unless one incorrectly sets an incorrect value for `challengeDeposit` and `applicationFee` (e.g., `115792089237316195423570985008687907853269984665640564039457584007913129639934` and `1`). In any case, using `SafeMath` is always a good practice.

- What is the expected gas cost of `resolveChallenge()`? Is it even worth calling with a reward of 1 DAI? Please perform gas analysis.

- The comments of the `onlyMemberOwnerOrDelegate` and `onlyMemberOwner` modifiers in `Everest.sol` are misleading, because they indicate that the member would be interacting with Everest by calling the function; however, it is not the member itself making the call, but the owner of the member. We recommend aligning the comment with the implementation.

- The comment of the `challenge` function in `Everest.sol` is not accurate, because it says that the `_challenger` and `_chalengee` parameters are member names. However, these are member addresses.

- Unnamed constants should be replaced with a meaningful constant name -- e.g., `L543` in `Everest.sol` uses the constant value `10`. Name your constants.

- In `Everest.sol`, the `submitVotes` function contains a for loop on L524, which is bounded by a user determined array. If this array has a large enough length the transaction will reach the block gas limit and fail after using a large amount of gas. We suggest performing a gas analysis and determine which is the `voteChoices` array length at which this function call would fail. Knowing such length, add a require statement at the beginning of the function that permits execution only if the array is less than the previously determined value. Otherwise it should indicate that the array is too long in the error message.

Test Results

Test Suite Results

```
Deploying token to a test network and minting 100M DAI.....
Giving tokens to 4 accounts
Allocating 20000000 DAI tokens to 0xFFcf8FDEE72ac11b5c542428B35EEF5769C409f0.
Allocating 20000000 DAI tokens to 0x22d491Bde2303f2f43325b2108D26f1eAbA1e32b.
Allocating 20000000 DAI tokens to 0xE11BA2b4D45Eaed5996Cd0823791E0C93114882d.
Allocating 20000000 DAI tokens to 0xd03ea8624C8C5987235048901fB614fDcA89b117.
mock DAI Address: 0xCfEB869F69431e42cdB54A4F4f105C19C080A601
Ethr DID Address: 0xe982E462b094850F12AF94d21D470e21bE9D0E9C
ReserveBank Address: 0x0290FB167208Af455bB137780163b7B7a9a10C16
Registry Address: 0x9b1f7F645351AF3631a656421eD2e40f2802E6c0
Everest Address: 0x67B5656d60a809915323Bf2C40A8bEF15A152e3e
  Contract: EthereumDIDRegistry
    identityOwner()
      default owner
         ✓ should return the identity address itself (52ms)
      changed owner
         ✓ should return the delegate address
    changeOwner()
      using msg.sender
        as current owner
           ✓ should change owner mapping
           ✓ should sets changed to transaction block
           ✓ should create DIDDelegateChanged event
        as new owner
           ✓ should change owner mapping
           ✓ should sets changed to transaction block
           ✓ should create DIDOwnerChanged event
        as original owner
           ✓ should fail (74ms)
        as attacker
           ✓ should fail (87ms)
      using signature
        as current owner
           ✓ should change owner mapping
           ✓ should sets changed to transaction block
           ✓ should create DIDOwnerChanged event
    addDelegate()
      using msg.sender
         ✓ validDelegate should be false
        as current owner
           ✓ validDelegate should be true (52ms)
           ✓ should sets changed to transaction block
           ✓ should create DIDDelegateChanged event
        as attacker
           ✓ should fail (73ms)
      using signature
        as current owner
           ✓ validDelegate should be true (67ms)
           ✓ should sets changed to transaction block (58ms)
           ✓ should create DIDDelegateChanged event
    revokeDelegate()
      using msg.sender
         ✓ validDelegate should be true (62ms)
        as current owner
           ✓ validDelegate should be false
           ✓ should sets changed to transaction block
           ✓ should create DIDDelegateChanged event
        as attacker
           ✓ should fail (95ms)
      using signature
        as current owner
           ✓ validDelegate should be false (44ms)
           ✓ should sets changed to transaction block
           ✓ should create DIDDelegateChanged event
    setAttribute()
      using msg.sender
        as current owner
           ✓ should sets changed to transaction block
           ✓ should create DIDAttributeChanged event
        as attacker
           ✓ should fail (161ms)
      using signature
        as current owner
           ✓ should sets changed to transaction block
           ✓ should create DIDDelegateChanged event
    revokeAttribute()
      using msg.sender
        as current owner
           ✓ should sets changed to transaction block
           ✓ should create DIDAttributeChanged event
        as attacker
           ✓ should fail (53ms)
      using signature
        as current owner
           ✓ should sets changed to transaction block
           ✓ should create DIDDelegateChanged event
  Contract: everest
    Test challenges. Functions: challenge(), submitVote(), resolveChallenge(), memberChallengeExists(),
isMember()
Member 0x1dF62f291b2E969fB0849d99D9Ce41e2F137006e successfully added
Member 0xACa94ef8bD5ffEE41947b4585a84BdA5a3d3DA6E successfully added
Member 0x28a8746e75304c0780E011BEd21C72cD78cd535E successfully added
Member 0x3E5e9111Ae8eB78Fe1CC3bb8915d5D461F3Ef9A9 successfully added
Member 0x95cED938F7991cd0dFcb48F0a06a40FA1aF46EBC successfully added
      1) should allow a member to be challenged, lose, and be removed.Also tests challengee cannot vote on their
own challenge
    Events emitted during test:
    ---------------------------
    Dai.Transfer(
      src: <indexed> 0x90F8bf6A479f320ead074411a4B0e7944Ea8c9C1 (type: address),
      dst: <indexed> 0x0290FB167208Af455bB137780163b7B7a9a10C16 (type: address),
      wad: 1000000000000000000 (type: uint256)
    )
    Everest.MemberChallenged(
      member: <indexed> 0x95cED938F7991cd0dFcb48F0a06a40FA1aF46EBC (type: address),
      challengeID: <indexed> 1 (type: uint256),
      challenger: <indexed> 0x1dF62f291b2E969fB0849d99D9Ce41e2F137006e (type: address),
      challengeEndTime: 1586008216 (type: uint256),
      details: 0x5555555555555555555555555555555555555555555555555555555555554444 (type: bytes32)
    )
    Everest.SubmitVote(
      challengeID: <indexed> 1 (type: uint256),
      submitter: <indexed> 0x90F8bf6A479f320ead074411a4B0e7944Ea8c9C1 (type: address),
      votingMember: <indexed> 0x1dF62f291b2E969fB0849d99D9Ce41e2F137006e (type: address),
      voteChoice: Everest.VoteChoice.Yes (type: enum Everest.VoteChoice),
      voteWeight: 415 (type: uint256)
    )
    Everest.SubmitVote(
      challengeID: <indexed> 1 (type: uint256),
      submitter: <indexed> 0xFFcf8FDEE72ac11b5c542428B35EEF5769C409f0 (type: address),
      votingMember: <indexed> 0xACa94ef8bD5ffEE41947b4585a84BdA5a3d3DA6E (type: address),
      voteChoice: Everest.VoteChoice.Yes (type: enum Everest.VoteChoice),
      voteWeight: 415 (type: uint256)
    )
    Everest.SubmitVote(
      challengeID: <indexed> 1 (type: uint256),
      submitter: <indexed> 0x22d491Bde2303f2f43325b2108D26f1eAbA1e32b (type: address),
      votingMember: <indexed> 0x28a8746e75304c0780E011BEd21C72cD78cd535E (type: address),
      voteChoice: Everest.VoteChoice.Yes (type: enum Everest.VoteChoice),
      voteWeight: 415 (type: uint256)
    )
    Dai.Transfer(
      src: <indexed> 0x0290FB167208Af455bB137780163b7B7a9a10C16 (type: address),
      dst: <indexed> 0x90F8bf6A479f320ead074411a4B0e7944Ea8c9C1 (type: address),
      wad: 1900000000000000000 (type: uint256)
    )
    Dai.Transfer(
      src: <indexed> 0x0290FB167208Af455bB137780163b7B7a9a10C16 (type: address),
      dst: <indexed> 0xd03ea8624C8C5987235048901fB614fDcA89b117 (type: address),
      wad: 100000000000000000 (type: uint256)
    )
    Everest.ChallengeSucceeded(
      member: <indexed> 0x95cED938F7991cd0dFcb48F0a06a40FA1aF46EBC (type: address),
      challengeID: <indexed> 1 (type: uint256),
      yesVotes: 1245 (type: uint256),
      noVotes: 0 (type: uint256),
      voterCount: 3 (type: uint256),
      challengerReward: 1900000000000000000 (type: uint256),
      resolverReward: 100000000000000000 (type: uint256)
    )
    ---------------------------
```

```
       ✓ should allow a member to be challenged, win, and stay (816ms)
       ✓ challenge should fail when no one votes except the challenger (595ms)
       ✓ challenger cant challenge self. challenger must exist. challengee must exist (235ms)
      2) challengee cannot have two challenges against them. and challengee cannot exit during ongoing challenge
    Events emitted during test:
    ---------------------------
    Dai.Transfer(
      src: <indexed> 0x90F8bf6A479f320ead074411a4B0e7944Ea8c9C1 (type: address),
      dst: <indexed> 0x0290FB167208Af455bB137780163b7B7a9a10C16 (type: address),
      wad: 1000000000000000000 (type: uint256)
    )
    Everest.MemberChallenged(
      member: <indexed> 0x3E5e9111Ae8eB78Fe1CC3bb8915d5D461F3Ef9A9 (type: address),
      challengeID: <indexed> 4 (type: uint256),
      challenger: <indexed> 0x1dF62f291b2E969fB0849d99D9Ce41e2F137006e (type: address),
      challengeEndTime: 1586526622 (type: uint256),
      details: 0x5555555555555555555555555555555555555555555555555555555555554444 (type: bytes32)
    )
    Everest.SubmitVote(
      challengeID: <indexed> 4 (type: uint256),
      submitter: <indexed> 0x90F8bf6A479f320ead074411a4B0e7944Ea8c9C1 (type: address),
      votingMember: <indexed> 0x1dF62f291b2E969fB0849d99D9Ce41e2F137006e (type: address),
      voteChoice: Everest.VoteChoice.Yes (type: enum Everest.VoteChoice),
      voteWeight: 831 (type: uint256)
    )
    ---------------------------
  Contract: Everest
    Delegates - Testing delegate voting
Member 0x1dF62f291b2E969fB0849d99D9Ce41e2F137006e successfully added
Member 0xACa94ef8bD5ffEE41947b4585a84BdA5a3d3DA6E successfully added
Member 0x28a8746e75304c0780E011BEd21C72cD78cd535E successfully added
Member 0x3E5e9111Ae8eB78Fe1CC3bb8915d5D461F3Ef9A9 successfully added
       ✓ Allows a delegate to vote for the owner (490ms)
  Contract: Everest
    Everest owner functionality. Functions: withdraw(), updateCharter()
       ✓ should allow owner to update the charter (58ms)
       ✓ should allow owner to update the categories (74ms)
Member 0x1dF62f291b2E969fB0849d99D9Ce41e2F137006e successfully added
       ✓ should allow owner to withdraw DAI from reserve bank (562ms)
       ✓ should allow owner the transfer of ReserveBank (68ms)
       ✓ should allow owner the transfer of Registry (123ms)
  Contract: Everest
    Member joining and leaving. Functions: applySignedWithAttribute(), memberExit()
Member 0x1dF62f291b2E969fB0849d99D9Ce41e2F137006e successfully added
       ✓ Should allow member to join the registry (598ms)
       ✓ Should prevent a member from double joining (468ms)
       ✓ Should allow a member to exit (127ms)
       ✓ Should revert if non-owner try to exit a member (88ms)
    Member editing. Functions: setAttribute()
       ✓ should allow an updated owner to set attribute (77ms)
       ✓ should disallow a non-owner to set attribute (80ms)
  Contract: everest
    Test the submitVotes() function
Member 0x1dF62f291b2E969fB0849d99D9Ce41e2F137006e successfully added
Member 0xACa94ef8bD5ffEE41947b4585a84BdA5a3d3DA6E successfully added
Member 0x28a8746e75304c0780E011BEd21C72cD78cd535E successfully added
Member 0x3E5e9111Ae8eB78Fe1CC3bb8915d5D461F3Ef9A9 successfully added
Member 0x95cED938F7991cd0dFcb48F0a06a40FA1aF46EBC successfully added
       ✓ submitVotes() works as intended, and fails with unequal arrays (1183ms)
  Contract: Everest
    Test voting require statements and functionality
Member 0x1dF62f291b2E969fB0849d99D9Ce41e2F137006e successfully added
Member 0xACa94ef8bD5ffEE41947b4585a84BdA5a3d3DA6E successfully added
Member 0x28a8746e75304c0780E011BEd21C72cD78cd535E successfully added
Member 0x3E5e9111Ae8eB78Fe1CC3bb8915d5D461F3Ef9A9 successfully added
Member 0x95cED938F7991cd0dFcb48F0a06a40FA1aF46EBC successfully added
       ✓ Voting on a challenge that does not exist fails (79ms)
       ✓ Voting must be yes or no, any other choice fails (457ms)
       ✓ Vote weight is calculated as sqrt(challengeEndTime - memberStartTime) (217ms)
       ✓ Double voting on a challenge fails. (125ms)
       ✓ Voting by a non-member fails (64ms)
       ✓ Voting on an expired challenge fails (120ms)
  61 passing (22s)
  2 failing
  1) Contract: everest
       Test challenges. Functions: challenge(), submitVote(), resolveChallenge(), memberChallengeExists(),
isMember()
         should allow a member to be challenged, lose, and be removed.Also tests challengee cannot vote on their
own challenge:
      Reserve bank did not send out challenge deposit and application fee
      + expected - actual
      -4900000000000000000
      +4000000000000000000

      at Object.resolveChallenge (test/helpers.js:367:20)
      at processTicksAndRejections (internal/process/next_tick.js:81:5)
  2) Contract: everest
       Test challenges. Functions: challenge(), submitVote(), resolveChallenge(), memberChallengeExists(),
isMember()
         challengee cannot have two challenges against them. and challengee cannot exit during ongoing
challenge:
      Wrong kind of exception received
      + expected - actual
      -challenge - Existing challenge must be resolved first -- Reason given: challenge - Existing challenge
must be resolved first.
      +challengeCanBeResolved - Current challenge is not ready to be resolved

      at expectException (node_modules/openzeppelin-test-helpers/src/expectRevert.js:20:30)
      at processTicksAndRejections (internal/process/next_tick.js:81:5)
```

Code Coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	72.06	100	100	
Everest.sol	100	73.44	100	100	
Registry.sol	100	100	100	100	
ReserveBank.sol	100	50	100	100	
contracts/abdk-libraries-**solidity/**	93.75	64.29	100	100	
ABDKMath64x64.sol	93.75	64.29	100	100	
contracts/lib/	75.22	39.58	75.93	75.63	
AddressUtils.sol	100	100	100	100	
Context.sol	50	100	66.67	33.33	25,26
EthereumDIDRegistry.sol	100	83.33	100	100	
Ownable.sol	81.82	50	85.71	83.33	57,58
SafeMath.sol	57.89	25	62.5	57.89	… 137,154,155
dai.sol	57.5	34.62	53.33	61.54	… 114,117,120
lib.sol	100	100	0	0	32,34
All files	**88.31**	**59.23**	**84.71**	**88.58**	

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure smart contracts at scale using computer-aided reasoning tools, with a mission to help boost adoption of this exponentially growing technology.

Quantstamp's team boasts decades of combined experience in formal verification, static analysis, and software verification. Collectively, our individuals have over 500 Google scholar citations and numerous published papers. In its mission to proliferate development and adoption of blockchain applications, Quantstamp is also developing a new protocol for smart contract verification to help smart contract developers and projects worldwide to perform cost-effective smart contract security audits.

To date, Quantstamp has helped to secure hundreds of millions of dollars of transaction value in smart contracts and has assisted dozens of blockchain projects globally with its white glove security auditing services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Finally, Quantstamp's dedication to research and development in the form of collaborations with leading academic institutions such as National University of Singapore and MIT (Massachusetts Institute of Technology) reflects Quantstamp's commitment to enable world-class smart contract innovation.