Quantstamp Contract Security Certificate

June 24th 2020 — Quantstamp Verified

xDAI STAKE Token Distribution

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

Executive Summary

Type	Token Bridge
Auditors	Sebastian Banescu, Senior Research Engineer Jan Gorzny, Blockchain Researcher Kevin Feng, Software Engineer
Timeline	2020-06-04 through 2020-06-15
EVM	Muir Glacier
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README.md STAKE Token Distribution Docs POSDAO White Paper STAKE (formerly $DPOS) Token Incentives and Distribution Added 22-06-2020: STAKE Distribution Added 23-06-2020: Token release schedule
Documentation Quality	Medium
Test Quality	Medium

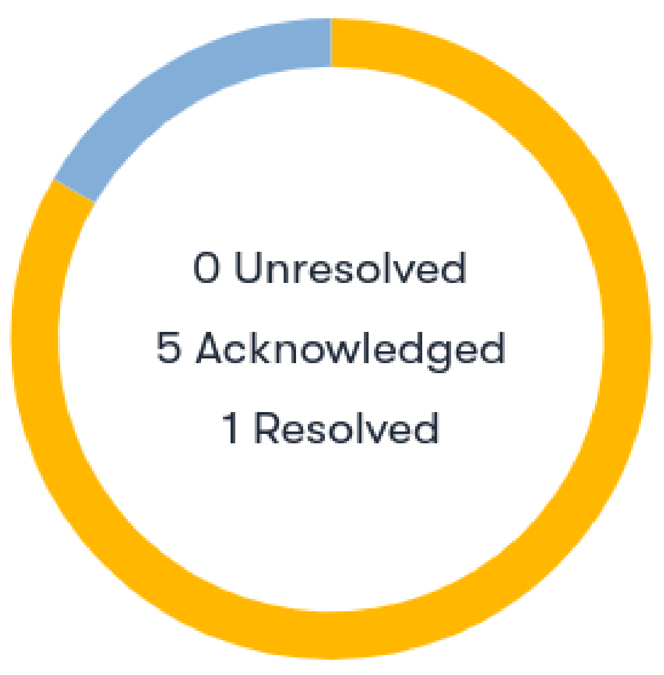
Source Code

Repository	Commit
stake-token	5da873b
stake-token	ad79abb
stake-token	7e6109b
stake-token	5a5ede0

Changelog

- 2020-06-15 - Initial report based on commit `5da873b`
- 2020-06-22 - Updated report based on commit `ad79abb`
- 2020-06-23 - Updated report based on commit `7e6109b`
- 2020-06-23 - Updated report based on commit `5a5ede0`

Total Issues	**6** (1 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	**2** (1 Resolved)
Informational Risk Issues	**2** (0 Resolved)
Undetermined Risk Issues	**2** (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.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

Quantstamp has performed a security assessment of the diff between v1.0.2 and v1.0.5 of the STAKE Token Distribution smart contracts. Several issues were found, ranging from low to undetermined severity. There are several other important deviations from the specification indicated in the "Adherence to Specification" section. The test suite in this repository does have a good number of assertions in each test case, however, tests only cover around 80% of the branches in the code on average, with as low as 39% of branches covered in `ERC677MultiBridgeToken.sol`. Branch coverage should be brought up to 100%, while keeping the number of assertions in the tests high. Quantstamp strongly recommends addressing all issues indicated in this audit report including those in the "Code Documentation" and "Adherence to Best Practices" sections, before deploying this project into production.
Update: The dev team has fixed 1 of our findings and has acknowledged the rest of the findings in this report. Additionally they have addressed the most important issues we have pointed out in the "Adherence to Specification" and "Code Documentation" sections. However, the vast majority of issues in the "Adherence to Best Practices" and "Automated Analyses" section have not been addressed. More importantly, the test suite has not been extended to improve the code coverage suggested in the first audit. We stress that testing is of utmost importance to avoid functional bugs.
Important note: Only the diff between v1.0.2 and v1.0.5 was in scope for this audit. For more information about our findings in v1.0.2 we refer the reader to the previous audit report.

ID	Description	Severity	Status
QSP-1	Gas Usage / `for` Loop Concerns	Low	Acknowledged
QSP-2	Replaying signatures from testnets possible	Low	Fixed
QSP-3	Block Timestamp Manipulation	Informational	Acknowledged
QSP-4	Privileged Roles and Ownership	Informational	Acknowledged
QSP-5	Misplaced input validation leads to incorrect behavior	? Undetermined	Acknowledged
QSP-6	Off-by-one error	? Undetermined	Acknowledged

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
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. 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:
   i. 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.
   ii. 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 Gas Usage / `for` Loop Concerns

Severity: Low Risk

Status: Acknowledged

File(s) affected: `MultipleDistribution.sol`

Related Issue(s): SWC-128

Description: Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. Even worse, some gas usage issues may prevent the contract from providing services entirely. For example, if a `for` loop requires too much gas to exit, then it may prevent the contract from functioning correctly entirely. It is best to break such loops into individual functions as possible.
If the list of `participants` is too large, any loop over its members, such as the loops in `addParticipants` and `removeParticipant` may fail.

Recommendation: Ensure that the list of is small enough to avoid gas limits. This requires adding a check for the number of participants based on the result of a gas analysis.

Update: The dev team has acknowledged that this issue is of no concern to them. The have added a code comment at the beginning of each of these 2 functions indicating that: "This function doesn't limit max gas consumption, so adding too many participants can cause it to reach the out-of-gas error."

QSP-2 Replaying signatures from testnets possible

Severity: Low Risk

Status: Fixed

File(s) affected: `ERC20Permittable.sol`

Related Issue(s): SWC-121

Description: The chain ID is hard-coded on `L32` as the value `1` and therefore the signatures that are used on test networks can be replayed on the Ethereum mainnet.

Recommendation: Add a `chainId` parameter to the constructor of the `ERC20Permittable` contract.

Update: This issue has been fixed as of commit `ad79abb`

QSP-3 Block Timestamp Manipulation

Severity: Informational

Status: Acknowledged

File(s) affected: `ERC20Permittable.sol`

Related Issue(s): SWC-116

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.
Miners can influence the value of `block.timestamp` by up to 900 seconds. There are multiple control-flow conditions that depend on the timestamp of the current block:

1. On `L59` in `ERC20Permittable.sol` the value of the timestamp influences whether or not the transfer of funds is allowed by comparing it to the expiration date.
2. On `L91` in `Distribution.sol` the value of the timestamp determines whether the installments for a given pool have started or not.
3. On `L230` in `Distribution.sol` the value of the timestamp determines the start of the distribution.
4. On `L353` in `Distribution.sol` the value of the timestamp determines the number of installments for a given pool.

Since the last 3 items contain an explicit comment for solium to disable the security warning due to the use of the `block.timestamp`, we assume those occurrences are acknowledged. However, we did not see such a comment on `L59` in `ERC20Permittable.sol.`

Recommendation: Add an explicit warning in the end-user documentation indicating that expiration timestamps can have a 900 second error. Add a comment on `L59` in `ERC20Permittable.sol` saying that the use of `block.timestamp` is acceptable.

Update: The explicit warning has been added in the comments of the source code.

QSP-4 Privileged Roles and Ownership

Severity: Informational

Status: Acknowledged

File(s) affected: `ERC677BridgeToken.sol`, `Distribution.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. There are multiple privileged roles in the audited codebase:

1. `distributionAddress`, `privateOfferingDistributionAddress`, `advisorsRewardDistributionAddress` have the priviledge to burn tokens by calling `ERC677BridgeToken.transferDistribution`.
2. The owner of `Distribution` can delay the time between the pre-initialization and the initialization by 90 days.
3. The bridge role is able to mint arbitrary tokens for any account.

Recommendation: Such roles and their privileged actions should be clearly documented in the documentation of the project such that they are clearly visible and accessible to end-users.

Update: The dev team has acknowledged this issue and has updated the `README.md` file to indicate that the bridge role is able to mint arbitrary tokens for any account.

QSP-5 Misplaced input validation leads to incorrect behavior

Severity: Undetermined

Status: Acknowledged

File(s) affected: `ERC677BridgeToken.sol`

Description: The `require` statements on `L217-220`, which are meant to check the value of the recipient address, are inside the `if`-statement, which means that the recipient address is only validated if the call to the `_contractFallback` function returns `false`. Otherwise, if the function returns `true` those checks are never performed, which would allow transferring funds to any of the 3 distribution addresses, in case their `onTokenTransfer` function would return `true`. According to the error messages in the `require` statements, this should not be allowed.

Moreover, if any of the `require` statements are `false`, the transaction reverts and the `ContractFallbackCallFailed` event is not emitted. This is contrary to the function description that says: "Emits an event when the callback failed." Note that there is no test in the current test suite which emits this event and check that it was correctly emitted.

Recommendation: Move the call to `_contractFallback` after `L220` and check its return value appropriately. The `ContractFallbackCallFailed` event should only be emitted if the return value is `false`. Also add a test case that checks if the event is emitted.

Update: The dev team has clarified that the placement of the `require` statements is intentional, because there is an exception, namely `MultipleDistribution.onTokenTransfer`, which only allows STAKE tokens to be sent to the `MultipleDistribution` contract by the `Distribution` contract. In this case, those `require` statements should be skipped to allow the transfer. They have updated the function specification (comment) to say that: "Used by the `transfer` and `transferFrom` functions. In case the recipient is a contract, tries to call its `onTokenTransfer` function. Reverts if `onTokenTransfer` fails and the recipient is a bridge/distribution contract." Note that this behavior is conditioned by the correct deployment of the distribution contracts and setting the `distributionAddress` to a contract of the type defined in `Distribution.sol` and the `privateOfferingDistributionAddress` and `advisorsRewardDistributionAddress` to contracts of the type defined in `MultipleDistribution.sol`. This is the case for the migration scripts in the current version of the repository (more importantly `2_deploy_contracts.js`), whose file signatures are indicated in the Appendix of this report.

Regarding the `ContractFallbackCallFailed` event, they have extended the function comment to indicate that the event is emitted only "when `onTokenTransfer` fails and the recipient is not a bridge/distribution contract." To further clarify the function purpose they have added the following sentence to the comments: [This function is] "Needed for reverting transfer to a bridge when the bridge doesn't accept tokens due to daily limitations, and to revert transfer to a distribution contract (Distribution or MultipleDistribution) if the contract doesn't accept tokens due to its limitations defined in `onTokenTransfer` function."

QSP-6 Off-by-one error

Severity: Undetermined

Status: Acknowledged

File(s) affected: `ERC20Permittable.sol`

Description: On `L116` the `require` statement compares `_now() <= _expiry`, that is if the input argument passed in as the expiry timestamp is greater or equal than the current block timestamp. This allows using the current timestamp as the expiry timestamp, which would most likely be a mistake by the user calling this function and should not be allowed.

Recommendation: Change the comparison to `_now() < _expiry`.

Update: The dev team has acknowledged that this is not an issue saying that the `expiry` is used to limit infinite approval in time so that `transferFrom` for the unlimited approval would only work before the `expiry` timestamp is reached.

Automated Analyses

Mythril

Mythril signaled an unprotected self-destruct in `Sacrifice.sol` which we marked as a false-positive.

Securify

Securify identified several issues including a few high severity issues such as re-entrancy and unrestricted writes to storage. After an inspection of these findings some were determined to be false positives, others were also identified by the manual analysis and others were added to the best practices section.

Slither

Slither indicated 128 issues in total. After filtering out the false positives, we marked the remaining issues as best practice concerns, which should be addressed:

1. Return values should not be ignored:

   - MultipleDistribution._withdraw(address) (MultipleDistribution.sol#260-272) ignores return value by token.transferDistribution(_recipient,currentShare) (MultipleDistribution.sol#269)
   - Distribution.preInitialize(address,uint256) (Distribution.sol#184-219) ignores return value by token.transferDistribution(poolAddress[PUBLIC_OFFERING],stake[PUBLIC_OFFERING]) (Distribution.sol#198)
   - Distribution.preInitialize(address,uint256) (Distribution.sol#184-219) ignores return value by

- token.transfer(poolAddress[PRIVATE_OFFERING],privateOfferingPrerelease) (Distribution.sol#199)
- Distribution.preInitialize(address,uint256) (Distribution.sol#184-219) ignores return value by token.transferDistribution(poolAddress[LIQUIDITY_FUND],liquidityFundDistribution) (Distribution.sol#202)
- Distribution.preInitialize(address,uint256) (Distribution.sol#184-219) ignores return value by token.transferDistribution(address(0),_initialStakeAmount) (Distribution.sol#204)
- Distribution.makeInstallment(uint8) (Distribution.sol#250-278) ignores return value by token.transfer(poolAddress[_pool],value) (Distribution.sol#272)
- Distribution.makeInstallment(uint8) (Distribution.sol#250-278) ignores return value by token.transferDistribution(poolAddress[_pool],value) (Distribution.sol#274)

2. Solidity naming conventions are not respected by the following:

- Variable MultipleDistribution.TOTAL_STAKE (MultipleDistribution.sol#58) is not in mixedCase
- Variable MultipleDistribution.POOL_NUMBER (MultipleDistribution.sol#59) is not in mixedCase
- Constant Distribution.supply (Distribution.sol#68) is not in UPPER_CASE_WITH_UNDERSCORES
- Constant ERC20Permittable.version (Token/ERC20Permittable.sol#13) is not in UPPER_CASE_WITH_UNDERSCORES
- Variable ERC20Permittable.DOMAIN_SEPARATOR (Token/ERC20Permittable.sol#16) is not in mixedCase

3. The visibility of many functions should be set to indicate exactly where they are allowed to be used:

- makeInstallment(uint8) should be declared external:
  - Distribution.makeInstallment(uint8) (Distribution.sol#250-278)
- onTokenTransfer(address,uint256,bytes) should be declared external:
  - Distribution.onTokenTransfer(address,uint256,bytes) (Distribution.sol#281-283)
- transferDistribution(address,uint256) should be declared external:
  - ERC677BridgeToken.transferDistribution(address,uint256) (Token/ERC677BridgeToken.sol#123-132)
- claimTokens(address,address) should be declared external:
  - ERC677BridgeToken.claimTokens(address,address) (Token/ERC677BridgeToken.sol#148-162)
- push(address,uint256) should be declared external:
  - ERC20Permittable.push(address,uint256) (Token/ERC20Permittable.sol#74-76)
- pull(address,uint256) should be declared external:
  - ERC20Permittable.pull(address,uint256) (Token/ERC20Permittable.sol#82-84)
- move(address,address,uint256) should be declared external:
  - ERC20Permittable.move(address,address,uint256) (Token/ERC20Permittable.sol#90-92)

Update: The dev team has acknowledged the issues above, however, none of them have been fixed.

Adherence to Specification

The implementation mostly seems to adhere to the specification. However, we have found several important parameters to be different in the implementation. The following deviations from the specification were identified:

1. The specification entitled "STAKE (formerly $DPOS) Token Incentives and Distribution" specifies that "The STAKE token has a fixed supply of 27,000,000. During the initial exchange offering (IEO), a limited 3.7% of the total STAKE supply (1,000,000) will be available to the public." The specification also provides a table with how values are distributed among different participants. This is not reflected on `L68` in `Distribution.sol`, where `supply = 8537500 ether;`, nor on `L130-135`:

```
stake[ECOSYSTEM_FUND] = 4000000 ether;
stake[PUBLIC_OFFERING] = 400000 ether;
stake[PRIVATE_OFFERING] = IMultipleDistribution(poolAddress[PRIVATE_OFFERING]).poolStake();
stake[ADVISORS_REWARD] = IMultipleDistribution(poolAddress[ADVISORS_REWARD]).poolStake();
stake[FOUNDATION_REWARD] = 699049 ether;
stake[LIQUIDITY_FUND] = 816500 ether;
```

Moreover, we were not able to map any of the rows from the table in the aforementioned specification to the following two named constants in the implementation: `ADVISORS_REWARD` (it seems this one is bundled with the foundation reward in the specification) and `LIQUIDITY_FUND`.
Update: FIXED This issue has been addressed in the new user documentation page about STAKE Distribution.

1. The `PUBLIC_OFFERING` pool does not seem to have a `cliff` value associated with it in the implementation. On `L198` in `Distribution.sol` it seems like 100% of the funds in this pool will be distributed at IEO. However in the specification it is said that: "50% released at IEO, 50% at day 84 (network upgrade date)". This is in contradiction with the `README.md` file of the repo which says: "The `preInitialize` function releases 100% of Public Offering, 100% of Liquidity Fund, and 25% of Private Offering tokens."
   Update: FIXED This issue has been addressed in the new user documentation page about STAKE Distribution.

2. The `FOUNDATION_REWARD` pool has the `valueAtCliff` set to 20% on `L155` in `Distribution.sol`, which does not match with the specification that indicates: "Released at day 336: 1,088,102 (10%)".
   Update: FIXED This issue has been addressed in the new user documentation page about Token release schedule.

3. The specification does not mention that only `ECOSYSTEM_FUND` or `FOUNDATION_REWARD` are allowed to change their pool addresses, which is implemented by the externally callable `changePoolAddress` function in `Distribution.sol`.
   Update: There is no mention of changing pool addresses on the new documentation page.

4. There are undefined terms in the code comments (e.g., "cliff"), which are likely those from other domains, but should be explicitly defined in the specification.
   Update: There is no definition of the term "cliff" on the new documentation page.

Code Documentation

The code generally has code comments, which are of great help for maintainability and auditing. The following issue were identified in the code comments:

1. There seems to be a typo on L211 of ERC677BridgeToken.sol: “callback” -> “fallback”
2. The following non-trivial function does not have any code comments withdraw in MultipleDistribution.sol.
   Update: FIXED Added comment to describe withdrawfunction.
3. The comments on L190-191 of ERC677BridgeToken.sol say that: "Allow sending tokens to address(0) by Distribution, PrivateOffering, or AdvisorsReward contract". This may be interpreted as the only use of this function by the Distribution, PrivateOffering, or AdvisorsReward contracts is for burning. However, that is not the case. The comment should be updated to indicate that burning is not the sole use by Distribution, PrivateOffering, or AdvisorsReward contracts.
   Update: FIXED Comment was updated.
4. The comment on L67 in Distribution.sol mentions “token” but the value on L68 is “ether”. The comment should be augmented with an indication as to why "ether" is used to avoid any confusion.
   Update: FIXED Comment was extended to indicate why "ether" is used.

Adherence to Best Practices

1. L17 in ERC20Permittable.sol contains commented out code that calls a keccak256 hash function on a string, which resembles the function signature for the permit function declared on L106 of the same file. However, the name of the function is spelled with an uppercase letter at the beginning of the string. Is this intentional?
   Update: The dev team has acknowledged that this is intentional.
2. Calling the keccak256 hash on L29-31 is not necessary. As a gas optimization those calls can be removed.
3. Event parameters should be indexed wherever possible. The following are occurrences of unindexed event parameters:
   - L40 in ERC677BridgeToken.sol
   - L18,22,28,34 in Distribution.sol
   - L18,23,28,38,51,56 in MultipleDistribution.sol
4. The constructor of the ERC677BridgeToken contract should be marked as internal instead of public, to better indicate that this is an abstract contract since the isBridge function does not have an implementation.
5. Typo on L55 in ERC677MultiBridgeToken.sol: "bridge isn't existed" -> "bridge doesn’t exist".
6. Add a dedicated burn function that should only be callable by the distributionAddress, privateOfferingDistributionAddress and advisorsRewardDistributionAddress. Otherwise, it is likely that funds are burned by mistake.
   Update: The dev team has acknowledged that this issue and added a clarifying comment inside the _superTranfer function.
7. Magic numbers should be replaced with named constants. The following are occurrences of such magic numbers in the code:
   - L101,104,105,107 in MultipleDistribution.sol.
   - L51 in ERC20Permissible.sol, namely uint256(-1).
   - L119 in ERC20Permissible.sol, namely "\x19\x01".
8. Gas optimizations in the preInitialize function:
   - Use tokensLeft[PUBLIC_OFFERING] = 0 on L207
   - Use tokensLeft[LIQUIDITY_FUND] = 0 on L209
   - Move L217 inside the if-statement on L203.
9. We suggest adding error messages as accurate as possible. The error message "not a contract address" on L72 of ERC677BridgeToken.sol is not as precise as it could be. Another example is "transfer failed" on L208 of ERC677BridgeToken.sol, where "transferFrom failed" would be more accurate.
10. Migrations.sol has an unlocked solidity version pragma solidity >=0.4.21 <0.6.0;
11. Consider changing the visibility of the internal _now() functions in Distribution.sol and ERC20Premittable.sol to private to indicate that this function should only be used in the current contract.
12. The _value parameter of the onTokenTransfer function in MultipleDistribution.sol should be checked to be greater than 0. Otherwise, the function would just waste gas.
13. The distributionAddress state variable of the MultipleDistribution contract should be initialized to 0 in the constructor to avoid any confusion.

Test Results

Test Suite Results

All 111 tests pass and tests seem to have a proper number of assert statements.

```
Contract: Distribution
  makeInstallment
     ✓ should make all installments (ECOSYSTEM_FUND) - 1 (31949ms)
     ✓ should make all installments (ECOSYSTEM_FUND) - 2 (time past more than cliff) (36519ms)
     ✓ should make all installments (ECOSYSTEM_FUND) - 3 (time past more than cliff + all installments) (396ms)
     ✓ should make all installments (FOUNDATION_REWARD) - 1 (27105ms)
     ✓ should make all installments (FOUNDATION_REWARD) - 2 (time past more than cliff) (23888ms)
     ✓ should make all installments (FOUNDATION_REWARD) - 3 (time past more than cliff + all installments) (275ms)
     ✓ should make all installments (PRIVATE_OFFERING) - 1 (33628ms)
     ✓ should make all installments (PRIVATE_OFFERING) - 2 (time past more than cliff) (17994ms)
     ✓ should make all installments (PRIVATE_OFFERING) - 3 (time past more than cliff + all installments) (410ms)
     ✓ should make all installments (ADVISORS_REWARD) - 1 (31424ms)
     ✓ should make all installments (ADVISORS_REWARD) - 2 (time past more than cliff) (20319ms)
     ✓ should make all installments (ADVISORS_REWARD) - 3 (time past more than cliff + all installments) (310ms)
     ✓ cannot make installment if not initialized (1116ms)
     ✓ cannot make installment for wrong pool (218ms)
```

```
      ✓ should revert if no installments available (196ms)

  Contract: Distribution
    constructor
      ✓ should be created (187ms)
      ✓ cannot be created with wrong values (570ms)
    preInitialize
      ✓ should be pre-initialized (514ms)
      ✓ cannot be pre-initialized with not a token address (55ms)
      ✓ cannot be pre-initialized twice (251ms)
      ✓ cannot be pre-initialized with wrong token (101ms)
      ✓ should fail if not an owner (44ms)
      ✓ cannot be pre-initialized if Private Offering or Advisors Reward participants are not finalized (1108ms)
    initialize
      ✓ should be initialized (154ms)
      ✓ cannot be initialized if not pre-initialized (337ms)
      ✓ cannot be initialized twice (90ms)
      ✓ should fail if not an owner first 90 days (39ms)
      ✓ can be initialized by anyone after 90 days (125ms)
    changePoolAddress
      ✓ should be changed (115ms)
      ✓ should fail if wrong pool (68ms)
      ✓ should fail if not authorized (101ms)
      ✓ should fail if invalid address (55ms)
    onTokenTransfer
      ✓ should fail (not allowed) (483ms)

  Contract: MultipleDistribution
    constructor
      ✓ should fail if wrong number of pool (342ms)
    addParticipants
      ✓ should be added (103ms)
      ✓ should be added 50 participants (1646ms)
      ✓ cannot be added with wrong values (504ms)
      ✓ cannot be added if finalized (247ms)
      ✓ should be added and validated (250 participants) (6580ms)
      ✓ should fail if not an owner (55ms)
      ✓ should be added after 1 participant has been removed (230ms)
      ✓ should be added after all participants have been removed (304ms)
      ✓ should add removed participant (269ms)
    editParticipant
      ✓ should be edited (154ms)
      ✓ should fail if invalid address (128ms)
      ✓ should fail if not an owner (156ms)
      ✓ should fail if finalized (275ms)
      ✓ should fail if participant does not exist (255ms)
      ✓ should fail if new stake is 0 (116ms)
      ✓ should fail if wrong sum of stakes after editing (140ms)
    removeParticipant
      ✓ should be removed (247ms)
      ✓ should fail if invalid address (172ms)
      ✓ should fail if not an owner (151ms)
      ✓ should fail if finalized (217ms)
      ✓ should fail if participant does not exist (149ms)
      ✓ should remove some participants (10892ms)
      ✓ should remove all participants (290ms)
    finalizeParticipants
      ✓ should be finalized (196ms)
      ✓ should be finalized with sum of stakes which is less than the whole pool stake (138ms)
      ✓ should be finalized with 250 participants (6678ms)
      ✓ cannot be finalized twice (160ms)
      ✓ should fail if not an owner (99ms)
      ✓ should be finalized after editing and removing (254ms)
    initialize
      ✓ should be initialized
      ✓ should fail if sender is not a distribution address
      ✓ cannot be initialized twice (70ms)
      ✓ should fail if not finalized (126ms)
    setDistributionAddress
      ✓ should be set (52ms)
      ✓ cannot be set twice (68ms)
      ✓ should fail if not an owner (44ms)
      ✓ should fail if not the Distribution contract address (426ms)
    withdraw
      ✓ should be withdrawn (902ms)
      ✓ should be withdrawn 10 times by 20 participants (28158ms)
      ✓ should be withdrawn in random order (3098ms)
      ✓ cannot be withdrawn by not participant (853ms)
      ✓ cannot be withdrawn when no tokens available (940ms)
    burn
      ✓ should be burnt (924ms)
      ✓ should be burnt after withdrawals (3841ms)
      ✓ cannot be burnt by not an owner (962ms)
      ✓ cannot be burnt if zero address stake is zero (1091ms)
      ✓ cannot be burnt when no tokens available (1543ms)
    onTokenTransfer
      ✓ should be called (55ms)
      ✓ should fail if "from" value is not the distribution contract
      ✓ should fail if caller is not the token contract (54ms)

  Contract: Token
    constructor
      ✓ should be created (607ms)
      ✓ should fail if invalid address (828ms)
    addBridge
      ✓ should add (87ms)
      ✓ should fail if invalid or wrong address (98ms)
      ✓ should fail if not an owner
    transferAndCall
      ✓ should transfer and call (118ms)
      ✓ should fail if wrong custom data (99ms)
      ✓ should fail if recipient is bridge, Distribution, or MultipleDistribution contracts (367ms)
    transfer
      ✓ should transfer (60ms)
      ✓ should fail if recipient is bridge, Distribution, or MultipleDistribution contracts (367ms)
    transferFrom
      ✓ should transfer (106ms)
      ✓ should fail if recipient is bridge, Distribution, or MultipleDistribution contracts (543ms)
    move
      ✓ should transfer (113ms)
      ✓ should fail if recipient is bridge, Distribution, or MultipleDistribution contracts (582ms)
    mint
      ✓ should mint (172ms)
      ✓ should fail if sender is not bridge
    permit
      ✓ should permit (369ms)
      ✓ should fail when invalid expiry (170ms)
      ✓ should consider expiry (339ms)
      ✓ should disallow unlimited allowance (394ms)
      ✓ should fail when invalid signature or parameters (303ms)
    claimTokens
      ✓ should claim tokens (100ms)
      ✓ should fail if invalid recipient (80ms)
      ✓ should fail if not an owner
      ✓ should claim eth (183ms)
      ✓ should claim eth to non-payable contract (220ms)
    renounceOwnership
      ✓ should fail (not implemented) (615ms)


  111 passing (6m)
```

Code Coverage

Test coverage is very low for files like `ERC677MultiBridgeToken.sol` and `ERC20.sol`, which contain a large part of the business logic for this repository. Also, the test coverage for other files like `ERC677BridgeToken.sol`, `ERC20Permittable.sol`, `Distribution.sol` and `MultipleDistribution.sol` should be increased. We strongly recommend that the test coverage for all files be increased to 100% branch coverage and that all tests contain a sufficiently large number of assertions to check that the output and side effects of function calls are as expected.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	98.42	93	90.63	98.46	
Distribution.sol	98.25	90.91	87.5	98.28	291,295
MultipleDistribution.sol	98.68	94.64	93.75	98.73	207
contracts/Token/	79.14	67.11	76.92	79.17	
ERC20.sol	60.61	41.67	61.54	60.61	… 202,233,234
ERC20Permittable.sol	85	91.67	57.14	85.71	82,90,148
ERC677BridgeToken.sol	98.15	82.35	100	98.21	227
ERC677MultiBridgeToken.sol	62.5	38.89	80	61.76	… 69,70,71,73
All files	**90.27**	**81.82**	**83.1**	**90.27**	

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
390f614d4634543fc18e0dcfcf01a32d83bea71c1b2c1ea9d7bea1fa3879ff49 ./2_deploy_contracts.js
b97c0ed0b3bbf8eef24d80214ed8aa8a1dac81f28b72397a04b776c45cb20b0e ./1_initial_migration.js
0636b43d4def3b59bdb068e5201948cb66747405ea4d3809e70acd3456b03ca5 ./contracts/IDistribution.sol
0dbf49e620b4ea66d5648e365222a285afd606d9684a6380ad3d4eb81754eaea ./contracts/IMultipleDistribution.sol
12a36a6757b7c660e97df684c2ae6e1e2dcdee9475868772c3adcbab92d24bce ./contracts/Distribution.sol
df24290ca011483abee2f487aa0dfe076cd763dbb2f0b89bc9401920d6f54ac2 ./contracts/Migrations.sol
aaa0d1959ff506834e0e8f83d0b8dc85e6190387737530e1c4f6eb8ac2147a1f ./contracts/MultipleDistribution.sol
858db8357a21912e5f1ddba68e21b5aa49b37b82e9101274f571df53e306b3bc ./contracts/Token/IERC677MultiBridgeToken.sol
ced5b0af26d41e19d7f01e1b2c97ebb58115da210baa77cb6aa75be4b1e88981 ./contracts/Token/ERC677MultiBridgeToken.sol
c83ebbf2c6f42321b1899a5d48287b151eb42681017e55454375dfc127631a47 ./contracts/Token/ERC20Permittable.sol
86357e1c6562ebe1d26e3c00228e981f71a3cdbd174e5c00de919bd13933273d ./contracts/Token/Sacrifice.sol
ca2cac64244fbd071585be1b149194cd8447c577d0970d9ab49207dba09d5f10 ./contracts/Token/ERC20.sol
4317d20d8613914c85d61410e59d75ce97464bf480c44743c1295ddd46a25624 ./contracts/Token/ERC677BridgeToken.sol
70db336a5a71e590d0b79848078db1242a98c15ed8df86a0b7ad91a018d419da ./contracts/mocks/RecipientMock.sol
603fa8993f386279b1af7f20d2673f29b053ba6f6d4d48294ccaf05c561c0466 ./contracts/mocks/BridgeMock.sol
793f1266a1f5656c10f120ec0d0d7de08490ecc27b3952d3dc9aa2477cc582f7 ./contracts/mocks/TokenMock.sol
57a7d1654acd402a4cd9e6ba0785d1648547bcc8286b25d7b98f4b0c0bd0e76f ./contracts/mocks/EmptyContract.sol
b6812b8c4c017ac40103b1e7010306de4a0754fc1bd261919d24424998520c8d ./contracts/mocks/DistributionMock.sol
3d94a2e0d3d8730bc2ada42f193514c8e905224fea5a42f50d5288165590bd8d ./contracts/mocks/ERC677MultiBridgeTokenMock.sol
fe9f4feb52a18c93d4d547695461fcd2d4bfe6825f5841b0244b8b6b50276585 ./contracts/mocks/MultipleDistributionMock.sol
b24421f21f58545b7c942c3346f8e6cba13d4311838d446f4595f7d8dc2a3803 ./contracts/mocks/BridgeTokenMock.sol
```

Tests

```
fcac43eb621c829d977479a0800231c5bf4bbc8899cb36f74e5af2bff25baae4 ./test/token.test.js
8124784438f99df5a191df25fc93b0a6d2e4aec4c1bcf70da3705f43df1e540e ./test/distribution.installments.test.js
642d711977166080d870c4aa6a4b68c3437b25f90374bc2d8bb7f8a6579cc2eb ./test/multipleDistribution.test.js
8c83c68fc0def8263b41f322405f33d440856ce8eb26a6cf903220482acb46c7 ./test/distribution.test.js
7498f5e9063e1de17028e1481f48d088119b6157d82d031c639c2a2a4a9a3969 ./test/utils/eip712.sign.permit.js
e595ef98f7f976a942e9ad8a398f463332f4659b20eb0b8b6dbce263b553566f ./test/utils/constants.js
```

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.