# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer**: Derivable
**Date**: 20 September, 2023

## Document

| | |
|---|---|
| **Name** | Smart Contract Code Review and Security Analysis Report for Derivable |
| **Approved By** | Paul Fomichov \| Lead Solidity SC Auditor at Hacken OÜ |
| **Tags** | ERC1155 token, ERC20 token; DEX |
| **Platform** | EVM |
| **Language** | Solidity |
| **Methodology** | Link |
| **Website** | - |
| **Changelog** | 14.08.2023 - Initial Review<br>20.09.2023 - Second Review |

## Table of contents

## Introduction

Hacken OÜ (Consultant) was contracted by Derivable (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

## System Overview

The Derivable project audit consists of different parts.

Derivable pool is a liquidity pool of perpetual derivatives for a single leverage. The participants of this system are long traders, short traders, and liquidity providers.

The ERC1155Maturity module implements an ERC1155 token with the additional functionality of adding a soft lock mechanism for newly minted tokens. These tokens gain their full functionality after the maturing time has elapsed.

The Shadow Token module is an extension of ERC-1155 that allows any of its IDs to deploy a Shadow ERC-20 token with its contract address for DeFi composability.

The files in the scope:

- **ERC1155Maturity.sol** - The ERC1155 modification which allows for the maturing functionality.
- **IERC1155Maturity.sol** - The interface for ERC1155Maturity.sol
- **TimeBalance.sol** - The helper library for handling the maturing times of ERC1155Maturity.sol
- **MetaProxy.sol** - The helper library for deploying shadow ERC20 contracts.
- **Shadow.sol** - The ERC20 contract that is deployed by individual IDs of ERC1155Maturity tokens.
- **ShadowFactory.sol** - The extension of ERC1155Maturity.sol which can deploy shadow tokens.
- **IERC1155Supply.sol** - The interface for returning the totalSupply of ERC1155 tokens.
- **IShadowFactory.sol** - The interface for the ShadowFactory.sol
- **PoolFactory.sol** - Factory contract to deploy Derivable pool using ERC-3448.
- **PoolLogic.sol** - The mathematic and finance logic of Derivable pool.
- **PoolBase.sol** - The base implementation of Derivable pool.
- **Fetcher.sol** - Interacts with oracles to obtain prices for swaps.
- **Token.sol** - A single ERC-1155 token shared by all Derivable pools which is also a ShadowFactory extension.
- **Constants.sol** - The contract for storing constant values.

- **Storage.sol -** The contract responsible for some of the variables in PoolBase.sol.

## Privileged roles

- Users: Can trade long/short positions, and become liquidity providers.
- Pools: Can mint, burn its specific tokens that are used in the pools. A pool has this functionality on its own tokens only.
- Descriptor Setter: Sets a new descriptor or sets the address for ITokenDescriptor, which is responsible for storing the metadata of a token.

## Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

### Documentation quality

The total Documentation Quality score is **4** out of **10**.

- Functional requirements are partially missed.
- Technical description is partially provided.
- NatSpec should be improved.
- Description of the development environment is missing.

### Code quality

The total Code Quality score is **8** out of **10**.

- The development environment is configured.
- Soldity Style Guides violations are present.
- Some variables contain bad naming.

### Test coverage

Code coverage of the project is **100%** (branch coverage).

- Deployment and basic user interactions are covered with tests.

### Security score

As a result of the audit, the code contains **1** medium, and **1** low severity issues. The security score is **9** out of **10**.

All found issues are displayed in the “Findings” section.

### Summary

According to the assessment, the Customer's smart contract has the following score: **8.3**. The system users should acknowledge all the risks summed up in the risks section of the report.

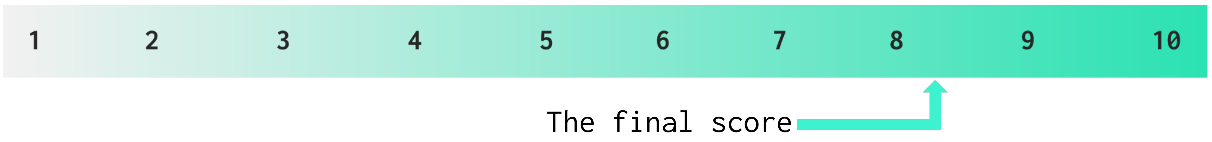

*Table. The distribution of issues during the audit*

| Review date | Low | Medium | High | Critical |
|---|---|---|---|---|
| 14 August 2023 | 1 | 3 | 1 | 0 |
| 20 September 2023 | 1 | 1 | 0 | 0 |

## Risks

- There are no additional risks.

## Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

| Item | Description | Status | Related Issues |
|---|---|---|---|
| **Default Visibility** | Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously. | Passed | |
| **Integer Overflow and Underflow** | If unchecked math is used, all math operations should be safe from overflows and underflows. | Passed | |
| **Outdated Compiler Version** | It is recommended to use a recent version of the Solidity compiler. | Passed | |
| **Floating Pragma** | Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. | Passed | |
| **Unchecked Call Return Value** | The return value of a message call should be checked. | Passed | |
| **Access Control & Authorization** | Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users. | Passed | |
| **SELFDESTRUCT Instruction** | The contract should not be self-destructible while it has funds belonging to users. | Not Relevant | |
| **Check-Effect-Interaction** | Check-Effect-Interaction pattern should be followed if the code performs ANY external call. | Failed | M03 |
| **Assert Violation** | Properly functioning code should never reach a failing assert statement. | Passed | |
| **Deprecated Solidity Functions** | Deprecated built-in functions should never be used. | Passed | |
| **Delegatecall to Untrusted Callee** | Delegatecalls should only be allowed to trusted addresses. | Passed | |
| **DoS (Denial of Service)** | Execution of the code should never be blocked by a specific contract state unless required. | Passed | |

| | | | |
|---|---|---|---|
| **Race Conditions** | Race Conditions and Transactions Order Dependency should not be possible. | Passed | |
| **Authorization through tx.origin** | tx.origin should not be used for authorization. | Not Relevant | |
| **Block values as a proxy for time** | Block numbers should not be used for time calculations. | Not Relevant | |
| **Signature Unique Id** | Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification. | Not Relevant | |
| **Shadowing State Variable** | State variables should not be shadowed. | Passed | |
| **Weak Sources of Randomness** | Random values should never be generated from Chain Attributes or be predictable. | Not Relevant | |
| **Incorrect Inheritance Order** | When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order. | Passed | |
| **Calls Only to Trusted Addresses** | All external calls should be performed only to trusted addresses. | Passed | |
| **Presence of Unused Variables** | The code should not contain unused variables if this is not justified by design. | Passed | |
| **EIP Standards Violation** | EIP standards should not be violated. | Passed | |
| **Assets Integrity** | Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract. | Passed | |
| **User Balances Manipulation** | Contract owners or any other third party should not be able to access funds belonging to users. | Passed | |
| **Data Consistency** | Smart contract data should be consistent all over the data flow. | Passed | |

| | | | |
|---|---|---|---|
| **Flashloan Attack** | When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used. Contracts shouldn't rely on values that can be changed in the same transaction. | Passed | |
| **Token Supply Manipulation** | Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer. | Passed | |
| **Gas Limit and Loops** | Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit. | Passed | |
| **Style Guide Violation** | Style guides and best practices should be followed. | Failed | I01 |
| **Requirements Compliance** | The code should be compliant with the requirements provided by the Customer. | Passed | |
| **Environment Consistency** | The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code. | Passed | |
| **Secure Oracles Usage** | The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles. | Passed | |
| **Tests Coverage** | The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested. | Passed | |
| **Stable Imports** | The code should not reference draft contracts, which may be changed in the future. | Passed | |

# Findings

## ■■■■ Critical

No critical severity issues were found.

## ■■■ High

### H01. Requirements Violation

| | |
|---|---|
| **Impact** | Medium |
| **Likelihood** | High |

The implementation of the system does not adhere to the technical requirements.

It is mentioned in the documentation for ERC1155Maturity.sol that the balance size is decreased from *uint256* to *uint192* and an additional *uint32* is added for the maturity time value. However, the code is implemented in such a way that the decreased balance value is not *uint192,* but *uint224*.

This means that the implementation may not meet the intended purpose or design of the system, which could result in major issues down the line.

**Path:** ./README.md : ERC1155Maturity.sol, TimeBalance.sol

**Recommendation**: Fix the mismatch between the code and the requirements.

**Found in:** fba1423

**Status**: Fixed (Revised commit: 71e5d24)

## ■■ Medium

### M01. EIP Standards Violation

| | |
|---|---|
| **Impact** | Low |
| **Likelihood** | High |

In the Shadow.sol implementation of ERC20 tokens, the *decreaseAllowance()* call does not decrease the allowance of addresses that were given maximum approvals.

This is a violation of the ERC20 standard since this implementation the only way to decrease the allowances of maximum approval given addresses is through the *approve()* function.

**Path:** ./contracts/Shadow.sol : decreaseAllowance()

**Recommendation**: Follow the EIP standards.

**Found in:** b255de3

**Status**: Fixed (Revised commit: a67bd5d)

### M02. Unfinalized Code

| | |
|---|---|
| **Impact** | Medium |
| **Likelihood** | Medium |

The provided code should be implemented in the full logic of the project. Since any missing parts, TODOs, or drafts can change in time, the robustness of the audit cannot be guaranteed.

Incomplete code impacts on project reliability and makes it harder to evaluate project security.

**Path:** ./contracts/PoolLogic.sol

**Recommendation**: Remove draft code/commented code parts or implement them.

**Found in:** d68cf8f

**Status**: Fixed (Revised commit: 695dde1)

### M03. Check-Effects Interaction Violation

| | |
|---|---|
| **Impact** | Medium |
| **Likelihood** | Medium |

In the contracts, during the functions execution, some state variables are updated after the external calls, which is against best practices.

This may lead to reentrancies, race conditions, and denial of service vulnerabilities during the implementation of new functionality.

**Path:** ./contracts/PoolBase.sol : init()

**Recommendation**: Common best practices should be followed, functions should be implemented according to the Check-Effect-Interaction pattern. If not possible, the nonReentrant modifier can be used.

**Found in:** fba1423

**Status**: Reported (The *init()* call has Check-Effects-Interaction Pattern violation.)

## Low

### L01. Missing Zero Address Validation

| | |
|---|---|
| **Impact** | Low |
| **Likelihood** | Medium |

Address parameters are being used without checking against the possibility of 0x0.

This can lead to unwanted external calls to 0x0.

**Paths:**

./contracts/PoolBase.sol : constructor()
./contracts/PoolFactory.sol : constructor()
./contracts/PoolLogic.sol : constructor()
./contracts/Token.sol : constructor()
./contracts/Shadow.sol : constructor()

**Recommendation**: Implement zero address checks.

**Found in:** d68cf8f, b255de3

**Status**: Reported (Token constructor() does not check for zero addresses.)

## Informational

### I01. Solidity Style Guide Violation

Contract readability and code quality are influenced significantly by adherence to established style guidelines. In Solidity programming, there exist certain norms for code arrangement and ordering. These guidelines help to maintain a consistent structure across different contracts, libraries, or interfaces, making it easier for developers and auditors to understand and interact with the code.

The suggested order of elements within each contract, library, or interface is as follows:

- Type declarations
- State variables
- Events
- Modifiers
- Functions

Functions should be ordered and grouped by their visibility as follows:

- Constructor
- Receive function (if exists)
- Fallback function (if exists)
- External functions

- Public functions
- Internal functions
- Private functions

Within each grouping, view and pure functions should be placed at the end.

Furthermore, following the Solidity naming convention and adding NatSpec annotations for all functions are strongly recommended. These measures aid in the comprehension of code and enhance overall code quality.

**Paths:**

./contracts/ShadowFactory.sol
./contracts/PoolBase.sol
./contracts/Token.sol

**Recommendation**: Consistent adherence to the official Solidity style guide is recommended. This enhances readability and maintainability of the code, facilitating seamless interaction with the contracts. Providing comprehensive NatSpec annotations for functions and following Solidity's naming conventions further enrich the quality of the code.

**Found in:** d68cf8f, b255de3

**Status**: Reported - some variables do not follow the naming convention:

- Token → s_descriptor, s_descriptorSetter.

### I02. Floating Pragma

The project uses floating pragmas ^0.8.0.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version, which may include bugs that affect the system negatively.

**Paths:** all contracts

**Recommendation**: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

**Found in:** b255de3, d68cf8f, fba1423

**Status**: Fixed (Revised commits: 695dde1, a67bd5d, 71e5d24)

### I03. Non-Explicit Unit Size

The project often declares variables with type “uint”, without expliciting the exact unit size. The default size for uint is 256 bytes, but not having the unit size impacts code readability.

**Paths:**

./contracts/Fetcher.sol
./contracts/PoolBase.sol
./contracts/PoolLogic.sol
./contracts/Token.sol
./contracts/ERC1155Maturity.sol
./contracts/TimeBalance.sol
./contracts/MetaProxy.sol
./contracts/Shadow.sol
./contracts/ShadowFactory.sol

**Recommendation**: Declare all uint variables with explicit unit size.

**Found in:** d68cf8f, b255de3, fba1423

**Status**: Reported (ERC1155Maturity contains non-explicit size uint variables.)

### I04. Bad Require Messages

The project contains some non-explicit or non-explanatory error messages that lead the user to confusion. Errors should be well written and should be able to tell the user what is wrong/why the execution failed.

**Paths:**

./contracts/PoolBase.sol : init(), ensureStateIntegrity(), swap()
./contracts/PoolLogic.sol : _swap()
./contracts/libs/OracleLibrary.sol : consult()

**Recommendation**: Improve error messages in order to be explanatory and human-readable, for example:

- In PoolBase.init() : “AI”, “IP”, “ZP”, “BP”
- In PoolBase.ensureStateIntegrity() : “SI”
- In PoolBase.swap() : “BP”, “II”
- In PoolLogic._swap() : “SS”, “OA”, “OB”, “MI:R”, “MI:NR”, “MI:A”, “MI:NA”, “MI:NB”, “MS”, “MR:C”, “MR:A”, “MR:B”
- In OracleLibrary.consult : “BP”

**Found in:** d68cf8f

**Status**: Fixed (Revised commit: 695dde1)

### I05. Bad Variable Naming

The project contains some variables with non-explanatory names increasing code complexity and making it harder to read and maintain. Variables should have self-explanatory names and follow the official style guide.

**Paths:**

./contracts/PoolBase.sol
./contracts/subs/Storage.sol
./contracts/PoolLogic.sol
./contracts/TimeBalance.sol

**Recommendation**: Improve variable names.

- In PoolBase.sol : State.R, State.a, State.b, Slippable.xk, Slippable.R, Slippable.rA, Slippable.rB
- In Storage.sol : s_i, s_a, s_f, s_b
- In PoolLogic.sol : xk, rA, rB
- In TimeBalance.sol : t, b, x, xt, yt, yb, xb, zb,

**Found in:** d68cf8f, fba1423

**Status**: Mitigated (The documentation of the variables increases readability.)

## Disclaimers

### Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

### Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

## Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

| Risk Level | High Impact | Medium Impact | Low Impact |
|---|---|---|---|
| **High Likelihood** | Critical | High | Medium |
| **Medium Likelihood** | High | Medium | Low |
| **Low Likelihood** | Medium | Low | Low |

### Risk Levels

**Critical**: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

**High**: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

**Medium**: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

**Low**: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

## Impact Levels

**High Impact**: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

**Medium Impact**: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

**Low Impact**: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

## Likelihood Levels

**High Likelihood**: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

**Medium Likelihood**: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

**Low Likelihood**: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

## Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

## Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

### Initial review scope

| | |
|---|---|
| **Repositories** | https://github.com/derivable-labs/derivable-core<br>https://github.com/derivable-labs/shadow-token<br>https://github.com/derivable-labs/erc1155-maturity |
| **Commits** | d68cf8f70dbc9b09e6e2d61682ecf506ef24c17f<br>b255de3bc075d8bc4d7c76d87b6d5bde51c05476<br>fba14233ceac63b88727f0ca573483260d1a45bb |
| **Requirements** | Link<br>Link<br>Link |
| **Technical Requirements** | Link<br>Link<br>Link |
| **Contracts** | [derivable-core]<br>File: contracts/Fetcher.sol<br>SHA3: 5e5bddb231904c1c194a5266af4033456dc1e3edb8367d91e14e10ea8fe8222e<br><br>File: contracts/PoolBase.sol<br>SHA3: 96de5563bc73103d5bdc064ca74c74f6a0d6ae1d1fb6c2553a717725acfbf0c0<br><br>File: contracts/PoolFactory.sol<br>SHA3: f8249a6d62d0e9e67e984a76bdf6c0b7d613ad913f46a44da92e5fd233e4a22e<br><br>File: contracts/PoolLogic.sol<br>SHA3: 77a14604cee1f9b154e0120e0f5cd4d6075dacf33bb1d88d10d815f5cedbc0d7<br><br>File: contracts/Token.sol<br>SHA3: 47d7769e30ab90156fddf5186cadb6a2cd5ec97ecf7a3c21800c1b4e9eaa5996<br><br>File: contracts/libs/OracleLibrary.sol<br>SHA3: 50a517ab7a49b96cb32f83979d03cff32e95fe3438ce71682306f838c0d29149<br><br>[shadow-token]<br>File: contracts/MetaProxy.sol<br>SHA3: b0af1d721de3a05086fcdc1f727813c0476f41e6564af1ef13c46b0c30556fe8<br><br>File: contracts/Shadow.sol<br>SHA3: 7f9e0aad9dfa639a381ff49087d47997d9faf060d6566e0ead63e17f2bf887aa<br><br>File: contracts/ShadowFactory.sol<br>SHA3: 5e6705a7309e366630e3257bd235cc8d71484ba15acb5330a0046cd3faa104a8<br><br>[erc1155-maturity]<br>File: contracts/token/ERC1155/ERC1155Maturity.sol<br>SHA3: 0b5a95f86f29f8ec4b09bf1ea0dde6ceb052694787fad49d1eef2c54f4af8611<br><br>File: contracts/token/ERC1155/libs/TimeBalance.sol<br>SHA3: 3d9771718556c36afe700931736bab6e6aa77a24b38455ccae24e2d07d22f973 |

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

## Second review scope

| | |
|---|---|
| **Repositories** | https://github.com/derivable-labs/derivable-core<br>https://github.com/derivable-labs/shadow-token<br>https://github.com/derivable-labs/erc1155-maturity |
| **Commits** | 695dde103890f5be0763c742cccf901d6a4cf6cd<br>a67bd5d59af3491c7b3b51c9c09367a308313121<br>71e5d249cdf871162860ccf5c1bbe54e4a17b1ac |
| **Requirements** | Link<br>Link<br>Link |
| **Technical Requirements** | Link<br>Link<br>Link |
| **Contracts** | [derivable-core]<br>File: contracts/Fetcher.sol<br>SHA3: b30d651385f9d34860a17f5a6facaacab6a52a274c03f7bb20a21861491a859c<br><br>File: contracts/PoolBase.sol<br>SHA3: 0ed2c6b61599af18e9f6e0e5dd45328a895eedc28c167ae72df6998d7e867723<br><br>File: contracts/PoolFactory.sol<br>SHA3: 34b7ee73bb102ef961fe1018f7e0507012b980789e5ce6f85ab0ea6e3194a1b3<br><br>File: contracts/PoolLogic.sol<br>SHA3: 4b5afe3a323bfe9779a4a8ffe89c99f9737b0deea83c5842ad7b8385649187b2<br><br>File: contracts/Token.sol<br>SHA3: 619e7071020df79f2ed331190fa1392edb5d2d546650af756bf7f355a47dba3d<br><br>File: contracts/subs/Storage.sol<br>SHA3: f139330020e54483eea584ae7e88c4a1971ba3d410b410239e7a04ace1b2f16b<br><br>File: contracts/subs/Constants.sol<br>SHA3: a77bbd814fd20020d6ac0876286a7e98e7bace96e1b7a2e9c393a0f09f641058<br><br>[shadow-token]<br>File: contracts/MetaProxy.sol<br>SHA3: af0071a3492fa557b3589fdbe43daa8fc5ec59625bdc97eeb78b4aa806613a4b<br><br>File: contracts/Shadow.sol<br>SHA3: 4756feb34e4535fe6604b1d825008ca6af52e6934d0318a0b7a2ff9ca4fd0638<br><br>File: contracts/ShadowFactory.sol<br>SHA3: 57deef60a65f2b53c712074b72010a150040f147f65858776775e3b3b5c4a4c9<br><br>[erc1155-maturity]<br>File: contracts/token/ERC1155/ERC1155Maturity.sol<br>SHA3: 41662a2bbc5ae0a88bbbc228a2e06bbc567c9f4c30da5f2ae61fba955bb38671<br><br>File: contracts/token/ERC1155/libs/TimeBalance.sol<br>SHA3: b7cf90bf11de529f63c1d6445398b998310ea2e49fe8c30ec36c2ea277d8bb1d |