Report

v. 1.0

Customer

Enso

Smart Contract Audit

# EnsoToken

11th July 2025

Report prepared by

ABDK Consulting

# Contents


**1 Changelog** **3**

**2 Introduction** **4**

**3 Project scope** **5**

**4 Methodology** **6**

**5 Our findings** **7**

**6 Moderate Issues** **8**
CVF-1. FIXED . . . . . 8

**7 Recommendations** **9**
CVF-2. INFO . . . . . 9
CVF-3. FIXED . . . . . 9
CVF-4. FIXED . . . . . 9

# 1 Changelog

| # | Date | Author | Description |
|---|---|---|---|
| 0.1 | 11.07.25 | A. Zveryanskaya | Initial Draft |
| 0.2 | 11.07.25 | A. Zveryanskaya | Minor revision |
| 1.0 | 11.07.25 | A. Zveryanskaya | Release |

# 2 Introduction



The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

A network that encompasses all blockchains and smart contracts into one network, enabling developers to focus solely on their product, community, and distribution rather than the intricacies of blockchain development by integrating 1 tool: Enso.

# 3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

```
/
        EnsoToken.sol
```

# 4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Recommendations** contain code style, best practices and other suggestions.

# 5 Our findings

We provided the client with some recommendations.

| Moderate | Info | Fixed |
| --- | --- | --- |
| | 0 | 1 |

Fixed 1 out of 1 issues

# 6 Moderate Issues

## CVF-1 FIXED

- **Category** Unclear behavior
- **Source** EnsoToken.sol

**Description** Unchained initializers should be used here.

```
38  __ERC20_init(name, symbol);
    __ERC20Permit_init(name);
40  __ERC20Votes_init();
    __Ownable_init(owner);
    __UUPSUpgradeable_init();
```

# 7 Recommendations

## CVF-2 INFO

- **Category** Procedural
- **Source** EnsoToken.sol

**Description** Consider specifying as “^0.8.0” unless there is something special regarding this particular version.

**Client Comment** *We are limited to this due to dependencies.*

```
2 pragma solidity ^0.8.20;
```

## CVF-3 FIXED

- **Category** Procedural
- **Source** EnsoToken.sol

**Description** Declaring a top-level structure in a file named after a contract makes it harder navigating through code.

**Recommendation** Move the structure declaration into the contract or move it into a separate file.

**Client Comment** *Moved to a separate file.*

```
16 struct Distribution {
```

## CVF-4 FIXED

- **Category** Documentation
- **Source** EnsoToken.sol

**Description** It is a good practice to put a comment into an empty block to explain why the block is empty.

```
87 function _authorizeUpgrade(address) internal override onlyOwner { }
```

ABDK
Consulting

## About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

## Contact


**Email**

dmitry@abdkconsulting.com

**Website**

abdk.consulting

**Twitter**

twitter.com/ABDKconsulting

**LinkedIn**

linkedin.com/company/abdk-consulting