ABDK CONSULTING

CIRCUITS AUDIT

ZeroPool

libzeropool

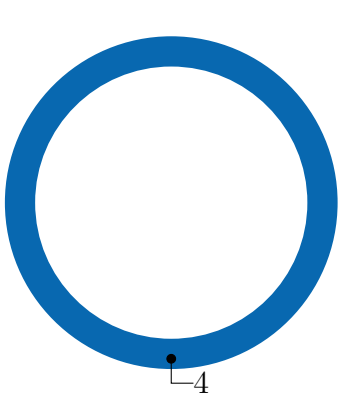
Rust

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
29th June 2022

We've been asked to review updates to 5 files in a Github repo. We found 4 minor issues.



Findings

ID	Severity	Category	Status
CVF-1	Moderate	Overflow/Underflow	Info
CVF-2	Minor	Suboptimal	Info
CVF-3	Minor	Suboptimal	Info
CVF-4	Minor	Suboptimal	Info

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	June 28, 2022	D. Khovratovich	Initial Draft
0.2	June 28, 2022	D. Khovratovich	Minor revision
1.0	June 29, 2022	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting 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.
We have reviewed the differences in the following files:

- src/circuit/account.rs
- src/circuit/tx.rs
- src/constants.rs
- src/native/account.rs
- src/native/tx.rs

2.1 About ABDK

ABDK Consulting, 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.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.3 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best 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 and that 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 is 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 that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check 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.

3 Detailed Results

3.1 CVF-1

- **Severity** Moderate
- **Status** Info
- **Category** Overflow/Underflow
- **Source** account.rs

Description Overflow is possible when calculating the sum.
Recommendation Consider handling it somehow or explaining why it is not possible.
Client Comment max(HEIGHT, BALANCE_SIZE_BITS, ENERGY_SIZE_BITS) is assumed small enough. Will not fix.

Listing 1:

```
29 +(self.i.as_num()+self.b.as_num()+self.e.as_num()).is_zero() &
   ↪ self.d.as_num().is_eq(poolid)
```

3.2 CVF-2

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** tx.rs

Description The "clone" calls for "eta" and "path" are redundant.
Recommendation Consdier removing them.
Client Comment Will not fix. Pointers are passed to c_nullfifier, and array of objects is used by c_poseidon.

Listing 2:

```
51 +[in_account_hash.clone(), eta.clone(), path.clone()].as_ref(),

57 +[in_account_hash.clone(), intermediate_hash].as_ref(),
```

3.3 CVF-3

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** tx.rs

Description The "clone" call for "in_account_hash" is redundant.
Recommendation Consider removing it.
Client Comment Will not fix. Pointers are passed to c_nullfifier, and array of objects is used by c_poseidon.

Listing 3:

```
57 +[in_account_hash.clone(), intermediate_hash].as_ref(),
```

3.4 CVF-4

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** tx.rs

Recommendation If this parameter is responsible for domain separation, consider making it nullifier-specific.
Client Comment Nullifiers are not stored in arbitrary sized Merkle trees, so there is no collision. If we decide to store nullifiers in such structures in future versions of the protocol, we will add nullifier specific parameters to the hash. Will not fix.

Listing 4:

```
58 params.compress(),
```