Version
v 1.0

Customer
NEBRA

08 / 29
2024

Specifications Audit. Part I

ABDK Consulting

Your guide in the world of Crypto

Contents

1 Changelog

#	Date	Author	Description
0.1	28.08.24	A. Zveryanskaya	Initial Draft
0.2	29.08.24	D. Khovratovich	Minor revision
1.0	29.08.24	A. Zveryanskaya	Release

2 Introduction

All modifications to this document are prohibited. Violators will be prosecuted to the full extent of the U.S. law.

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 ABDK Comments.

NEBRA is a team of ZK cryptographers and Ethereum devs focused on democratizing and unlocking use cases for on-chain ZKP. Lead by Shumo - founder of NEBRA, research partner of Manta, and a Computer Science Ph. D. , this project will modify the NEBRA Universal Proof Aggregation protocol so that the proofs are submitted via a private RPC. This could further reduced the gas cost when NEBRA is deployed to L2s. As a result, the proof submission cost of Worldcoin protocols will be drastically reduced.

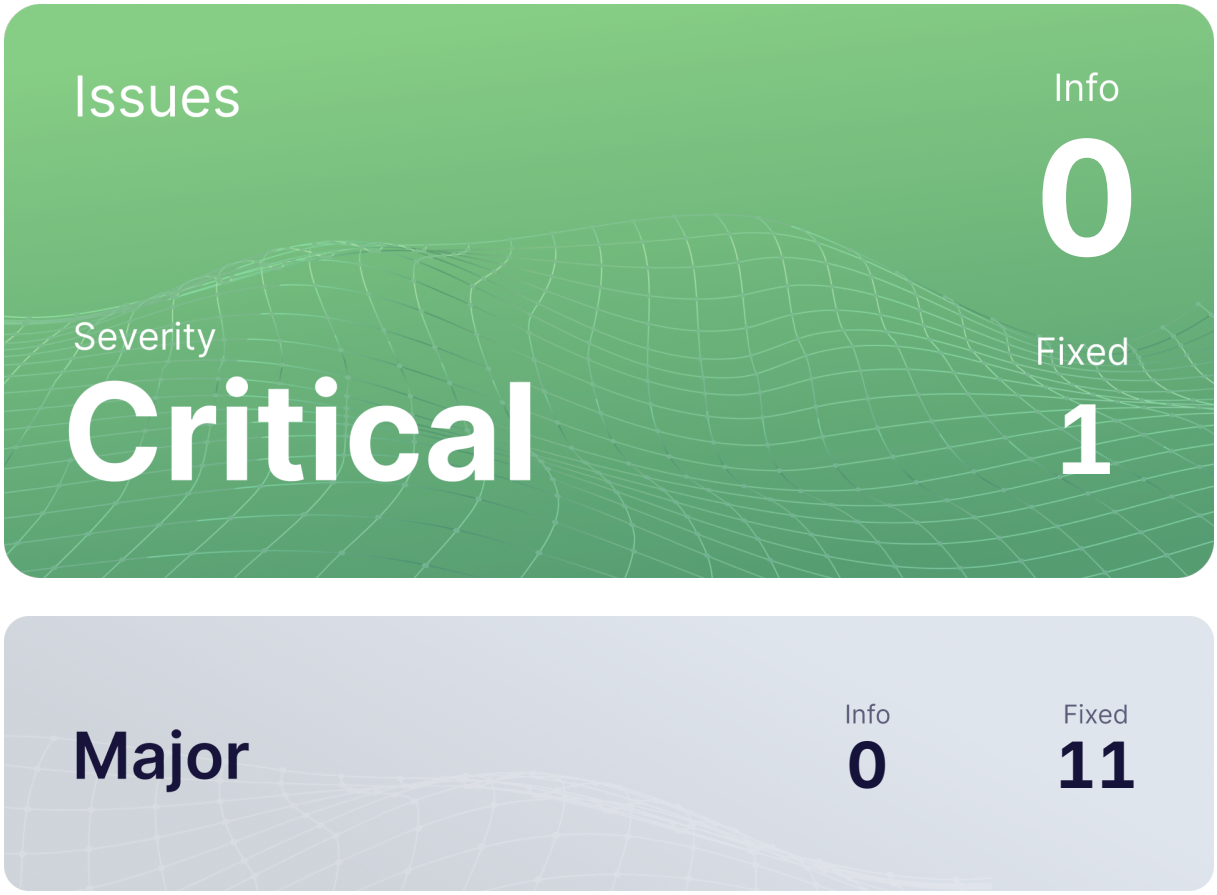
3 Project scope

We were asked to review the following documentation:

- Universal Outer Circuit Specification: Version 1, Version 2
- Variable length keccak circuit specification: Version 1, Version 2
- Universal Batch Verifier Specification: Version 1, Version 2

4 Our findings

We found 1 critical, 11 major, and a few less important issues. An attack scenario was provided to the client.



Fixed 12 out of 12 issues

5 Universal Outer Circuit Specification. Version 1

CVF-1

- **Severity** Moderate
- **Source** Overview

Specification Excerpt The outer circuit will take an outer batch of B_O UBV proofs with inner batch size B_I and a keccak proof and check:

ABDK Comment The specs for UBV and keccak proofs do not specify which proof system is used. Consider saying explicitly it is Groth16.

6 Variable length keccak circuit specification. Version 1

CVF-2

- **Severity** Moderate
- **Source** Configuration

Specification Excerpt The variable length keccak circuit is determined by the following parameters:

- `degree_bits`: The log_2 of the number of rows.
- `lookup_bits`: The log_2 of the number of rows allocated for lookup tables. Usually `degree_bits-1`.
- `inner_batch_size`: The number of Groth16 proofs to be verified in a batch.
- `outer_batch_size`: The number of batches.
- `num_public_inputs`: The maximum number of public inputs allowed in one application proof instance. We call this L.

ABDK Comment Some of these parameters are implementation details and are not used in the spec below. Consider refactoring.

CVF-3

- **Severity** Major
- **Source** Keccak phase 0

Specification Excerpt Flagged indices: The hashmap `flagged_indices` contains the positions of the cells with the output bytes. The key consists of a pair `(chunk_index, row_index)` and the value is a pair `(column_index, byte_index)`.

ABDK Comment This statement is confusing. Consider elaborating what the output of the keccak hash is.

CVF-4

- **Severity** Major
- **Source** Keccak phase 0

Specification Excerpt Flagged words: The hashmap `flagged_words` contains the positions of the cells with the input words. The key is the row index and the valuea is the column index.

ABDK Comment This statement is confusing. Please elaborate if this is a user-controlled input.

CVF-5

- **Severity** Moderate
- **Source** Sketch

Specification Excerpt Let's sketch what this function does. First, it converts bytes to bits and pads them to the next chunk length (which is 136 bytes or 1088 bits) with 10..01, with as many zeroes as needed.

ABDK Comment As we are not working on the bit level, consider explaining the padding with bytes. Consider also explaining what happens when less than two bits are needed for padding.
If this padding is assumed to be Ethereum-compatible, it should be said so. Also the padding explanation is ambiguous, as it doesn't specify how many zeros are between two ones. It should be the minimum number of zeros (possibly zero) that makes the length of a padded data to be a multiple of 1088 bits.

CVF-6

- **Severity** Moderate
- **Source** Bytes to keccak ...

Specification Excerpt

- Input: a vector of bytes $\vec{b}$ of length l
- Output: a vector of keccak padded words $\vec{w}$ of length

$$l' = 17 \left\lceil \frac{l+1}{136} \right\rceil,$$

  where $\lceil \cdot \rceil$ denotes the ceiling function.

Description:

- Convert $\vec{b}$ to bits.
- Pad them to chunk length (1088 bits) by adding the bit string 10..01.

ABDK Comment If this padding is assumed to be Ethereum-compatible, it should be said so. The padding explanation is ambiguous, as it doesn't specify how many zeros are between two ones. It should be the minimum number of zeros (possibly zero) that makes the length of a padded data to be a multiple of 1088 bits.

CVF-7

- **Severity** Major
- **Source** Step 1

Specification Excerpt For $i = 1, \ldots, M$, the circuit computes the byte decomposition of $\mathsf{C}_{\mathsf{ID}i}$ and $\overline{P}_i$, defined as the concatenations of the byte decompositions of $\mathsf{C}_{\mathsf{ID}i}$ and $\overline{P}_{ji}$ for $j = 1, \ldots, L$.
Each element $f \in \mathbb{F}_r$, where $f = \mathsf{C}_{\mathsf{ID}i}$ or $f = \overline{P}_{ji}$ for some j, is decomposed as follows:

- Decompose f into 32 bytes $b_k{}_{k=1}^{32}$ out of circuit (in big endian representation), and assign them as witnesses.
- Range check each byte $0 \le b_k < 2^8$.
- Enforce $f = \sum_{k=1}^{32} b_{32-k+1} 2^{8(k-1)}$.

ABDK Comment The range check for f (or the upper byte of each input field element) is missing. This problem was also found in the code.

CVF-8

- **Severity** Major
- **Source** Step 3

Specification Excerpt For $i = 1, \ldots, M$, this function computes the keccak query. We drop the index i below to simplify the notation.

ABDK Comment It is unclear which part of the circuit is actually computed here. Also, since the function is not just a simple fixed-input Keccak, the math description should be provided separately from the circuit, possibly using the Keccak inner permutation as a black box. Consider refactoring.

CVF-9

- **Severity** Moderate
- **Source** Step 3

Specification Excerpt

- We denote by $W_{\vec{b}}$ the word resulting from packing (as in the last step of Bytes to keccak padded words[1]) the bitstring $\vec{b}$. Assign the following constants:
  - $W_{00..00} \in \mathbb{F}_r$
  - $W_{00..01} \in \mathbb{F}_r$
  - $W_{10..00} \in \mathbb{F}_r$
  - $W_{10..01} \in \mathbb{F}_r$

ABDK Comment The role of these constants is unclear.

CVF-10

- **Severity** Moderate
- **Source** Step 3

Specification Excerpt

- Use the word position in the chunk (first, last or else) and the bitmask $\mathfrak{b}_{eq}(i)$ at that index to select a filler word, which encodes the keccak padding algorithm at that position.
- Use the bitmask $\mathfrak{b}_{\text{words}}(i)$ to select either w'_i or the filler word computed above.

ABDK Comment The functions $\mathfrak{b}_{eq}()$ and $\mathfrak{b}_{\text{words}}()$ are not defined.

CVF-11

- **Severity** Major
- **Source** Step 3f

ABDK Comment The Step 3f: Unpad input bytes (out-of-circuit) subsection is not rigorous so that it is not clear if the keccak computation is handled properly. It also mixes the circuit spec with the implementation details like data structures. This should be refactored.

CVF-12

- **Severity** Moderate
- **Source** Page 12. Step 5

Specification Excerpt This step takes the 32 output bytes O of the previous step and composes them into 2 field elements.

- Let $\mathsf{O}_1 = \mathsf{O}[0..16]$ and $\mathsf{O}_2 = \mathsf{O}[16..32]$

ABDK Comment Usually the [] sign means `inclusive`. Consider rewriting to make the intervals non overlapping.

CVF-13

- **Severity** Major
- **Source** Step 6

ABDK Comment The Step 6: assign keccak cells subsection is not rigorous so that it is not clear if the computation is handled properly. It also mixes the circuit spec with the implementation details like data structures. This should be refactored.

7 Universal Batch Verifier Specification. Version 1

CVF-14

- **Severity** Moderate
- **Source** Overview

Specification Excerpt The Universal Batch Verifier (UBV) circuit is one of the three circuits which constitute the Universal Proof Aggregator (UPA), together with the Variable Length Keccak circuit and the Outer circuit.
The high level idea is: given a batch of Groth16 proofs $(vk_i, \pi_i, P_i)_{i=1}^{B}$, it:

- Verifies they are all valid.
- Returns the hashes of the vk_is, i.e., the circuit IDs, as public outputs.

ABDK Comment It should be written that vk is a circuit trusted setup, π is a proof, and P is a public input

CVF-15

- **Severity** Moderate
- **Source** Conventions and Terminology

ABDK Comment It should be specified that $\mathbb{G}_1$ and $\mathbb{G}_2$ are all points of the respective elliptic curve, excluding the infinity point.

CVF-16

- **Severity** Moderate
- **Source** Inputs

Specification Excerpt Let $(vk_i, \pi_i, P_i)_{i=1}^{B}$ be a batch of valid (we don't aggregate invalid proofs) Groth16 proofs. Let ℓ_i be the number of public inputs to the circuit for vk_i, so that $P_i = (P_{1i}, \ldots, P_{\ell_i i})$. Here

- vk_i denotes the verification key. Note we adhere to the $\gamma = 1$ convention. * : consider making a reference* It consists of:
  - $\alpha_i \in \mathbb{G}_1$
  - $\beta_i \in \mathbb{G}_2$
  - $\delta_i \in \mathbb{G}_2$
  - $s_i = (s_{0i}, s_{1i}, \ldots, s_{\ell_i i})$

ABDK Comment It should be said it belongs to $\mathbb{G}_1$.

CVF-17

- **Severity** Minor
- **Source** Components

Specification Excerpt

- Input: a natural number $N \in \mathbb{N}$ and a length $\ell \leq N$. We consider $\ell \in \mathbb{F}_r$.
- A vector of bytes (aka bitmask): $\mathfrak{b} = (b_0, \ldots, b_{N-1})$ such that:
  - $b_k = 1$ for $k = 0, 1, \ldots, \ell - 1$.
  - $b_k = 0$ for $k = \ell, \ell + 1, \ldots, N - 1$.
- Description:
  - Iterate over $i \in 0, 1, \ldots, N$.
  - For each i:
    * Assign $i \in \mathbb{F}_r$ as a witness.
    * $b_k =$ **is less than**(i, ℓ)

ABDK Comment Assuming that $l_i \leq N$, this can be checked with just a single constraint per entry.

CVF-18

- **Severity** Major
- **Source** Step 1. Check...

ABDK Comment It is never said that $l_i \leq N$ must be enforced.

CVF-19

- **Severity** Moderate
- **Source** Step 2: Compute ...

Specification Excerpt For $i = 1, \ldots, B$, the circuit computes

$$\mathsf{C}_{\mathsf{ID}i} = \mathsf{poseidon}(vk_i),$$

where poseidon denotes the hash function constructed by applying the duplex sponge construction (with $\mathrm{RATE} = 2$) to the Poseidon permutation (with 8 full rounds and 57 partial rounds).
It does so as follows:

- Absorb the keccak hash of the domain tag "Saturn v1.0.0. CircuitId".

ABDK Comment Please elaborate how the hash is reduced to fit the field size.z A more secure and less error prone way to hash a variable length input is to hash the length as well, otherwise the outputs of different inputs may be related.

CVF-20

- **Severity** Moderate
- **Source** Step 2: Compute ...

Specification Excerpt For $i = 1, \ldots, B$, the circuit computes

$$\mathsf{C}_{\mathsf{ID}i} = \mathsf{poseidon}(vk_i),$$

where poseidon denotes the hash function constructed by applying the duplex sponge construction (with $\mathrm{RATE} = 2$) to the Poseidon permutation (with 8 full rounds and 57 partial rounds).
It does so as follows:

- Absorb the keccak hash of the domain tag "Saturn v1.0.0. CircuitId".

ABDK Comment Please elaborate how the hash is reduced to fit the field size.

CVF-21

- **Severity** Major
- **Source** Step 2: Compute ...

Specification Excerpt - Absorb $\alpha_i||\beta_i||\delta_i||\overline{s}_{0i}$ in RATE-sized chunks. Let $O_0 \in \mathbb{F}_r^{\mathrm{RATE}+1}$ be the state after the absorption.

ABDK Comment As group elements with values from Fp are hashed, it should be documented how they are treated as Fr elements.

CVF-22

- **Severity** Major
- **Source** Step 2: Compute ...

Specification Excerpt

- For $j = 1, \ldots, L$, absorb $\overline{s}_{ji}$ into the state O_{j-1}, returning the state O_j.
- Compute an ℓ_i-th bit bitmask[2] $\mathfrak{b}$ from ℓ_i which has a 1 in the ℓ_i-th position (indexing from 0) and zeroes everywhere else.
- Let $o \in \mathbb{F}_r^{\mathrm{RATE}+1}$ be the matrix product $o = O \cdot \mathfrak{b}$.

ABDK Comment This is unclear. Please elaborate why not returning an element of O_j.

CVF-23

- **Severity** Major
- **Source** Step 2: Compute ...

Specification Excerpt

- Permute the state o (necessary because the number of field elements absorbed is a multiple of RATE) and return the result.

ABDK Comment This is unclear. Please explain how the full state is returned. A proper way to hash a vector is to make a SQUEEZE operation at the end.

CVF-24

- **Severity** Critical
- **Source** Step 3 Compute

ABDK Comment The challenge computation should involve all proof elements as well. Otherwise, if (A_i, B_i, C_i) can be selected after c is generated, the Prover can craft them to satisfy any malicious public inputs. Details provided in the next section.

8 Attack on Version 1 of the aggregation protocol

Denote the SRS part for the public inputs by $\overline{\psi} = (\psi_1, \psi_2, \dots,)$ with ψ_0 being the standalone term. It corresponds to $\overline{s}_0, \overline{s}_1, \dots$ in the original document.

1. Take a valid proof (A, B, C) with $l = 1$ for circuit $\mathcal{C}$. It then holds that

$$e(\psi_0 + P \circ \overline{\psi}, g_2)e(-A, B)e(\alpha, \beta)e(C, \delta) = 1, \tag{1}$$

   where $P \circ \overline{\psi} = \sum P_i \psi_i$.

2. Let $\widetilde{P}$ be invalid public input with $\widetilde{P} \circ \overline{\psi} = \psi + P \circ \overline{\psi}$
3. Start preparing a proof of aggregating two proofs for $\mathcal{C}$, where one proof has public input P and the second has public input $\widetilde{P}$.
4. Compute challenge c as a function of the trusted setup and $\widetilde{P}$.
5. We have to craft proofs (A_1, B_1, C_1) and (A_2, B_2, C_2) for which it holds that

$$e((c+1)\psi_0 + (P + c\widetilde{P}) \circ \overline{\psi}, g_2)e(-A_1, B_1)e(-cA_2, B_2)e((c+1)\alpha, \beta)e(C_1 + cC_2, \delta) = 1$$

6. We substitute

$$A_1 = A + \psi\frac{c^2}{c+1} \qquad B_1 = B + g_2 \qquad C_1 = C$$
$$A_2 = A - \psi\frac{c}{c+1} \qquad B_2 = B - (1/c)g_2 \qquad C_2 = C$$

7. We obtain

$$e((c+1)\psi_0 + (P + c\widetilde{P}) \circ \overline{\psi}, g_2)e(-A_1, B_1)e(-cA_2, B_2)e((c+1)\alpha, \beta)e(C_1 + cC_2, \delta) =$$
$$e((c+1)\psi_0 + c\psi + (c+1)P \circ \overline{\psi}, g_2)$$
$$e(-A - \psi\frac{c^2}{c+1}, B + g_2)e(-cA + \psi\frac{c^2}{c+1}, B - (1/c)g_2)e((c+1)\alpha, \beta)e((c+1)C, \delta) =$$

Removing terms from exponentiating (1) to $(c+1)$

$$e(c\psi, g_2)e(-\frac{c^2}{c+1}\psi, B)e(-A, g_2)e(-\frac{c^2}{c+1}\psi, g_2)e(\frac{c^2}{c+1}\psi, B)e(A, g_2)e(\frac{c^2}{c+1}\psi, -(1/c)g_2) =$$
$$e((c - \frac{c^2}{c+1} - \frac{c}{c+1})\psi, g_2) = 1$$

As a result, we get two forged proofs which pass the pairing check in a batch.

9 Universal Batch Verifier Specification. Version 2

CVF-25

- **Severity** Minor
- **Source** Commitment hash

Specification Excerpt Hash the Pedersen commitment point $\overline{\pi_i}.m$. The curve-to-field hash function used is:

ABDK Comment Consider naming it.

CVF-26

- **Severity** Minor
- **Source** Commitment hash

Specification Excerpt

- Decompose m into bytes: concatenate the 32-byte big-endian representations of the x and y coordinates of the affine representation of m to obtain a 64-byte representation of m.
- Compute the 32-byte Keccak digest of the byte representation of m.
- Interpret these bytes as the big-endian byte representation of a 256-bit integer.
- Reduce modulo r to obtain an $\mathbb{F}_r$ element comm_i. output.

ABDK Comment It would be much more efficient and still secure to just truncate a few leading bits of the Keccak.

CVF-27

- **Severity** Moderate
- **Source**

Specification Excerpt

- If $n_i = 1$, absorb $vk_i.h_1$ and $vk_i.h_2$
- Squeeze and return result.

ABDK Comment Consider always absorbing h_1, h_2, l_i, μ_i

10 Variable length keccak circuit specification. Version 2

CVF-28

- **Severity** Minor
- **Source** Curve-to-Field Hash

Specification Excerpt Each of the M application proofs above has also a $\mathbb{G}_1$ point m_i. For each m_i, the Keccak circuit

- Computes the 32-byte keccak digest of (the byte decomposition of) m_i
- Interprets the result as a 256-bit integer
- Returns the mod r reduction of that integer

ABDK Comment As the keccak gadget returns an array of bits, consider just truncating leading bits for fewer constraints.

CVF-29

- **Severity** Major
- **Source** Keccak phase 0

Specification Excerpt We do not change the output `rows` in any way. Therefore, assuming a thorough audit of the prior Keccak circuit, `keccak_phase0_with_flags` may be audited by confirming that the stored cell locations are indeed as described above.

ABDK Comment Currently for each proof a keccak function of maximum possible length L is called as if all inputs would be L words long, and after that some intermediate state is used for the digest using the actual length l_i. This is quite inefficient if most of inputs are shorter than the maximum L. A more efficient way to handle that is to bound the total number of Keccak permutations (24-round blocks) W, and to compute W permutations within the circuit. This is sufficient as long as $\sum_i l_i \leq W$. To pick up the right digest, the circuit runs the permutation counter and the proof counter $\sum_{j<i} l_j$ and uses the intermediate digest whenever the two collide. There will be some overhead to zeroize the state at the start of each proof but it is negligible.

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