Report

v. 1.0

Customer

Nebra

Circuits Audit

Saturn. Circuits Update

29th August 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	29.08.24	A. Zveryanskaya	Initial Draft
0.2	29.08.24	A. Zveryanskaya	Minor revision
1.0	29.08.24	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.

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:

- Original Code
- Code with Fixes

Files:

batch_verify/common/		
chip.rs	ecc.rs	types.rs
batch_verify/universal/		
chip.rs	mod.rs	types.rs
utils.rs		
keccak/		
chip.rs	inputs.rs	mod.rs
multivar.rs	utils.rs	variable.rs
outer/		
mod.rs	universal.rs	
utils/		
bitmask.rs	utils/commit-ment_point.rs	hashing.rs

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.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 3 major, and a few less important issues. All identified Major issues have been fixed.

Fixed 3 out of 3 issues

6 Major Issues

CVF-2. FIXED

- **Category** Procedural
- **Source** inputs.rs

Description This always panics.

Recommendation Resolve this TODO or just remove the function.

Client Comment *All legacy fixed circuit stuff was removed.*

```
61 +todo!()
```

CVF-3. FIXED

- **Category** Procedural
- **Source** mod.rs

Description This always panics.

Recommendation Resolve this TODO or remove the function.

Client Comment *All legacy fixed circuit stuff was removed.*

```
537 +todo!()
```

CVF-4. FIXED

- **Category** Unclear behavior
- **Source** utils.rs

Description This assumes that “num_bytes” is divisible by “NUM_PARTS” which isn’t guaranteed.

Recommendation Explicitly assert that “num_bytes” is divisible by “NUM_PARTS” or perform rounding up division.

Client Comment *All legacy fixed circuit stuff was removed.*

```
137 +let num_bytes_in_part = num_bytes / NUM_PARTS;
```

```
179 +let num_bytes_in_part = num_bytes / NUM_PARTS;
```

7 Moderate Issues

CVF-5. INFO

- **Category** Suboptimal
- **Source** commitment_point.rs

Description This check makes the “NUM_BYTES” constant redundant.

Recommendation Instead of the “NUM_BYTES” constant, use the same constant expression, that is used to calculate the “num_bytes” value.

Client Comment *Will fix in the future.*

```
32 +assert!(num_bytes == NUM_BYTES);
```

CVF-6. INFO

- **Category** Procedural
- **Source** commitment_point.rs

Description Here, a variable named “x_coordinate” is used to store both, “x” and “y” coordinates, which makes code harder to read.

Recommendation Use a separate variable for concatenation. or rename the “x_coordinate” variable.

Client Comment *Will fix in the future.*

```
90 +x_coordinate.append(&mut y_coordinate);
91 +x_coordinate
```

CVF-7. INFO

- **Category** Procedural
- **Source** commitment_point.rs

Description Here, a variable named “x_0_coordinate” is used to store several coordinates, which makes code harder to read.

Recommendation Use a separate variable for concatenation. or rename the “x_0_coordinate” variable.

Client Comment *Will fix in the future.*

```
108 +x_0_coordinate.append(&mut x_1_coordinate);
109 +x_0_coordinate.append(&mut y_0_coordinate);
110 +x_0_coordinate.append(&mut y_1_coordinate);
111 +x_0_coordinate
```

CVF-8. INFO

- **Category** Suboptimal
- **Source** hashing.rs

Description Absorbing the length separately is not needed if the domain tag is supported. It also complicates the further code.

Recommendation Consider injecting the length into the domain tag.

Client Comment *Won’t fix. We don’t call that function in the protocol anymore.*

```
155 +hasher.absorb(&len);
```

CVF-9. INFO

- **Category** Suboptimal
- **Source** utils.rs

Description An assertion regarding constants looks odd.

Client Comment *Will fix in the future.*

```
280 +assert!(
281 +    LIMB_BITS == 88 && NUM_LIMBS == 3,
282 +    "Unsupported␣limb␣decomposition"
283 +);
```

CVF-10. INFO

- **Category** Documentation
- **Source** chip.rs

Description The documentation prescribes the "either-or" condition whereas the code is simply "or".

Recommendation Documentation shall be made consistent.

Client Comment *Will fix in the future.*

```
169 +/// Enforces that either `vk.h1, vk.h2` are both assigned
170 +/// as G2 padding point or else `entry.has_commitment` is
171 +/// assigned as `true`.
```

```
197 +    let is_satisfied = self.gate().or_and(
```

CVF-11. INFO

- **Category** Documentation
- **Source** chip.rs

Description The documentation prescribes the "either-or" condition whereas the code is simply "or".

Recommendation Documentation shall be made consistent.

Client Comment *Will fix in the future.*

```
207 +/// G1 padding point or else `entry.has_commitment` is
208 +/// assigned as `true`.
209 +fn check_proof_commitment_padding(
```

```
229 +        ctx,
```

8 Minor Issues

CVF-1. INFO

- **Category** Documentation
- **Source** utils.rs

Description A value is smaller than the field size ‘p‘ if and only if its *higher* parts are equal to the higher parts of ‘p‘, and the first different part is smaller than that of ‘p‘ – not the lower parts.

Recommendation Check the inequalities starting with the most significant part.

Client Comment *We don’t believe this is an issue, as our algorithm is equivalent to the standard one you suggest. We ask you to decrease the severity to "minor" and the category to "documentation". We will add a comment explaining the algorithm better in the future*

```
189 +// It can satisfy all inequalities for the lower parts
190 +// AND be equal to the current part OR it can be strictly
191 +// smaller than the current part.
192 +let is_equal = chip.gate.is_equal(ctx, part, maximal_part);
193 +let is_equal_and_lower_parts =
194 +    chip.gate.and(ctx, lower_parts_satisfied, is_equal);
```

CVF-12. INFO

- **Category** Procedural
- **Source** commitment_point.rs

Description Other files define similar constants.

Recommendation Define common constants in one place.

Client Comment *Will fix in the future.*

```
21 +const NUM_BYTES_FQ: usize = 32;
```

CVF-13. INFO

- **Category** Documentation
- **Source** commitment_point.rs

Description The comment is confusing, as one could thing that the “num_limbs” argument could be used to control the number of limbs returned, however, actually, the argument value is only used to check that the number of libs is correct.

Recommendation Rephrase the comment.

Client Comment *Will fix in the future.*

```
52 +/// Decomposes `m` into `num_limbs` limbs.
```

```
66 +/// Decomposes `m` into `num_limbs` limbs.
```

CVF-14. INFO

- **Category** Procedural
- **Source** commitment_point.rs

Description This error message refers to the config, which the function actually doesn’t know about. The actual number of limbs is actually checked against an argument value.

Recommendation Rephrase the error message.

Client Comment *Will fix in the future.*

```
61 +"Number␣of␣limbs␣incompatible␣with␣the␣config"
```

```
75 +"Number␣of␣limbs␣incompatible␣with␣the␣config"
```

CVF-15. INFO

- **Category** Readability
- **Source** mod.rs

Recommendation This formula should use named constants instead of hardcoded numbers.

Client Comment *Will fix in the future.*

```
236 +NUM_LIMBS * (22 + 2 * len_s),
```

CVF-16. FIXED

- **Category** Procedural
- **Source** mod.rs

Recommendation This commented out code should be removed.

Client Comment *All legacy fixed circuit stuff was removed.*

```
384 +/*
185 385  let mut result = if self.is_fixed {
186 386      vec![self.app_vk_hash]
187 387  } else {
188 388      vec![self.len, self.app_vk_hash]
189 389  };
```

```
390 +*/
```

CVF-17. INFO

- **Category** Procedural
- **Source** mod.rs

Recommendation This commented out line should be removed.

Client Comment *Will fix in the future.*

```
945 +// fixed input = domain_tag || alpha || beta || gamma || delta ||
       ↪ vk_s length || vk_s[0] || vk_s[1]
```

CVF-18. FIXED

- **Category** Suboptimal
- **Source** mod.rs

Description Computing proof ID right before panic doesn't make sense.

Recommendation Just panic, don't compute anything.

Client Comment *All legacy fixed circuit stuff was removed.*

```
1119 +Self::compute_proof_id_fixed_length(
645 1120      ctx,
646 1121      &range,
647 1122      &mut keccak,
```

```
1123 +     &assigned_input,
650 1124  );
1125 +panic!("Keccak␣fixed␣length␣no␣longer␣supported");
```

CVF-19. INFO

- **Category** Readability
- **Source** multivar.rs

Recommendation This formula should use named constants instead of hardcoded numbers.

Client Comment *Will fix in the future.*

```
150 +    self.fixed_input.len() + var_preimage_len / 3 * 32,
```

```
154 +assert_eq!(var_bytes.len(), var_preimage_len / 3 * 32);
```

CVF-20. INFO

- **Category** Readability
- **Source** utils.rs

Description The same name “num_zeroes” is used for both, a vector and an element of that vector. This makes code harder to read.

Recommendation Use different names.

Client Comment *Will fix in the future.*

```
302 +.zip_eq(num_zeroes)
303 +.flat_map(|(limb, num_zeroes)| {
```

CVF-21. INFO

- **Category** Procedural
- **Source** utils.rs

Recommendation The description should mention the endianness of the input.

Client Comment *Will fix in the future.*

```
423 +/// Composes `bytes` into a field element
```

CVF-22. INFO

- **Category** Procedural
- **Source** utils.rs

Description Having the constants 3,3,5 purely in the numerical form is error prone as it is easy to forget changing them at the next protocol update.

Recommendation Consider naming these constants and defining them in some globally visible file.

Client Comment *Will fix in the future.*

```
587 ++ 3 // len + has_commitment + commitment_hash
```

```
589 +    * (NUM_FQ_PER_G1AFFINE * 3 // alpha + s[0] + commitment_point
590 +       + NUM_FQ_PER_G2AFFINE * 5 // beta + gamma + delta + h1 + h2
```

CVF-23. INFO

- **Category** Documentation
- **Source** utils.rs

Description It is unclear how this expression was derived

Recommendation Consider documenting

Client Comment *Will fix in the future.*

```
619 +.take(NUM_LIMBS * (24 + 2 * max_num_public_inputs))
```

CVF-24. INFO

- **Category** Procedural
- **Source** chip.rs

Description This comment looks like an unresolved TODO.

Recommendation Resolve or remove it.

Client Comment *Will fix in the future.*

```
739 +// TODO: why is this distinct from types.Groth16Pairs?
```

CVF-25. INFO

- **Category** Suboptimal
- **Source** types.rs

Description This is difficult to scale with the number of commitments

Recommendation Consider assuming throughout the code that the number of commitments is a constant but not necessarily 1.

Client Comment *Will fix in the future.*

```
36 +has_commitment: bool,
```

```
47 +    h1: vec![g2; has_commitment as usize],
48 +    h2: vec![g2; has_commitment as usize],
```

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