Report

v. 1.0

Customer

Nebra

Circuits Audit

Saturn. Circuits

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

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 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	kmod.rs
	utils.rs	variable.rs	
outer/			
	mod.rs	universal.rs	utils.rs
utils/			
	bitmask.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.
- **Recommendations** contain code style, best practices and other suggestions.

5 Our findings

We found 2 critical, 12 major, and a few less important issues. All identified Critical issues have been fixed.

Fixed 11 out of 14 issues

6 Critical Issues

CVF-1 FIXED

- **Category** Overflow/Underflow
- **Source** utils.rs

Description Overflow is possible when calculating inner product.

Recommendation There should be constraints to ensure the byte representation doesn't exceed the field size.

85
```
let result = chip.gate.inner_product(
    ctx,
    assigned_repr.clone().into_iter().map(QuantumCell::from),
    byte_decomposition_powers,
);
```

CVF-2 FIXED

- **Category** Flaw
- **Source** chip.rs

Recommendation The proofs must be hashed as well. Otherwise, it is possible to craft invalid proofs that are accepted for one particular challenge and public inputs.

240
```
for (vk_hash, entry) in vk_hashes.into_iter().zip_eq(entries.0.iter
    ↪ ()) {
    hasher.absorb(vk_hash);
    hasher.absorb(&entry.public_inputs.0.as_slice());
}
```

7 Major Issues

CVF-3 FIXED

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

Description This function returns 64 bytes, while only the first 32 bytes are actually used and the remaining bytes are zero.

Recommendation Consider returning 32 bytes.

```
46 pub fn domain_tag_bytes(domain_tag_str: &str) -> [u8; 64] {
```

CVF-4 FIXED

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

Description This usage of Poseidon does not handle variable length inputs properly, which may lead to collisions for inputs of different lengths.

Recommendation Consider using SAFE API for it, concretely hashing the input length into the domain tag.

Client Comment *We now absorb the domain tag followed by the public input length before hashing the vk data.*

```
80 let domain_tag: Option<AssignedValue<F>> = {
       if let Some(t) = domain_tag {
           assert!(!t.is_empty());
           Some(assigned_domain_tag(ctx, t))
       } else {
           None
       }
   };
```

CVF-5 FIXED

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

Description This won't work in case "max_parts" <= 0.

Recommendation Consider explicitly forbidding such a case.

225
```
let remainder_len = remainder_lens[0];
```

CVF-6 FIXED

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

Description This won't return "false" in case both, "is_fixed" and "is_var" are true.

Recommendation Consider returning "false" in such a case.

401
```
// Either all inputs are fixed or all inputs are var.
if !(self.is_fixed() || self.is_var()) {
    return false;
}
```

CVF-7 INFO

- **Category** Overflow/Underflow
- **Source** utils.rs

Description Overflow is possible here.

Recommendation Consider either preventing it or clearly explaining why it is fine here.

Client Comment *Overflow is impossible, explained why in the function comments.*

202
```
chip.gate.inner_product(
    ctx,
    chunk.into_iter().cloned().map(QuantumCell::from),
    byte_decomposition_powers.clone(),
)
```

CVF-8 INFO

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

Description This bit seems redundant as even without this bit it would be possible to distinguish padding bits from data bits.

Recommendation Consider removing this bit.

Client Comment *Didn't change. adding this bit is an integral part of the kessak padding algorithm. Note it's not merely used to distinguish padding bits from data bits, but also during tha packing call which computes the words.*

```
247 bits.push(ctx.load_constant(F::one()));
```

CVF-9 FIXED

- **Category** Suboptimal
- **Source** variable.rs

Recommendation This expensive range check can be replaced with one constraint per entry using the fact that there is index of the array equalling 'i'.

```
45 bitmask.push(chip.is_less_than(ctx, idx, i, num_bits as usize));
```

CVF-10 FIXED

- **Category** Suboptimal
- **Source** variable.rs

Recommendation This could be replaced with a single "mul_add" gate.

```
74 let len = range.gate.add(ctx, len, one);
   range.gate.mul(ctx, len, num_words)
```

CVF-11 FIXED

- **Category** Documentation
- **Source** variable.rs

Description The comment disagrees with the function name. The comment says that the function returns the number of chinks, while the function name says it returns the last chunk index.

```
126 /// Computes the length of an input measured in 17-word chunks from
        ↪ the length
    /// of an input in bytes.
```

```
132 pub fn byte_len_to_last_chunk_index<F: Field>(
```

CVF-12 FIXED

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

Description The “OsRng” random number generator is OS-specific and thus its properties cannot be guaranteed.

Recommendation Consider passing the random number generator to be used as an argument. Alternatively, consider passing a deterministic generator in order to stress that it is producing a dummy proof only.

```
83 let rng = OsRng;
```

CVF-13 FIXED

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

Description In case “total_len + 1” is less than “self.s.len()”, this function silently does noting. This could make error investigations harder.

Recommendation Consider asserting that this is not the case.

```
45 let padding = (self.s.len()..total_len + 1)
```

```
82 let padding = (self.0.len()..total_len).into_iter().map(|_| F::zero
       ↪ ());
```

CVF-14 INFO

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

Description This should be removed if gamma is assumed to be 0 and ignored in the implementation.

Client Comment *Didn't change. In the next version (with gnark support) we'll need 'gamma' to take arbitrary values, and not just the generator.*

```
253 pub gamma: G2InputPoint<F>,
```

8 Moderate Issues

CVF-15 FIXED

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

Description There is no length check for this array.

Recommendation Consider adding an assert to ensure it fits into a word.

```
254 bits: &[AssignedValue<F>],
```

CVF-16 INFO

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

Recommendation The constraint may actually be satisfied when “byte_len” is not a multiple of “NUM_BYTES_PER_WORD” but the result of the finite field division of “byte_len” by “NUM_BYTES_PER_WORD” is less than 2^MAX_INPUT_LEN_WORDS_LOG2”.

Client Comment *Didn’t change. I believe this is fine. The first constraint ensures that* $a = bc \backslash mod r$*, while the second makes sure that* $c < s$ *for some* s *such that* $bs < r$*, which ensures that* $a = bc$ *as integers.*

```
102 /// The constraints here will only be satisfied when `byte_len` is
    /// a multiple of [`NUM_BYTES_PER_WORD`]. The constraints may not be
```

CVF-17 INFO

- **Category** Suboptimal
- **Source** variable.rs

Description In case“byte_len” is a factor of “RATE”, the result of this calculation is incorrect.

Recommendation Consider calculating as: (byte_len + RATE - 1) / RATE

Client Comment *Didn’t change. Added an explanation in the docs of the function for the edge case (i.e. when it is a exact multiple).*

```
139 let (quotient, _) = range.div_mod(ctx, byte_len, RATE, num_bits as
       ↪ usize);
```

CVF-18 INFO

- **Category** Flaw
- **Source** chip.rs

Recommendation This should be Fp12::one().

Client Comment *Didn't change. `halo2\_ecc` calls it `F\_p` and `halo2\_proofs` calls it `F\_q`, so this confusion is unavoidable.*

```
243 let fp12_one = fp12_chip.load_constant(ctx, Fq12::one());
```

9 Minor Issues

CVF-19 INFO

- **Category** Bad naming
- **Source** bitmask.rs

Description The term “bitmask” seems to be used incorrectly here. A bitmask is an array of integers whose concatenated binary representations are treated as arrays of bits. I.e. each integer value is used to store several bit values. However, these functions store one bit per integer value, so here we have boolean arrays rather than bit masks.

Recommendation Consider renaming.

```
20 pub fn ith_bit_bitmask<F: ScalarField>(
```

```
43 pub fn first_i_bits_bitmask<F: ScalarField>(
```

CVF-20 FIXED

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

Recommendation This assertion is redundant, as it duplicates the assertion inside the “domain_tag_bytes” function.

```
82 assert!(!t.is_empty());
```

CVF-21 INFO

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

Description We didn’t review the “get_field_element” function.

Client Comment *Out of scope: it’s ok.*

```
231 .map(|b| range.gate.get_field_element(*b as u64)),
```

```
357     .map(|b| range.gate().get_field_element(*b as u64)),
```

CVF-22 INFO

- **Category** Bad naming
- **Source** inputs.rs

Description This makes naming inconsistent.

Recommendation Consider using more similar, but yet different names.

```
20 /// NOTE: This should not be renamed to circuit_id so that the
      ↪ Keccak
   /// prover daemon errors if it receives a request from the wrong
      ↪ type
   /// (fixed vs varlen) type of client.
```

CVF-23 FIXED

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

Recommendation Consider using the notation from the spec.

```
72 pub struct KeccakConfig {
       /// Log-2 degree of Keccak circuit's `GateThreadBuilder`.
       pub degree_bits: u32,
       /// Number of public inputs to application circuit.
       /// (Does not count poseidon hash of application vk.)
       pub num_app_public_inputs: u32,
       /// Number of application proofs checked by a `
          ↪ BatchVerifyCircuit`.
       pub inner_batch_size: u32,
80     /// Number of `BatchVerifyCircuit`s being aggregated.
       pub outer_batch_size: u32,
       /// Lookup bits
       pub lookup_bits: usize,
   }
```

CVF-24 FIXED

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

Recommendation This TODO should be resolved or removed.

194
```
// TODO: Keep this a purely native struct.  Context operations
   ↪ should
// happen elsewhere.
```

CVF-25 FIXED

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

Description It is unclear what “k” is.

Recommendation Consider elaborating more.

734
```
/// Calculates the optimal [`KeccakGateConfig`] for a given `k`.
```

CVF-26 FIXED

- **Category** Bad datatype
- **Source** mod.rs

Recommendation The value “50” should be a named constant.

753
```
params.rows_per_round = std::cmp::min(optimal_rows_per_round, 50);
```

CVF-27 FIXED

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

Recommendation Inverting the condition and wapping the branches would make the code more readable.

926
```
if !witness_gen_only {
```

968
```
} else {
```

CVF-28 INFO

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

Description We didn't review this crate.

Client Comment *Out of scope: it's ok.*

```
8 fixed::types::BatchVerifyConfig,
```

CVF-29 FIXED

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

Description Despite the comment, this function actually returns a vector, rather than an iterator.

```
43 /// Returns an iterator of field elements over the powers of 2^8.
   fn byte_decomposition_powers<F: EccPrimeField>() -> Vec<F> {
```

CVF-30 INFO

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

Recommendation This wouldn't be necessary if the function would return an iterator, as promised in the comment above.

```
45 let num_bytes: usize =
       ((F::NUM_BITS + BYTE_SIZE_IN_BITS - 1) / BYTE_SIZE_IN_BITS) as
          ↪ usize;
```

CVF-31 FIXED

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

Description The expression at the right looks quite generic as it uses the "BYTE_SIZE_IN_BITS" constant instead of a hardcoded value, while the variable name at the left explicitly says "eight".

Recommendation Consider either using 8 at both sides, or at neither side.

```
49 let two_to_eight = F::from(1 << BYTE_SIZE_IN_BITS);
```

CVF-32 FIXED

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

Recommendation Consider specifying range as [24..32].

```
172 let h_bytes_1 = u64::from_le_bytes(digest[24..].try_into().unwrap())
    ↪ ;
```

CVF-33 FIXED

- **Category** Bad datatype
- **Source** utils.rs

Recommendation 32 and 16 should be named constants.

```
190 bytes: &[AssignedValue<F>; 32],
```

```
199     .chunks(16)
```

CVF-34 INFO

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

Recommendation Consider dropping the bit operations entirely for the sake of code brevity.

```
214 /// Converts bytes into bits.
    fn into_bits<F>(
```

CVF-35 FIXED

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

Recommendation The value “8” should be replaced with the constant “BYTE_SIZE_IN_BITS” for consistency.

```
224 Vec::with_capacity(bytes.len() * 8);
```

CVF-36 FIXED

- **Category** Procedural
- **Source** variable.rs

Recommendation Shift would be more efficient than the “pow” function call.

```
119 word_len.value().get_lower_32() < 2u32.pow(MAX_INPUT_LEN_WORDS_LOG2)
      ↪ ,
```

CVF-37 INFO

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

Description We didn’t review these crates.

Client Comment *Out of scope: it’s ok.*

```
6         native::compute_vk_hash,
```

```
13 utils::advice_cell_count,
```

CVF-38 FIXED

- **Category** Documentation
- **Source** mod.rs

Description These checks try to match argument values with environment variables. The reason for such checks is unclear.

Recommendation Consider explaining.

```
225     if gate_config_env_json != gate_config_json {
            panic!(
```

```
239 assert_eq!(
240     lookup_bits, config.lookup_bits,
```

CVF-39 INFO

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

Description We didn't review these crates.

```
18 advice_cell_count,
```

```
21 reduced::FromReduced,
```

CVF-40 INFO

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

Recommendation This TODO should be resolved or removed.

```
146 // TODO: We may not need to check the vk padding. In that case, all
    ↪ the code
// above (after the bitmask computation) can be replaced by the
    ↪ following:
```

CVF-41 FIXED

- **Category** Suboptimal
- **Source** chip.rs

Recommendation This cope could be simplified by zipping “inputs” with “r_powers” and iterating through the zipped sequence.

413
```
for (idx, input) in inputs.iter().enumerate() {
```

418
```
            self.gate().mul(builder.main(0), *input, r_powers[idx])
```

CVF-42 INFO

- **Category** Documentation
- **Source** types.rs

Recommendation This comment should mention that the length of a created entry is the maximum number of public inputs.

141
```
/// Creates a dummy [`BatchEntry`] for `config`.
```

CVF-43 INFO

- **Category** Bad naming
- **Source** types.rs

Recommendation Consider using better names for the structure fields. Also the entries are not pairs of the same type but of different types (G1, G2), which is confusing.

312
```
/// Representation of the (A, B), (C, delta) and (alpha, beta) [`
   ↪ EcPointPair`]s
/// for a series of Groth16 proofs
pub(crate) type Groth16Pairs<F> = (
    Vec<EcPointPair<F>>,
    Vec<EcPointPair<F>>,
    Vec<EcPointPair<F>>,
);
```

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