# ABDK CONSULTING

SMART CONTRACT AUDIT

## Matter Labs

## ZkSync Phase 2 p.1

abdk.consulting

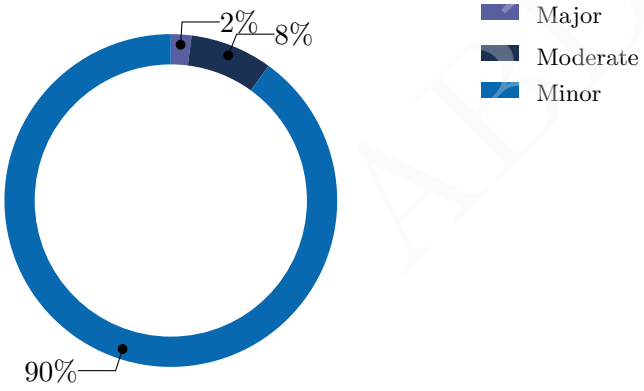
# SMART CONTRACT AUDIT CONCLUSION

Mikhail Vladimirov and Dmitry Khovratovich

22th December 2020

We've been asked to review the ZkSync v2 smart contracts and zero-knowledge circuit and witness generator given in separate files. We were given also a spec for the zero-knowledge protocol, which is an extension to Plonk. We found no issues in Rust files. The issues for Solidity contracts are presented in this document At some point we were also given the formal spec.





## Findings

| ID | Severity | Subject | Status |
|---|---|---|---|
| CVF-1 | Minor | Suboptimal condition | Opened |
| CVF-2 | Minor | The redundant check | Opened |
| CVF-3 | Minor | Revert absence | Opened |
| CVF-4 | Minor | Return absence | Opened |
| CVF-5 | Moderate | Suboptimal function | Opened |
| CVF-6 | Moderate | Suboptimal argument | Opened |
| CVF-7 | Minor | Suboptimal architecture | Opened |
| CVF-8 | Minor | Unspecified access level | Opened |
| CVF-9 | Minor | The struct field name | Opened |
| CVF-10 | Minor | Wrapped value | Opened |
| CVF-11 | Minor | Incorrect name function | Opened |
| CVF-12 | Minor | Suboptimal architecture-2 | Opened |
| CVF-13 | Minor | Not used constant | Opened |
| CVF-14 | Minor | Unspecified access level-2 | Opened |
| CVF-15 | Minor | Structure name | Opened |
| CVF-16 | Minor | Uncommon practice | Opened |
| CVF-17 | Minor | Undocumented function modify | Opened |
| CVF-18 | Minor | The redundant variable | Opened |
| CVF-19 | Minor | Adding address in loop | Opened |
| CVF-20 | Minor | Optimisation | Opened |
| CVF-21 | Minor | Value replacing | Opened |
| CVF-22 | Minor | Unclear purpose | Opened |
| CVF-23 | Minor | Incorrect Comment | Opened |
| CVF-24 | Minor | Inconsistent naming | Opened |
| CVF-25 | Minor | Unclear relate | Opened |
| CVF-26 | Minor | Redundant line | Opened |
| CVF-27 | Minor | Code simplification | Opened |
| CVF-28 | Minor | Code simplification-2 | Opened |
| CVF-29 | Minor | Merged code | Opened |
| CVF-30 | Minor | More efficient constant | Opened |

| ID | Severity | Subject | Status |
|---|---|---|---|
| CVF-31 | Minor | Redundant variable in several cases | Opened |
| CVF-32 | Minor | Calculating as a side effect | Opened |
| CVF-33 | Minor | Comment absence | Opened |
| CVF-34 | Minor | The typo | Opened |
| CVF-35 | Minor | Redundant lines | Opened |
| CVF-36 | Minor | Unseparated value | Opened |
| CVF-37 | Minor | Renaming | Opened |
| CVF-38 | Major | Incorrect index | Opened |
| CVF-39 | Minor | Indicator absence | Opened |
| CVF-40 | Minor | Redundant check-2 | Opened |
| CVF-41 | Minor | Function call out spec | Opened |
| CVF-42 | Minor | Identical function | Opened |
| CVF-43 | Minor | Cheap statement | Opened |
| CVF-44 | Minor | Redundant Value-2 | Opened |
| CVF-45 | Minor | Rewritten function | Opened |
| CVF-46 | Minor | Uninitialized variable | Opened |
| CVF-47 | Minor | A single array with two fields | Opened |
| CVF-48 | Minor | Require instead of asset | Opened |
| CVF-49 | Minor | Redundant assignment | Opened |
| CVF-50 | Moderate | Missed range checks | Opened |
| CVF-51 | Minor | Suboptimal calculating | Opened |
| CVF-52 | Minor | Redundant variable-2 | Opened |
| CVF-53 | Moderate | Overflow | Opened |
| CVF-54 | Minor | Incorrect assignment | Opened |
| CVF-55 | Minor | Constant instead value | Opened |
| CVF-56 | Minor | More efficient replacement | Opened |

# Contents

# 1 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the request by Matter Labs. This is part 2 of the audit of ZkSync Version 2, which includes the following smart contract and circuit files by release audit-09-11-2020

- KeysWithPlonkVerifier.sol
- PlonkCore.sol
- mod.rs

The audit goal was to verify the smart contracts for integrity, consistency, and correctness, and to check that the code at mod.rs is a valid part of zero-knowledge proof protocol.

## 1.1 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.

## 1.2 About Customer

Matter Labs is a private enterprise that specializes in Layer 2 solutions for Ethereum.

## 1.3 Disclaimer

The audit follows the best practices, which are relevant at the date of publication.

# 2 Detailed Results

## 2.1 CVF-1 Suboptimal condition

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** KeysWithPlonkVerifier.sol

**Recommendation** The condition `_size == uint32(630)` instead of the `(_size == uint32(630))` would be shorter and more readable.

Listing 1: Suboptimal condition

```
13      if (_size == uint32(630)) { return true; }
14      else { return false; }
```

## 2.2 CVF-2 The redundant check

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal Code
- **Source** KeysWithPlonkVerifier.sol

**Description** The `blockSizeToVkIndex` function always returns zero.
**Recommendation** Consider adding check for revert in case `chunks` is not 630.

Listing 2: The redundant check

```
17      function blockSizeToVkIndex(uint32 _chunks) internal pure
    returns (uint8) {
```

## 2.3 CVF-3 Revert absence

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal Code
- **Source** KeysWithPlonkVerifier.sol

**Description** The `getVkAggregate` function should probably revert in case `blocks` is not a supported value.
**Recommendation** Consider adding the revert operation.

Listing 3: Revert absence

```
22      function getVkAggregated(uint32 _blocks) internal pure
   ↪ returns (VerificationKey memory vk) {
```

### 2.4 CVF-4 Return absence

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** KeysWithPlonkVerifier.sol

**Description** Return operator is missing in the default branch

Listing 4: Return absence

```
18        if (_chunks == uint32(630)) { return 0; }
27        else if (_blocks == uint32(40)) { return
    ↪ getVkAggregated40(); }
```

### 2.5 CVF-5 Suboptimal function

- **Severity** Moderate
- **Status** Opened
- **Category** Suboptimal
- **Source** KeysWithPlonkVerifier.sol

**Description** The next function:

- the `getVkAggregated1`
- the `getVkAggregated5`

could be optimised. See example.

### 2.6 CVF-6 Suboptimal argument

- **Severity** Moderate
- **Status** Opened
- **Category** Suboptimal
- **Source** KeysWithPlonkVerifier.sol

**Description** The `new_fr` argument allocates a new structure in memory just to copy its contents into another place in memory. This applies to other similar allocations below.
**Recommendation** See gist above in the previous comment for suggestion how to address this.

Listing 5: Suboptimal argument

```
34        vk.omega = PairingsBn254.new_fr(0
    ↪ x18c95f1ae6514e11a1b30fd7923947c5ffcec5347f16e91b4dd654168326bede
    ↪ );
```

### 2.7 CVF-7 Suboptimal architecture

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Recommendation** According best code practices the `PairingsBn254` library should be in the separate file `PairingBn254.sol` to simplify code navigation.
It also true for the `VerifierWithDeserialize` contract.

Listing 6: Suboptimal architecture

```
4       library PairingsBn254 {1030    contract
    ↪ VerifierWithDeserialize is Plonk4VerifierWithAccessToDNext
    ↪ {
```

### 2.8 CVF-8 Unspecified access level

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** The next constants: `q_mod`, `r_mod`, `bn254_b_coeff` have no specified access level. The internal access will be used by default.
**Recommendation** Consider specifying access level explicitly. Also, rename constants in upper case.

Listing 7: Unspecified access level

```
5   uint256 constantq_mod=
21888242871839275222246405745257275088696311157297823662689037894645226208583;
    ↪
6   uint256 constant r_mod=
21888242871839275222246405745257275088548364400416034343698204186575808495617;
    ↪
7   uint256 constant bn254_b_coeff = 3;
```

### 2.9 CVF- 9 The struct field name

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** PlonkCore.sol

**Recommendation** The struct field names should start with lower case letter.

Listing 8: The struct field name

```
10  uint256 X;
11  uint256 Y;
```

### 2.10 CVF-10 Wrapped value

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** Wrapping a single value into a struct is inefficient.
**Recommendation** Consider using unwrapped value. In that case structs are always allocated in memory, while unwrapped values could be allocated on stack.

Listing 9: Wrapped value

```
15        uint256 value;
```

### 2.11 CVF-11 Incorrect name function

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** PlonkCore.sol

**Recommendation** Probably the name of the `new_fr` function should contains `checked` similarly to the `g1_checked`.

Listing 10: Incorrect name function

```
18        function new_fr(uint256 fr) internal pure returns (Fr
    ↪ memory) {
```

### 2.12 CVF-12 Suboptimal architecture-2

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Recommendation** The contract should be in its own file `Plonk4VerifierWithAccessToDNext.sol` to simplify code navigation.

Listing 11: Suboptimal architecture-2

```
308       contract Plonk4VerifierWithAccessToDNext {
```

### 2.13 CVF-13 Not used constant

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** The `ZERO` constant is not used.

Listing 12: Not used constant

```
316     uint256 constant ZERO = 0;
```

### 2.14 CVF-14 Unspecified access level-2

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** No access level specified for the constants in listing below, so internal access is used by default.
**Recommendation** Consider specifying access level explicitly.

Listing 13: Unspecified access level-2

```
316     uint256 constant ZERO = 0;
317     uint256 constant ONE = 1;
318     uint256 constant TWO = 2;
319     uint256 constant THREE = 3;
320     uint256 constant FOUR = 4;
321     uint256 constant STATE_WIDTH = 4;
322     uint256 constant NUM_DIFFERENT_GATES = 2;
323     uint256 constant NUM_SETUP_POLYS_FOR_MAIN_GATE = 7;
324     uint256 constant NUM_SETUP_POLYS_RANGE_CHECK_GATE = 0;
325     uint256 constant ACCESSIBLE_STATE_POLYS_ON_NEXT_STEP = 1;
326     uint256 constant NUM_GATE_SELECTORS_OPENED_EXPLICITLY =
    ↪ 1;
327     uint256 constant RECURSIVE_CIRCUIT_INPUT_COMMITMENT_MASK
    ↪ =0
    ↪ x00ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
    ↪ ;
328     uint256 constant LIMB_WIDTH = 68;
```

### 2.15 CVF-15 Structure name

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** The fields of this structure are named differently from the fields of `VerificationKey` structure defined in Rust code. This make is harder to compare Solidity and Rust code.
**Recommendation** Consider using consistent naming.

Listing 14: Structure name

```
332     struct VerificationKey {
```

### 2.16 CVF-16 Uncommon practice

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** PlonkCore.sol

**Description** in
the `NUM_SETUP_POLYS_FOR_MAIN_GATE + NUM_SETUP_POLYS_RANGE_CHECK_GATE` file indexes from 0 to `STATE_WIDTH+2` are used for this array. In `KeysWithPlonkVerifier.sol` indexes 0 to 6 are used. The different constants are used to calculate array size in struct declaration and to calculate indexes when array is accessed. It looks uncommon.

Listing 15: Uncommon practice

```
336     PairingsBn254.G1Point[NUM_SETUP_POLYS_FOR_MAIN_GATE +
    ↪ NUM_SETUP_POLYS_RANGE_CHECK_GATE] gate_setup_commitments;
```

### 2.17 CVF-17 Undocumented state changes

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** PlonkCore.sol

**Description** In line 452 and further the function modifies the state and some other inputs.
**Recommendation** Documentation comment needed.

Listing 16: Undocumented state changes

```
452     function verify_at_z(
```

### 2.18 CVF-18 Redundant variable

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** The `rhs` variable looks redundant. Values added to it could be just subtracted from `lhs`, and values subtracted from it could be added to the `lhs`.
**Recommendation** Consider refactoring. The final check would look like `lhs.value == 0` and renaming the `lhs` into the `sum` or something similar.

Listing 17: The redundant variable

```
462     PairingsBn254.Fr memory rhs =PairingsBn254.copy(proof.
    ↪ linearization_polynomial_at_z);
```

### 2.19 CVF-19 Unrolled loops of length 1

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Recommendation** The array should be indexed in a loop till `NUM_GATE_SELECTORS_OPENED_EXPLICITLY` as for other arrays of length 1.

Listing 18: Adding address in loop

```
473     inputs_term.mul_assign(proof.gate_selector_values_at_z
    ↪ [0]);
651     tmp_g1 = vk.gate_setup_commitments[STATE_WIDTH+2].
    ↪ point_mul(proof.wire_values_at_z_omega[0]);
655     res.point_mul_assign(proof.gate_selector_values_at_z[0]);
    ↪  // these is only one explicitly opened selector
905     transcript.update_with_fr(proof.gate_selector_values_at_z
    ↪ [0]);
```

### 2.20 CVF-20 Optimisation

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Recommendation** The operation described in the comment can be computed with 3 mulmods if further optimization is needed.

Listing 19: Optimisation

```
476     // now we need 5th power
```

### 2.21 CVF-21 Value replacing

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** the value `proof.permutation_polynomials_at_z.length` always equals `STATE_WIDTH-1`.
**Recommendation** Consider replacing that value.

Listing 20: Value replacing

```
484     for (uint256 i = 0; i < proof.
    ↪ permutation_polynomials_at_z.length;i++) {
```

### 2.22 CVF-22 Redundancy

- **Severity** Minor
- **Status** Opened
- **Category** Unclear
- **Source** PlonkCore.sol

**Description** the `current_alpha` parameter equals to one at the only place where this function is called. Is it really necessary to pass it?

Listing 21: Unclear purpose

```
515     PairingsBn254.Fr memory current_alph
```

### 2.23 CVF-23 Incorrect Comment

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** PlonkCore.sol

**Description** the comment is incorrect.

- the `alpha` is true only when `current_alpha` is one.
- `c - 4d` according to the spec seems to be `4d-c`
- `4b - c` according to the spec seems to be `b-4c`
- `alpha^2` is true only when `current_alpha` is one.

Listing 22: Incorrect Comment

```
518     // we multiply selector commitment by all the factors (
    ↪ alpha*(c - 4d)(c - 4d - 1)(..-2)(..-3) + alpha^2 * (4b - c)
    ↪ ()()() + {} + {})
```

### 2.24 CVF-24 Inconsistent naming

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** PlonkCore.sol

**Description** variable names differ from both, the comment in the beginning of the function and the code below.
**Recommendation** Consider using consistent naming across all the code and documentation.

Listing 23: Inconsistent naming

```
533     // high - 4*low
```

### 2.25 CVF-25 Unclear relationship

- **Severity** Minor
- **Status** Opened
- **Category** Unclear
- **Source** PlonkCore.sol

**Description** It is unclear to say if 3 is related to the `STATE_WIDTH` or not.

Listing 24: Unclear relate

```
530     for (uint256 i = 0; i < 3; i++) {
```

### 2.26 CVF-26 Redundant line

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** `t0 = PairingsBn254.copy(t1);` is redundant as t1 could be used instead of t0 in the lines below.

Listing 25: Redundant line

```
549     t0 = PairingsBn254.copy(t1);
```

### 2.27 CVF-27 Code suboptimality

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** the `t0` is already equals to `t1`. The lines below could be replaced with `t0.sub_assign(one_fr);`

Listing 26: Code simplification

```
554     t0 = PairingsBn254.copy(t1);
555     t0.sub_assign(two_fr);
559     t0 = PairingsBn254.copy(t1);
560     t0.sub_assign(three_fr);
```

### 2.28 CVF-28 Code suboptimality

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Recommendation** the lines below could be simplified `res.add_assign(t2).mul_assign(state.alpha)`. It makes the `current_alpha` unnecessary.

Listing 27: Code simplification-2

```
554     t2.mul_assign(current_alpha);

565     res.add_assign(t2);
```

### 2.29 CVF-29 Code suboptimality

- **Severity** Minor
- **Status** Open
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** the code below is very similar to the body of the loop above.
**Recommendation** It could be merged with the loop removing duplication, by changing a single line that calculates initial value for
`t1: t1 = i <= 2 ? PairingsBn254.copy(proof.wire_values_at_z[2 - i]) :PairingsBn254.copy(proof.wire_values_at_z_omega[0]);`

Listing 28: Merged code

```
568     // now also d_next - 4a
```

### 2.30 CVF-30 More efficiently with constant

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Recommendation** The comment is correct, but it would be more efficient to have a named constant equal to `STATE_WITH+1 (or 2 for the line 651) named INDEX_Q_CONST` or equivalent example.

Listing 29: More efficient constant

```
633     res = PairingsBn254.copy_g1(vk.gate_setup_commitments[
    ↪ STATE_WIDTH + 1]); // index of q_const(x)
651     tmp_g1 = vk.gate_setup_commitments[STATE_WIDTH+2].
    ↪ point_mul(proof.wire_values_at_z_omega[0]); // index of
    ↪ q_d_next(x)
```

### 2.31 CVF-31 Redundant variables

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** The `tmp_g1`variable is redundant and the `PairingsBn254.P1()` value is never used.

- in the line `635, 640, 651, 661, 744, 753, 759` the result of `point_mul` could be passed directly to `point_add_assign`.
- in the line `713, 736` is actual comment above and then the result of this call could be passed directly to `point_add_assign`.
- in the line `779 tmp_q1 = v^(6+i)`.

Listing 30: Redundant variables

```
635     PairingsBn254.G1Point memory tmp_g1 = PairingsBn254.P1();
640     tmp_g1 = vk.gate_setup_commitments[i].point_mul
    (proof.wire_values_at_z[i]);
647     tmp_g1 = vk.gate_setup_commitments[STATE_WIDTH].point_mul
    ↪ (tmp_fr);
651     tmp_g1 = vk.gate_setup_commitments[STATE_WIDTH+2].
    ↪ point_mul(proof.wire_values_at_z_omega[0]); // index of
    ↪ q_d_next(x)res.point_add_assign(tmp_g1);
661     tmp_g1 = vk.gate_selector_commitments[1].point_mul(tmp_fr
    ↪ ); // selector commitment for range constraint gate *
    ↪ scalarres.point_add_assign(tmp_g1);
```

Listing 31: Redundant variables

```
713     tmp_g1 = proof.copy_permutation_grand_product_commitment.
    ↪ point_mul(grand_product_part_at_z);
736     PairingsBn254.G1Point memory tmp_g1 = PairingsBn254.P1()
744     tmp_g1 = proof.quotient_poly_commitments[i].point_mul(
    ↪ tmp_fr);
753     tmp_g1 =proof.wire_commitments[i].point_mul(
    ↪ aggregation_challenge);
759     tmp_g1 = vk.gate_selector_commitments[0].point_mul(
    ↪ aggregation_challenge);
765     tmp_g1 = vk.copy_permutation_commitments[i].point_mul(
    ↪ aggregation_challenge);
779     tmp_g1 = proof.wire_commitments[STATE_WIDTH - 1].
    ↪ point_mul(tmp_fr);
```

### 2.32 CVF-32 Calculating as a side effect

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** Calculating `alpha^4` as a side effect of `add_contribution_from_range_constraint_gates` is suboptimal. It requires three additional multiplications inside that function, while it could be calculated via two multiplications outside. See comments CVF-22-29

Listing 32: Calculating as a side effect

```
660     tmp_fr = add_contribution_from_range_constraint_gates
    (state, proof, current_alpha);
```

### 2.33 CVF-33 Comment missing

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** PlonkCore.sol

**Description** the formula in the listing looks very different from what the code below actually does. **Recommendation** Consider adding comments to the code describing where each value is calculated and used.

Listing 33: Comment absence

```
669     // z * non_res * beta + gamma + a
697     // - (a(z) + beta*perm_a + gamma)*()*()*z(z*omega) * beta
    ↪  * perm_d(X)
```

### 2.34 CVF-34 Typo

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** PlonkCore.sol

**Recommendation** There was a typo in the line: immedately – immediately.

Listing 34: The typo

```
717     // multiply them by v immedately as linearization has a
    ↪ factor of v^1
```

### 2.35 CVF-35 Redundant lines

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** PlonkCore.sol

**Recommendation** Perhaps, the line above should be omitted

Listing 35: Redundant lines

```
722     // now we need to add a part that is the rest
723     // for z(x*omega):
724     // - (a(z) + beta*perm_a + gamma)*()*()*(d(z) + gamma) *
    ↪ z(x*omega)
```

### 2.36 CVF-36 Unseparated value

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** the `memory` returns a memory reference to an array of two memory references to G1Point structures.
**Recommendation** Consider just returning two separate value of type `PairingsBn254.G1Point memory` giving them descriptive names. This would make code more efficient and more readable.

Listing 36: Unseparated value

```
731     ) internal view returns (PairingsBn254.G1Point[2] memory
    ↪ res) {
```

### 2.37 CVF-37 Renaming

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** PlonkCore.sol

**Recommendation** Perhaps renaming from the `tmp_fr` to the `z_to_i_times_n` would help reading.

Listing 37: Renaming

```
731         tmp_fr.mul_assign(z_in_domain_size);
```

### 2.38 CVF-38 Incorrect index

- **Severity** Major
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Recommendation** There should be `i`instead of 0.

Listing 38: Incorrect index

```
759         tmp_g1 = vk.gate_selector_commitments[0].point_mul(
    ↪ aggregation_challenge);
```

### 2.39 CVF-39 Indicator absence

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** some return paths do not initialize the full `state` object, which is not reflected in it but only in the companion boolean variable `valid`. Maybe `PartialVerifierState` should have some sort of indicator which keeps this information.

Listing 39: Indicator absence

```
919         return true;
```

## 2.40 CVF-40 Redundant check-2

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description**

- The `proof.wire_values_at_z.length` always equals to the `TATE_WIDTH`.
- The `proof.gate_selector_values_at_z.length` always equals to the `NUM_GATE_SELECTORS_OPENED_EXPLICITLY`.
- The `proof.permutation_polynomials_at_z.length` equals to the `STATE_WIDTH-1`.
- The `proof.wire_values_at_z_omega.length` equals to the `ACCESSIBLE_STATE_POLYS_ON_NEXT_STEP`.
- The `proof.permutation_polynomials_at_z.length` equals to the `STATE_WIDTH-1`.

Listing 40: Redundant check-2

```
793       proof.wire_values_at_z.length(proof.
    ↪ copy_permutation_grand_product_commitment.point_mul(tmp_fr)
    ↪ );
801       for (uint i = 0; i < proof.gate_selector_values_at_z.
    ↪ length; i++){
808       for (uint i = 0; i < proof.permutation_polynomials_at_z.
    ↪ length;i++) {
897       for (uint256 i = 0; i < proof.wire_values_at_z.length; i
    ↪ ++) {
901       for (uint256 i = 0; i < proof.wire_values_at_z_omega.
    ↪ length; i++){
907       for (uint256 i = 0; i < proof.
    ↪ permutation_polynomials_at_z.length
     ; i++) {
```

## 2.41 CVF-41 Spec missing

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** PlonkCore.sol

**Description** The `update_with_fr` is not part of the spec.

Listing 41: Spec missing

```
911       transcript.update_with_fr(proof.
    ↪ copy_grand_product_at_z_omega);
```

### 2.42 CVF-42 Identical function

- **Severity** Minor
- **Status** Opened
- **Category** Supoptimal
- **Source** PlonkCore.sol

**Description** The function seems identical to `plonk/Template.sol (former Verifier.sol, 3c458af)` except that selector polynomial check is 1 point only.

Listing 42: Identical function

```
850     function verify_initial(
```

### 2.43 CVF-43 Cheap statement

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** the `require(vk.num_inputs >= 1);` statement should go first as it is very cheap.

Listing 43: Cheap statement

```
856     require(vk.num_inputs >= 1)
```

### 2.44 CVF-44 Redundant Value-2

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** the `valid` is redundant.
**Recommendation** the next example would be shorter and more readable. In the lines 1140, 1159 the value returned by verify could be returned directly without assigning to a variable. `if (!verify_at_z(state, proof, vk))return false.`

Listing 44: Redundant Value-2

```
889     bool valid = verify_at_z(state, proof, vk);
1140    bool valid = verify(proof, vk);
1159    bool valid = verify_recursive(proof, vk,
    ↪ recursive_vks_root,max_valid_index, recursive_vks_indexes,
```

### 2.45 CVF-45 Suboptimal function

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Recommendation** the fragment in the listing 940-945 could be rewritten much shorter:
`if (valid) part = aggregate_commitments(state, proof, vk);`
Or even:
`if (valid = verify_initial(state, proof, vk))`
`part = aggregate_commitments(state, proof, vk);`
The fragment 951-957 could be rewritten too:

```
if (valid)
valid = PairingsBn254.pairingProd2(recursive_proof_part[0],
PairingsBn254.P2(), recursive_proof_part[1], vk.g2_x);
```

Listing 45: Suboptimal function

```
940     if (valid == false) {
941         return (valid, part);
942     }
943     part = aggregate_commitments(state, proof, vk);
944
945     (valid, part);
951     if (valid == false) {
952         return false;
953     }
954
955     valid = PairingsBn254.pairingProd2(recursive_proof_part
    ↪ [0], PairingsBn254.P2(), recursive_proof_part[1], vk.g2_x);
956
957     return valid;
```

### 2.46 CVF-46 Uninitialized variable

- **Severity** Moderate
- **Status** Opened
- **Category** Integrity
- **Source** PlonkCore.sol

**Description** The `part` variable has not been initialized.

Listing 46: Uninitialized variable

```
941     return (valid, part);
```

### 2.47 CVF-47 Two arrays

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Recommendation** Passing a single array of structs with two fields would be more efficient and would make length check unnecessary.

Listing 47: Two arrays

```
992     uint8[] memory recursive_vks_indexes
993     uint256[] memory individual_vks_inputs,
```

### 2.48 CVF-48 Require instead of assert

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Recommendation** There should be `require` according to common best practice. Also, as failed condition in this line means invalid proof. Probably the function should just return false instead of reverting the whole transaction. The code in the listing looks like it is always executed, while it is executed only when valid is true. Consider putting this code into `else` branch explicitly.

Listing 48: Require instead of asset

```
996     assert
    (recursive_vks_indexes.length == individual_vks_inputs.length
      ↪ );
974     assert(recursive_input == proof.input_values[0]);
975
976     (bool valid, PairingsBn254.G1Point[2] memory
   ↪ recursive_proof_part)aggregate_for_verification(proof, vk);
977     if (valid == false) {
978     if (valid == false) {
979         return false;
```

### 2.49 CVF-49 Redundant assignment

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** The assignment in this line is redundant, as the assigned value could be returned directly without assigning to any variable.

Listing 49: Redundant assignment

```
984         valid = PairingsBn254.pairingProd2(recursive_proof_part
    ↪ [0], PairingsBn254.P2(), recursive_proof_part[1], vk.g2_x);
```

### 2.50 CVF-50 Missed range checks

- **Severity** Moderate
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** The range checks are missing for the `subproofs_aggregated` input though some values cause overflow.

Listing 50: Missed range checks

```
994         uint256[] memory subproofs_aggregated
```

### 2.51 CVF-51 Suboptimal calculating

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Description** The line in the listing is suboptimal as new bytes array is allocated on every look iteration.
**Recommendation** Consider calculating concatenated all at once:

```
bytes memory concatenated =
abi.encodePacked(
recursive_vks_root,
recursive_vks_indexes,
recursive_vks_indexes,
subproofs_aggregated);
```

after checking values of all arrays for valitidy.

Listing 51: Subotimal calculating

```
1002            concatenated = abi.encodePacked(concatenated,
    ↪ index);
```

### 2.52 CVF-52 Redundant variable-2

- **Severity** Minor
- **Status** Opened
- **Category** Suboptomal
- **Source** PlonkCore.sol

**Description** The `bytes32 commitment` variable is redundant as it is assigned and read only once right after the assignment.

Listing 52: Redundant variable

```
1013          bytes32 commitment = sha256(concatenated)
```

### 2.53 CVF-53 Overflow

- **Severity** Moderate
- **Status** Opened
- **Category** Suboptomal
- **Source** PlonkCore.sol

**Description** The overflow is possible while calculating formula in the listing and similar formulas below in the code.

Listing 53: Overflow

```
1017          subproofs_aggregated[0] + (subproofs_aggregated[1] <<
    ↪ LIMB_WIDTH) + (subproofs_aggregated[2] << 2*LIMB_WIDTH) +(
    ↪ subproofs_aggregated[3] << 3*LIMB_WIDTH),
```

### 2.54 CVF-54 Incomplete assignment

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PlonkCore.sol

**Recommendation** In the line instead of 34 for consistency there should be:
`2*STATE_WIDTH + 2 +2*STATE_WIDTH+STATE_WIDTH +`
`ACCESSIBLE_STATE_POLYS_ON_NEXT_STEP+`
`NUM_GATE_SELECTORS_OPENED_EXPLICITLY +STATE_WDITH-1 +1+1+1 +2+2`

Listing 54: Incorrect assignment

```
1030      uint256 constant SERIALIZED_PROOF_LENGTH = 34;
```

### 2.55 CVF-55 Constant instead of value

- **Severity** Minor
- **Status** Opened
- **Category** Suboptomal
- **Source** PlonkCore.sol

**Description** the `j` value as well as similar indexes below are always the same, constants would be more efficient.

Listing 55: Constant instead value

```
1046            serialized_proof[j],
```

### 2.56 CVF-56 More efficient replacement

- **Severity** Minor
- **Status** Opened
- **Category** Suboptomal
- **Source** PlonkCore.sol

**Recommendation**

- In the line `1076` should be `ACCESSIBLE_STATE_POLYS_ON_NEXT_STEP` instead of the `proof.wire_values_at_z_omega.length`
- In the line `1084` should be `NUM_GATE_SELECTORS_OPENED_EXPLICITLY` instead of the `proof.gate_selector_values_at_z.length`
- In the line `1092` should be `STATE_WIDTH-1` instead of the `proof.permutation_polynomials_at_z.length`

Listing 56: More efficient replacement

```
1076            for (uint256 i = 0; i < proof.wire_values_at_z_omega.
    ↪ length;i++) {
1084            for (uint256 i = 0; i < proof.
    ↪ gate_selector_values_at_z.length;i++) {
1092            for (uint256 i = 0; i <proof.
    ↪ permutation_polynomials_at_z.length; i++) {
```

## References

[1] *Solidity Documentation*
https://docs.soliditylang.org/en/v0.6.0/060-breaking-changes.html