ABDK CONSULTING

CIRCUIT
AUDIT

ZkSwap

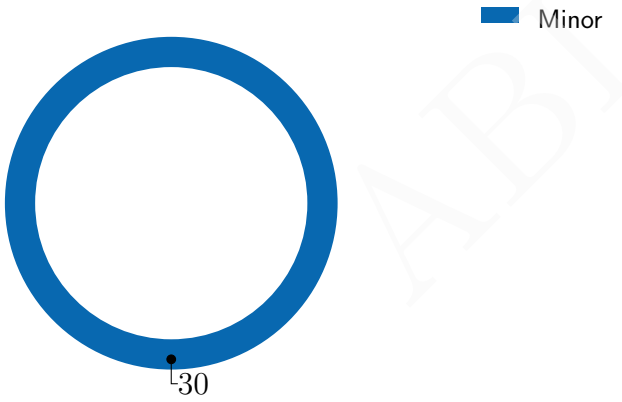
Rust. v0.9.8

abdk.consulting

CIRCUIT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
5th April 2021

We audited circuit files that are related to the witness construction in the circuit folder. We have found only a few minor issues.



ABDK

Findings

ID	Severity	Category	Status
CVF-1	Minor	Procedural	Opened
CVF-2	Minor	Procedural	Opened
CVF-3	Minor	Suboptimal	Opened
CVF-4	Minor	Documentation	Opened
CVF-5	Minor	Unclear behavior	Opened
CVF-6	Minor	Suboptimal	Opened
CVF-7	Minor	Suboptimal	Opened
CVF-8	Minor	Suboptimal	Opened
CVF-9	Minor	Documentation	Opened
CVF-10	Minor	Documentation	Opened
CVF-11	Minor	Overflow/Underflow	Opened
CVF-12	Minor	Unclear behavior	Opened
CVF-13	Minor	Suboptimal	Opened
CVF-14	Minor	Procedural	Opened
CVF-15	Minor	Procedural	Opened
CVF-16	Minor	Suboptimal	Opened
CVF-17	Minor	Procedural	Opened
CVF-18	Minor	Suboptimal	Opened
CVF-19	Minor	Suboptimal	Opened
CVF-20	Minor	Suboptimal	Opened
CVF-21	Minor	Procedural	Opened
CVF-22	Minor	Procedural	Opened
CVF-23	Minor	Procedural	Opened
CVF-24	Minor	Bad naming	Opened
CVF-25	Minor	Bad naming	Opened
CVF-26	Minor	Bad naming	Opened
CVF-27	Minor	Procedural	Opened

ID	Severity	Category	Status
CVF-28	Minor	Suboptimal	Opened
CVF-29	Minor	Documentation	Opened
CVF-30	Minor	Suboptimal	Opened

ABDK

Contents

ABDK

1 Document properties

Version

Version	Date	Author	Description
0.1	Apr. 3, 2021	D. Khovratovich	Initial Draft
0.2	Apr. 3, 2021	D. Khovratovich	Minor revision
1.0	Apr. 4, 2021	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

ABDK

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting. We were given access to the ZkSwap repo, release v0.9.8. This part of the report is devoted to certain files used to construct a circuit for proving ZkSwap transactions in zero knowledge. These files are written in Rust language and are located here. This part provides findings in the following files:

- account.rs;
- allocated_structures.rs;
- lib.rs;
- operation.rs;
- utils.rs;
- circuit.rs
- lp_exit_circuit.rs;
- witness/add_liquidity.rs;
- witness/remove_liquidity.rs;
- witness/swap.rs;
- witness/create_pair.rs;
- witness/_change_pubkey_offchain.rs;
- witness/close_account.rs;
- witness/deposit.rs;
- witness/full_exit.rs;
- witness/mod.rs;
- witness/noop.rs;
- witness/transfer_to_new.rs;
- witness/transfer.rs;
- witness/utils.rs;
- witness/withdraw.rs;

The audit goal was:

- to review the constraints enforced by the circuit and to check whether they are both necessary and sufficient for the protocol to operate according to the business logic;

- to check the code for suboptimal data structures and functions;
- to check whether the documentation and code comments match the code;
- to find scalability issues that could complicate the further development of the project.
- to detect bad code practices;
- to issue recommendations when necessary.

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.

ABDK

3 Detailed Results

3.1 CVF-1

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** swap.rs

Recommendation This comment should also be removed.

Listing 1:

```
83 / TODO: need to fix this bug that data.fee is encoded in fee—
   ↪ float point repr
```

3.2 CVF-2

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** swap.rs

Description Is this comment still relevant?

Listing 2:

```
87 / TODO: add assertion that fee_a, fee_b are both zero
```

3.3 CVF-3

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** swap.rs

Recommendation These two blocks could be merged into a single one returning two values.

Listing 3:

```
128 et val_fee = {

136 et amount_a_pair_fe = {
```

ABDK

3.4 CVF-4

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** swap.rs

Description It is unclear why a fee is stored in the "full_amount" field.
Recommendation Consider adding a comment similar to the comment in circuit.rs.

Listing 4:

```
135 rgs.full_amount = Some(BigUint_to_Fr::<Bn256>(&val_fee));
```

3.5 CVF-5

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** remove_liquidity.rs

Description Why were these checks removed? They look relevant.

Listing 5:

```
110 ssert!(remove_liq.a0 >= remove_liq.fee_in_t0);
    ssert!(remove_liq.a1 >= remove_liq.fee_in_t1);
```

3.6 CVF-6

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** add_liquidity.rs

Description Currently it is very easy to confuse which function should be called.
Recommendation Consider using different types for the inputs or results.

Listing 6:

```
280 ub(crate) fn u128_to_fe(x: u128) -> Fr {

287 n u128_to_fee_fe(x: u128) -> Fr {
```

3.7 CVF-7

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** utils.rs

Recommendation This function could be implemented in a more efficient way as sum x_i != 0.

Listing 7:

```
201 n multi_or<E: Engine, CS: ConstraintSystem<E>>(
```

3.8 CVF-8

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** utils.rs

Recommendation This function could be implemented in a more efficient way as sum x_i == n.

Listing 8:

```
219 ub fn multi_and<E: Engine, CS: ConstraintSystem<E>>(
```

3.9 CVF-9

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** utils.rs

Recommendation Typo: should be ‘calculate‘.

Listing 9:

```
667 et {b_i}, i=1..CAPACITY be the binary representation of a number
    ↪ . First you calcualte:
```

ABDK

3.10 CVF-10

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** utils.rs

Recommendation It should be clarified that in order for all valid witnesses to pass the test it should hold that CAPACITY >= 2*(max length of x,y).

Listing 10:

```
667 et {b_i}, i=1..CAPACITY be the binary representation of a number
    ↪ . First you calcualte:
```

3.11 CVF-11

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** utils.rs

Description Actually, overflow is still not guaranteed, but the absence of "ones" guarantees not overflow. Even in this example, q = 000101 = 5 and y = 001100 = 12, q * y = 60, i.e. q * y actually does fit into 6 bits!

Listing 11:

```
695 here is a "one" bit, thus overflow!
```

3.12 CVF-12

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** utils.rs

Description Why to disallow bit_length == CAPACITY?

Listing 12:

```
703 ssert!((bit_length as u32) < (E::Fr::CAPACITY));
```

3.13 CVF-13

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** utils.rs

Recommendation There should be a "Boolean::or" function for this.

Listing 13:

```
729 et mut result = Boolean::and(
730     cs.namespace(|| format!("safe_mul x_bits_rep {}", i)),
        &state.not(),
        &x_le_bit.not(),

    unwrap()
    not();

803 et mut result = Boolean::and(
        cs.namespace(|| format!("safe_mul y_bits_rep {}", i)),
        &state.not(),
        &y_le_bit.not(),

    unwrap()
    not();
```

ABDK

3.14 CVF-14

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** utils.rs

Recommendation This commented code should be removed or should be uncommented and surrounded by "if (debug)" statements.

```
741 for (i, x_bit) in x_bits.iter().enumerate() {
        println!("x bits: {} - {}", i, x_bit.get_value().grab()?);
    }

792 */

814 /*
    for (i, y_bit) in y_bits.iter().enumerate() {
        println!("y bits: {} - {}", i, y_bit.get_value().grab()?);
    }
    */

844 /*
    for (i, bit_and) in bits_and.iter().enumerate() {
    println!("and bits: {} - {}", i, bit_and.get_value().grab()?);
    }
    */
```

3.15 CVF-15

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** utils.rs

Recommendation This commented code block should be removed.

Listing 14:

```
953 / if s.get_value().is_some() {
    /     println!("s: {:?}", s.get_value().unwrap());
    / }
```

3.16 CVF-16

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** utils.rs

Description For a=2^(bit_length) and b=0 (valid by comment) this will overflow and thus valid witness won't pass.
Recommendation Consider requiring 'a < 2^(bit_length)'.

Listing 15:

```
1012 et diff_a_b_bits =
        diff_a_b.into_bits_le_fixed(cs.namespace(|| "diff_a_b bits"),
           ↪ bit_length + 1)?;
```

3.17 CVF-17

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** exit_circuit.rs

Recommendation Tests should be moved away from production code.

Listing 16:

```
179 [cfg(test)]
180 od test {
```

3.18 CVF-18

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** circuit.rs

Recommendation This assert is probably redundant given the definition of 'validator_address_bits'.

Listing 17:

```
383 et validator_address_padded = validator_address.pad(params::
        ↪ ACCOUNT_ID_BIT_WIDTH);

    let validator_address_bits = validator_address_padded.
        ↪ get_bits_le();
    assert_eq!(validator_address_bits.len(), params::
        ↪ ACCOUNT_ID_BIT_WIDTH);
```

3.19 CVF-19

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** circuit.rs

Recommendation This variable is redundant as its value is used only once in the next line.

Listing 18:

```
900 let b = (1 & (CHAIN_ID >> i)) == 1;
```

3.20 CVF-20

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** circuit.rs

Description This reversion is redundant.
Recommendation Just modify the bit extracting formula above to generate bits in reverse order: let b = (128 & (CHAIN_ID « i)) == 128;

Listing 19:

```
903 rev()
```

3.21 CVF-21

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** circuit.rs

Recommendation This commented code block should be removed or uncommented and guarded with "if (debug)" statement.

Listing 20:

```
1368 / if is_amm_tx.get_value().is_some() && is_amm_tx.get_value().
       ↪ unwrap() {
/       println!("update_f0: {:?}", update_f0.get_value().unwrap()
       ↪ );
1370 /       println!("update_f1: {:?}", update_f1.get_value().unwrap()
       ↪ );
/ }
```

3.22 CVF-22

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** circuit.rs

Recommendation This commented block should be removed.

Listing 21:

```
1398 / let val_fee_num_fe = E::Fr::from_str(&
     ↪ SWAP_VALIDATOR_FEE_NUMERATOR.to_string());
     / let val_fee_denom_fe =
1400 /     E::Fr::from_str(&SWAP_VALIDATOR_FEE_DENOMINATOR.to_string
     ↪ ());
     /
     / let val_fee_num = alloc_const(cs.namespace(|| "val_fee_num"),
     ↪ &val_fee_num_fe)?;
     / let val_fee_denom =
     /     alloc_const(cs.namespace(|| "val_fee_denom"), &
     ↪ val_fee_denom_fe)?;
     /
     / let fee = quote(
     /     cs.namespace(|| "validatorFee = div(amountIn*5,10000)"),
     /     &op_data.amount_unpacked.get_number(),
     /     &val_fee_denom,
1410 /     &val_fee_num,
     /     explicit_zero,
     /     params::BALANCE_BIT_WIDTH,
     / )?;
```

3.23 CVF-23

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** circuit.rs

Recommendation This commented block should be removed.

Listing 22:

```
1841 / flags.push(no_nonce_overflow(
     /     cs.namespace(|| "no nonce overflow"),
     /     &cur.account.nonce.get_number(),
     / )?);
```

3.24 CVF-24

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** circuit.rs

Description These three statements are very similar but the relationships between them is unclear. The namespace name is still inconsistent with the comments nor with the code logic.
Recommendation Consider removing comments and making the namespace name consistent with the code. You may use "xor" in the namespace name. Also note, that "x xor y" is equavalent to "x iff not y", i.e. x is true if and only if y is false. Thus, you may use the following namespace name: "keys are the same iff the account is not empty".

Listing 23:

```
1972 /keys are same and account is not empty
     /keys are not same and account is empty

1975     cs.namespace(|| "keys are same or account is empty"),
```

3.25 CVF-25

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** circuit.rs

Description The name is confusing, as the value is actually greater by one than the maximum possible ERC20 token ID.
Recommendation Consider renaming.

Listing 24:

```
2391 et max_erc20_token_id = Expression::constant::<CS>(

2662 et max_erc20_token_id = Expression::constant::<CS>(
```

3.26 CVF-26

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** circuit.rs

Description The name is confusing, as it doesn't give any clue about what kind of liquidity it is, other than this is not the initial liquidity.
Recommendation Consider renaming to "updated_liquidity" or "new_liquidity".

Listing 25:

```
3287 et non_initial_liquidity = min(
```

3.27 CVF-27

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** deposit.rs

Description The function "apply_leaf_operation" is not used anymore.
Recommendation Consider removing corresponding import.

Listing 26:

```
32 tils::{apply_leaf_operation, get_audits},
```

3.28 CVF-28

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** transfer.rs

Recommendation Consider using "debug!" macro of some other consistent way to do debug output.

Listing 27:

```
317 if env::var("PLONK_PROVER_DEBUG").is_ok() {
```

3.29 CVF-29

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** utils.rs

Description The meaning of the second returned value is unclear.
Recommendation Consider adding a documentation comment.

Listing 28:

```
416 > (AccountWitness<Bn256>, Fr) {
```

3.30 CVF-30

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** utils.rs

Description This function has much in common with the "get_account_witness" function.
Recommendation Consider merging them into one function, or implementing one function on top of another, or removing code duplication in some other way.

Listing 29:

```
434 n get_pair_account_witness(
```

ABDK