ABDK CONSULTING

SMART CONTRACT AUDIT

Zeropool

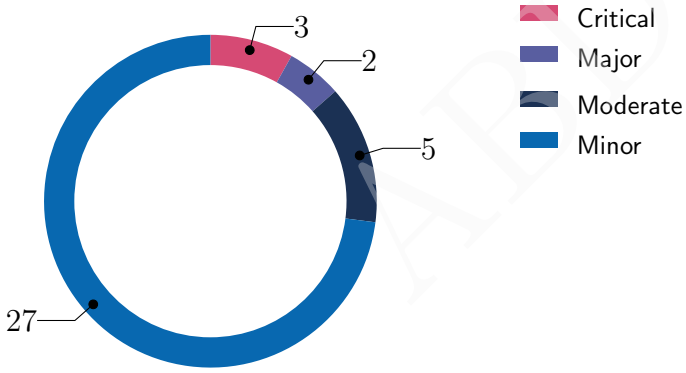
Rust

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Dmitry Khovratovich and Mary Maller
17th August 2021

We've been asked to review the AUDIT NAME smart contracts given in separate files. At some point we were also given the formal spec.



Findings

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

ID	Severity	Category	Status
CVF-28	Moderate	Bad datatype	Opened
CVF-29	Minor	Flaw	Opened
CVF-30	Critical	Flaw	Opened
CVF-31	Minor	Readability	Opened
CVF-32	Minor	Suboptimal	Opened
CVF-33	Minor	Procedural	Opened
CVF-34	Minor	Unclear behavior	Opened
CVF-35	Minor	Procedural	Opened
CVF-36	Minor	Overflow/Underflow	Fixed
CVF-37	Moderate	Suboptimal	Opened

Contents

1 Document properties **6**

2 Introduction **7**
2.1 About ABDK . . . 7
2.2 Disclaimer . . . 7
2.3 Methodology . . . 7

3 Detailed Results **9**
3.1 CVF-1 . . . 9
3.2 CVF-2 . . . 9
3.3 CVF-3 . . . 9
3.4 CVF-4 . . . 10
3.5 CVF-5 . . . 10
3.6 CVF-6 . . . 10
3.7 CVF-7 . . . 10
3.8 CVF-8 . . . 11
3.9 CVF-9 . . . 11
3.10 CVF-10 . . . 11
3.11 CVF-11 . . . 11
3.12 CVF-12 . . . 12
3.13 CVF-13 . . . 12
3.14 CVF-14 . . . 12
3.15 CVF-15 . . . 13
3.16 CVF-16 . . . 13
3.17 CVF-17 . . . 14
3.18 CVF-18 . . . 14
3.19 CVF-19 . . . 15
3.20 CVF-20 . . . 15
3.21 CVF-21 . . . 15
3.22 CVF-22 . . . 16
3.23 CVF-23 . . . 16
3.24 CVF-24 . . . 17
3.25 CVF-25 . . . 17
3.26 CVF-26 . . . 18
3.27 CVF-27 . . . 18
3.28 CVF-28 . . . 18
3.29 CVF-29 . . . 19
3.30 CVF-30 . . . 19
3.31 CVF-31 . . . 19
3.32 CVF-32 . . . 20
3.33 CVF-33 . . . 20
3.34 CVF-34 . . . 20
3.35 CVF-35 . . . 21
3.36 CVF-36 . . . 21
3.37 CVF-37 . . . 22

1 Document properties

Version

Version	Date	Author	Description
0.1	August 16, 2021	D. Khovratovich	Initial Draft
0.2	August 17, 2021	D. Khovratovich	Minor revision
1.0	August 17, 2021	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting 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.

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.

3 Detailed Results

3.1 CVF-1

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** cs.rs

Description Variable index for a constant in the CS should probably be a named constant

Listing 1:

```
262 vec![(Num::ONE, Index::Input(0))],
```

3.2 CVF-2

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

Description The function will fail if the signal length exceeds the modulus length, even if the signal value is a valid field element.
Recommendation Consider adding a comment about it.

Listing 2:

```
60 return true if signal > ct
fn c_comp_constant<C: CS>(signal: &[CBool<C>], ct: Num<C::Fr>)
   ↪ -> CBool<C> {
```

3.3 CVF-3

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** bitify.rs

Recommendation Some comment on the code functionality would make it more readable, for example that each addend is -1, 0 or 1 if the pair of signal bits is <,=, or> the pair of constant bits.

Listing 3:

```
87 acc = acc
       + k * match (ct_l, ct_u) {
           (false, false) => &sig_l + &sig_u - sig_lu,
90         (true, false) => &sig_l + &sig_u * Num::from(2) - sig_lu
              ↪ - Num::ONE,
           (false, true) => sig_lu + &sig_u - Num::ONE,
           (true, true) => sig_lu - Num::ONE,
       };
```

3.4 CVF-4

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** tx.rs

Recommendation The ‘deposit_value‘ would be more readable.

Listing 4:

```
102 et mut total_value = delta_value;
```

3.5 CVF-5

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

Recommendation Should be ‘total_energy‘ or ‘deposit_energy‘.

Listing 5:

```
103 et mut total_enegry = delta_energy;
```

3.6 CVF-6

- **Severity** Critical
- **Status** Opened
- **Category** Flaw
- **Source** tx.rs

Recommendation Should be ‘IN’ instead of ’OUT’.

Listing 6:

```
116 or i in 0..OUT {
        for j in i+1..OUT {
```

3.7 CVF-7

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

Recommendation It is probably redundant to compute the full hash of each note. Comparing indices should suffice (thus no note would be double spent).

Listing 7:

```
118 +=(&in_note_hash[i]-&in_note_hash[j]).is_zero().as_num();
```

3.8 CVF-8

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

Recommendation Name is confusing. 'Account secret' would be better.

Listing 8:

```
147 /build decryption key
```

3.9 CVF-9

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

Recommendation A comment that a dummy account is needed for deposits would be helpful.

Listing 9:

```
166 /assert root == cur_root || account.is_dummy()
```

3.10 CVF-10

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

Recommendation This line should probably be removed.

Listing 10:

```
179 /let account_index = c_from_bits_le(s.in_proof.0.path.as_slice()
    ↪ );
```

3.11 CVF-11

- **Severity** Moderate
- **Status** Opened
- **Category** Procedural
- **Source** tx.rs

Description This condition makes all unspent notes in the interval unspendable in the future. As a result, notes must be spent consecutively. This approach is error-prone.

Listing 11:

```
191 et note_index_ok = (!c_comp(input_index, note_index, HEIGHT)) &
    ↪ c_comp(output_index, note_index, HEIGHT);
```

3.12 CVF-12

- **Severity** Moderate
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** tx.rs

Description Overflow is theoretically possible here since balance is u64, index is u48, energy is u112, but it can be more than one note with maximal balance that are owned. In the latter case the energy will not fit u112.
Recommendation A solution would be to use u120 for energy.

Listing 12:

```
196 otal_enegry += note_value * (output_index - note_index);
```

3.13 CVF-13

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

Description Overflow is possible here leading to malleable signatures.
Recommendation Consider using an explicit range check.

Listing 13:

```
38 let s_bits = c_into_bits_le(&s, Num::<J::Fs>::MODULUS_BITS as
   ↪ usize);
```

3.14 CVF-14

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** ecc.rs

Description The cofactor size is curve dependent.
Recommendation Consider making it a parameter.

Listing 14:

```
34 self.double(params).double(params).double(params)
```

3.15 CVF-15

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

Recommendation The inverse of 8 can be precomputed.

Listing 15:

```
58 .map(|p| p.mul(Num::from(8).checked_inv().unwrap(), params));

71     .mul(Num::from(8).checked_inv().unwrap(), params)
```

3.16 CVF-16

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** ecc.rs

Recommendation The window size and its exponent should be named constants.

Listing 16:

```
96  for _ in 0..8 {

114         let zeros_len = (2 * bits_len) % 3;

119         let nwindows = all_bits_len / 3;

129             base = base.double().double().double();

139             let res = c_mux3(&all_bits[3 * i..3 * (i + 1)], &
                    ↪ table);
```

3.17 CVF-17

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

Description Whereas EC operations can be performance critical out of ZK, they may not be the bottleneck for ZK as other operations such as hashing can be much more expensive.
Recommendation Thus usage of Montgomery representation may not be as beneficial in terms of performance given amount of additional code it requires. Thus it would make sense to have Edwards operations only

Listing 17:

```
134 let mut acc = CMontgomeryPoint::from_const(cs, &mp);
    let mut base = c_base;

    for i in 0..nwindows {
        let table = gen_table::<C, J>(&base, params);
        let res = c_mux3(&all_bits[3 * i..3 * (i + 1)], &table);
140     let p = CMontgomeryPoint {
            x: res[0].clone(),
            y: res[1].clone(),
        };
        acc = acc.add(&p, params);
        base = base.double().double().double();
    }

    let res = acc.into_edwards();
```

3.18 CVF-18

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

Recommendation The point at infinity should be a named constant, and checks for it probably should be a single macro.

Listing 18:

```
169 let empty_acc = CMontgomeryPoint {
170     x: CNum::from_const(cs, &Num::ZERO),
        y: CNum::from_const(cs, &Num::ZERO),
    };
```

3.19 CVF-19

- **Severity** Moderate
- **Status** Opened
- **Category** Suboptimal
- **Source** account.rs

Recommendation It would be more optimal to pack i,b,e to a single field element and hash only 3 elements instead of 5.

Listing 19:

```
20 oseidon(&[self.eta, self.i.to_num(), self.b.to_num(), self.e.
   ↪ to_num(), self.t.to_num()], params.account())
```

3.20 CVF-20

- **Severity** Critical
- **Status** Opened
- **Category** Suboptimal
- **Source** cipher.rs

Description The nonce is the same for different encryptions (for example, when sending to the same recipient twice) with the same key. This breaks the authenticated encryption property of ChaCha20-Poly1305.
Recommendation Consider using a counter nonce or a big enough random nonce. If each key is used only once, a different method must still be used, for example, a nonce can be a hash of the message.

Listing 20:

```
33 let nonce = Nonce::from_slice(&hash[0..12]);
```

3.21 CVF-21

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

Recommendation Should be ‘account_ciphertext‘.

Listing 21:

```
65 let acccount_ciphertext = symcipher_encode(&account_key, &
   ↪ account.try_to_vec().unwrap());
```

3.22 CVF-22

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** cipher.rs

Description This data is not sufficient to spend a note. A recipient must also know a path in the Merkle tree, but the first part of it is hidden in the subtree that is not published (only its root is published).

Listing 22:

```
70 let notes_data = note.iter().map(|e|{
       let a:Num<P::Fs> = sb.gen();
       let p_d = EdwardsPoint::subgroup_decompress(e.p_d, params.
          ↪ jubjub()).unwrap();
       let ecdh =  p_d.mul(a, params.jubjub());
       let key = keccak256(&ecdh.x.try_to_vec().unwrap());
       let ciphertext = symcipher_encode(&key, &e.try_to_vec().
          ↪ unwrap());
       let a_pub = derive_key_p_d(e.d.to_num(), a, params);
       (a_pub.x, key, ciphertext)
```

3.23 CVF-23

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

Description It is unclear why to encrypt account data so that it can be decrypted on eta, which is known at the time of encryption.

Listing 23:

```
82 let a_p_pub = derive_key_a(sb.gen(), params);
   let ecdh = a_p_pub.mul(eta.to_other_reduced(), params.jubjub());
   let key = keccak256(&ecdh.x.try_to_vec().unwrap());
   let text:Vec<u8> = core::iter::once(&account_data.0[..]).chain(
      ↪ notes_data.iter().map(|e| &e.1[..])).collect::<Vec<_>>().
      ↪ concat();
   let ciphertext = symcipher_encode(&key, &text);
   (a_p_pub.x, ciphertext)
```

3.24 CVF-24

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** cipher.rs

Description Publishing hashes of plaintext alongside with 'ciphertext' breaks confidentiality as it is easy to check for a candidate note if it is encrypted in the given transaction. It is not needed for authenticity either since an AEAD encryption scheme such as ChachaPoly guarantees to return an error if the ciphertext is modified by an adversary. Thus these hashes should be removed from the output and the check should be replaced with the check on what 'decrypt' returns.

Listing 24:

```
93  account.hash(params).serialize(&mut res).unwrap();

95  for e in note.iter() {
        e.hash(params).serialize(&mut res).unwrap();
    }

155 if account.hash(params)!= account_hash {
        return None;
    }

164     if note.hash(params) != note_hash[i] {
            None
        } else {
```

3.25 CVF-25

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** ecc.rs

Recommendation Using (u,v) for Edwards coordinates in contrast to (x,y) for Montgomery would make the code more readable.

Listing 25:

```
20  pub x: Num<Fr>,
    pub y: Num<Fr>,
```

3.26 CVF-26

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

Description This implementation looks generic but in fact it is quite specific to twisted curves with a=-1 and cofactor=8.
Recommendation Consider making explicit comments about that and/or making it clear in the code that it does not work for all twisted curves.

Listing 26:

```
52 PrimeField> EdwardsPoint<Fr> {
```

3.27 CVF-27

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

Description These functions map points from one curve to another rather than just change the representation.
Recommendation Consider making it explicit in the name that the result is an edwards curve point.

Listing 27:

```
195 pub fn into_extended(&self) -> EdwardsPointEx<Fr> {
209 pub fn into_affine(&self) -> EdwardsPoint<Fr> {
222 pub fn into_extended(&self) -> EdwardsPointEx<Fr> {
```

3.28 CVF-28

- **Severity** Moderate
- **Status** Opened
- **Category** Bad datatype
- **Source** ecc.rs

Description The cofactor size is curve dependent.
Recommendation Consider passing it as parameter.

Listing 28:

```
241 pub fn mul_by_cofactor(&self) -> EdwardsPointEx<Fr> {
```

3.29 CVF-29

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** poseidon.rs

Description This function is a bijection for a few curves only. It works for BN254 and BLS12, but not for some other curves.
Recommendation Consider binding this implementation to a specific curve or pass the exponent as a parameter.

Listing 29:

```
20 sigma<C: CS>(a: &CNum<C>) -> CNum<C> {
```

3.30 CVF-30

- **Severity** Critical
- **Status** Opened
- **Category** Flaw
- **Source** poseidon.rs

Description Such matrix is likely to be singular and thus admit collisions. Poseidon matrices must be MDS and additionally possess certain properties to be cryptographically secure. Such matrices should be taken from the reference Poseidon implementation.

Listing 30:

```
28 let m = (0..t)
       .map(|_| (0..t).map(|_| seedbox.gen()).collect())
30     .collect();

42 let m = (0..t)
       .map(|_| (0..t).map(|_| seedbox.gen()).collect())
       .collect();
```

3.31 CVF-31

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** num.rs

Recommendation Using the multiplication macro would be more readable.

Listing 31:

```
43 CS::enforce(&signal, other, self);

59 CS::enforce(self, &inv_signal, &self.derive_const(&Num::ONE));
```

3.32 CVF-32

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

Description Squaring self would probably suffice

Listing 32:

```
75 (res_signal*self).assert_zero();
```

3.33 CVF-33

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

Recommendation It is not obvious how ‘alloc‘ and ‘assert_const‘ should work for signals as linear combinations, so some comment is needed.

Listing 33:

```
69 fn alloc(cs: &RCS<C>, value: Option<&Self::Value>) -> Self {

76 fn assert_const(&self, value: &Self::Value) {
```

3.34 CVF-34

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

Description It is not clear from the comments what montgomery polynomial is.
Recommendation Probably it should be written that for curve By2̂=x3̂+Ax2̂+x we define U = B2̂+BA+1.

Listing 34:

```
55 // value of montgomery polynomial for x=montgomery_b (has no
   ↪ square root in Fr)
```

3.35 CVF-35

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

Description These constants use different units: bits vs bytes.
Recommendation Consider adding suffixes to avoid confusion.

Listing 35:

```
15 const DIVERSIFIER_SIZE: usize = 80;
   const BALANCE_SIZE: usize = 64;
   const ENERGY_SIZE: usize = BALANCE_SIZE+HEIGHT;
   const SALT_SIZE: usize = 80;

20
   const POLY_1305_TAG_SIZE: usize = 16;
   const NUM_SIZE: usize = 32;
   const NOTE_SIZE: usize = (DIVERSIFIER_SIZE + BALANCE_SIZE +
      ↪ SALT_SIZE)/8 + NUM_SIZE;
   const ACCOUNT_SIZE: usize = (BALANCE_SIZE + SALT_SIZE +
      ↪ ENERGY_SIZE + HEIGHT)/8 + NUM_SIZE;
   const COMMITMENT_TOTAL_SIZE: usize = NOTE_SIZE + ACCOUNT_SIZE +
      ↪ POLY_1305_TAG_SIZE*2 + NUM_SIZE*2;
```

3.36 CVF-36

- **Severity** Minor
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** note.rs

Description May overflow for big p_d.

Listing 36:

```
37 (self.d.as_num() + &self.p_d + self.b.as_num() + self.t.as_num()
      ↪ ).is_zero()
```

3.37 CVF-37

- **Severity** Moderate
- **Status** Opened
- **Category** Suboptimal
- **Source** params.rs

Description It is suboptimal to have Poseidon instances with different widths as this bloats the codesize.
Recommendation Consider using distinct capacity values within the sponge construction with single width (say, 3).

Listing 37:

```
26 ub hash: PoseidonParams<Fr>,
   ub compress: PoseidonParams<Fr>,
   ub note: PoseidonParams<Fr>,
   ub account: PoseidonParams<Fr>,
30 ub eddsa: PoseidonParams<Fr>,
   ub sponge: PoseidonParams<Fr>,
```