ABDK CONSULTING

SMART CONTRACT
AUDIT

ZkSync

Solidity and Rust

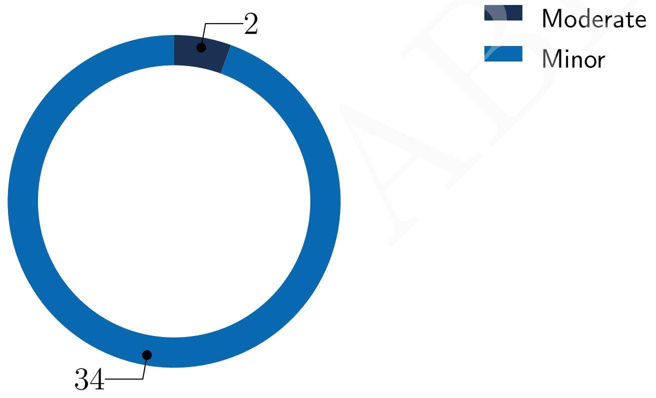
abdk.consulting

SMART CONTRACT AND CIRCUIT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich

29th June 2021

We've been asked to review ZkSync smart contracts related to the NFT functionality. We found only a few issues.



Findings

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

ID	Severity	Category	Status
CVF-28	Minor	Suboptimal	Opened
CVF-29	Minor	Suboptimal	Opened
CVF-30	Minor	Suboptimal	Opened
CVF-31	Minor	Bad naming	Opened
CVF-32	Minor	Flaw	Opened
CVF-33	Minor	Unclear behavior	Opened
CVF-34	Moderate	Flaw	Opened
CVF-35	Minor	Bad datatype	Opened
CVF-36	Minor	Suboptimal	Opened

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	June 28, 2021	D. Khovratovich	Initial Draft
0.2	June 28, 2021	D. Khovratovich	Minor revision
1.0	June 29, 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.

This is our fifth audit of the ZkSync project, the first was made in spring 2020. In this audit we review Solidity smart contracts in the following state.

- Release contract5.1:
  - witness/utils.rs;
  - circuit.rs;
  - utils.rs;
  - AdditionalZkSync.sol;
  - Config.sol;
  - DeployFactory.sol;
  - Events.sol;
  - Governance.sol;
  - Operations.sol;
  - RegenesisMultisig.sol;
  - Storage.sol;
  - TokenGovernance.sol;
  - ZkSync.sol;
  - ZkSyncNFTFactory.sol.

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
- **Category** Readability
- **Status** Opened
- **Source** Operations.sol

Recommendation A hexadecimal literal would be easier to read.

Listing 1:

```
uint256 internal constant LEGACY_MAX_TOKEN = 65535; // 2^16 - 1
```

3.2 CVF-2

- **Severity** Minor
- **Category** Readability
- **Status** Opened
- **Source** Operations.sol

Recommendation This still could be written as a single logical expression: 'return Utils.hashBytesToBytes20(writeDepositPubdataForPriorityQueue(op)) == hashedPubdata || op.tokenId <= LEGACY_MAX_TOKEN && Utils.hashBytesToBytes20(writeLegacyDepositPubdataForPriorityQueue(op)) == hashedPubdata;'.

Listing 2:

```
if (Utils.hashBytesToBytes20(writeDepositPubdataForPriorityQueue
    ↪ (op)) == hashedPubdata) {
      return true;
} else if (
      op.tokenId <= LEGACY_MAX_TOKEN &&
      Utils.hashBytesToBytes20(
          ↪ writeLegacyDepositPubdataForPriorityQueue(op)) ==
          ↪ hashedPubdata
) {
      return true;
} else {
      return false;
}

if (Utils.hashBytesToBytes20(
    ↪ writeFullExitPubdataForPriorityQueue(op)) == hashedPubdata
    ↪ ) {
      return true;
} else if (
      op.tokenId <= LEGACY_MAX_TOKEN &&
      Utils.hashBytesToBytes20(
          ↪ writeLegacyFullExitPubdataForPriorityQueue(op)) ==
          ↪ hashedPubdata
) {
      return true;
} else {
      return false;
}
```

3.3 CVF-3

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

Description Here the first 16 bits of token ID are dropped assuming that either these bits are zero, or the token is non-fungible, however it is not obvious that at least one of this conditions is always satisfied.

Recommendation Consider calculating the inverted 'multi_' or of the dropped bits and using this value instead of the 'is_fungible_token' flag when deciding whether an old signature is valid or not.

Listing 3:

```
1358 serialized_tx_bits_old1.extend_from_slice(&cur.token.get_bits_be()[16..32]);

1381 serialized_tx_bits_old2.extend_from_slice(&cur.token.get_bits_be()[16..32]);

2003 serialized_tx_bits_old1.extend_from_slice(&cur.token.get_bits_be()[16..32]);

2018 serialized_tx_bits_old2.extend_from_slice(&cur.token.get_bits_be()[16..32]);

3002 serialized_tx_bits_old1.extend_from_slice(&cur.token.get_bits_be()[16..32]);

3017 serialized_tx_bits_old2.extend_from_slice(&cur.token.get_bits_be()[16..32]);

3976 serialized_tx_bits_old1.extend_from_slice(&cur.token.get_bits_be()[16..32]);

4016 serialized_tx_bits_old2.extend_from_slice(&cur.token.get_bits_be()[16..32]);

4288 serialized_tx_bits_old.extend_from_slice(&cur.token.get_bits_be()[16..32]);
```

3.4 CVF-4

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

Description The name is confusing, as usually "reversed" bits are the same bits but in the reversed order, while “inverted” bits are what this function actually calculates.
Recommendation Consider renaming.

Listing 4:

```
reversed_tx_type_bits_be(tx_type: u8) -> Vec<Boolean> {
```

3.5 CVF-5

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

Recommendation These variables are redundant, as their values are used only once.

Listing 5:

```
let tx_bytes = get_bytes!(transfer_op);

let tx_bytes = get_bytes!(transfer_op);

let tx_bytes = get_bytes!(change_pubkey_op);

let tx_bytes = get_bytes!(withdraw_op);

let tx_bytes = get_bytes!(forced_exit_op);
```

3.6 CVF-6

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

Recommendation This expression could be simplified as: ’(a « i) & 0x80u8 != 0 or as a & (0x80u8 » i) != 0’.

Listing 6:

```
if (a & (1u8 << (7 - i))) != 0 {
```

3.7 CVF-7

- **Severity** Minor
- **Category** Bad naming
- **Status** Opened
- **Source** Governance.sol

Recommendation Consider renaming to "NewDefaultNFTFactory”.

Listing 7:

```
event SetDefaultNFTFactory(address indexed factory);
```

3.8 CVF-8

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** Governance.sol

Recommendation The parameter should probably have some more specific type.

Listing 8:

```
event SetDefaultNFTFactory(address indexed factory);
```

3.9 CVF-9

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

Recommendation The first cast is redundant.

Listing 9:

```
require(address(_factory) != address(0), "mb1"); // Factory
    ↪ should be non zero
```

3.10 CVF-10

- **Severity** Moderate
- **Category** Flaw
- **Status** Opened
- **Source** AdditionalZkSync.sol

Recommendation The approval of a notice period cut is not bound to a particular upgrade. This function should only be callable when an upgrade is already scheduled.

Listing 10:

```
function cutUpgradeNoticePeriod() external {
```

3.11 CVF-11

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

Recommendation A bit mask would be more efficient than a mapping.

Listing 11:

```
require(securityCouncilApproves[id] == false);
securityCouncilApproves[id] = true;
```

3.12 CVF-12

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

Description If any thresholds coincide, then the notice period will be written twice to the storage and an event will be logged twice as well.
Recommendation Consider handling this case explicitly.

Listing 12:

```
if (numberOfApprovalsFromSecurityCouncil ==
  ↪ SECURITY_COUNCIL_2_WEEKS_THRESHOLD) {

if (numberOfApprovalsFromSecurityCouncil ==
  ↪ SECURITY_COUNCIL_1_WEEK_THRESHOLD) {

}
```

3.13 CVF-13

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

Description Writing approved upgrade period is waste of gas.
Recommendation Just calculate it when needed.

Listing 13:

```
approvedUpgradeNoticePeriod = 2 weeks;

approvedUpgradeNoticePeriod = 1 weeks;

approvedUpgradeNoticePeriod = 3 days;
```

3.14 CVF-14

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

Description The value of the '_pubkeyHash' argument is ignored here, so one could set timer with one value and then, after the reset timelock time passed, actually set another value. Probably, not an issue.

Listing 14:

```
authFactsResetTimer[msg.sender][_nonce] = block.timestamp;
```

3.15 CVF-15

- **Severity** Minor
- **Category** Overflow/Underflow
- **Status** Opened
- **Source** AdditionalZkSync.sol

Description Overflow is possible (in theory) when converting to uint32.
Recommendation Consider calculating the minimum of 256-bit numbers and only then convert to uint32.

Listing 15:

```
uint32 blocksToRevert = Utils.minU32(uint32(_blocksToRevert.
    ↪ length), blocksCommitted - totalBlocksExecuted);
```

3.16 CVF-16

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

Recommendation In Solidity, types smaller than 256 bits are not more efficient, than 256-bit types. Probably, just using uint256 here would be fine.

Listing 16:

```
uint8 constant ADDRESS_FOOTPRINT_OFFSET = 0;
uint8 constant ADDRESS_SIZE_BITS = 160;

uint8 constant CREATOR_ID_FOOTPRINT_OFFSET =
    ↪ ADDRESS_FOOTPRINT_OFFSET + ADDRESS_SIZE_BITS;
uint8 constant CREATOR_ID_SIZE_BITS = 32;

uint8 constant SERIAL_ID_FOOTPRINT_OFFSET =
    ↪ CREATOR_ID_FOOTPRINT_OFFSET + CREATOR_ID_SIZE_BITS;
uint8 constant SERIAL_ID_SIZE_BITS = 32;
```

3.17 CVF-17

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

Description These two functions do exactly the same.
Recommendation Consider extracting common code into a utility function.

Listing 17:

```
function upgradeCanceled() external override {

function upgradeFinishes() external override {
```

3.18 CVF-18

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

Description These events are emitted even if the notice period didn't actually change.

Listing 18:

```
emit NoticePeriodChange(approvedUpgradeNoticePeriod);

emit NoticePeriodChange(approvedUpgradeNoticePeriod);

emit NoticePeriodChange(approvedUpgradeNoticePeriod);

emit NoticePeriodChange(approvedUpgradeNoticePeriod);
```

3.19 CVF-19

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** ZkSync.sol

Recommendation The types of the variables should be more specific: (Governance _governanceAddress, Verifier _verifierAddress, AdditionalZKSync _additionalZkSync, bytes32 _genesisStateHash) = abi.decode(initializationParameters, (Governance, Verifier, AdditionalZKSync, bytes32));

Listing 19:

```
(address _governanceAddress, address _verifierAddress, address
   ↪ _additionalZkSync, bytes32 _genesisStateHash) =
     abi.decode(initializationParameters, (address, address,
        ↪ address, bytes32));
```

3.20 CVF-20

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

Recommendation This type conversion is redundant, just change the type of the '_additionalZkSync' variable.

Listing 20:

```
additionalZkSync = AdditionalZkSync(_additionalZkSync);
```

3.21 CVF-21

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

Description Why the cast to an address is necessary here? What is the value of the macro?

Listing 21:

```
additionalZkSync = AdditionalZkSync(address($$(NEW_ADDITIONAL_ZKSYNC_ADDRESS)));
```

3.22 CVF-22

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

Recommendation It is possible to just delegate all the non-processed invocation, but implementing a fallback function.

Listing 22:

```
function cutUpgradeNoticePeriod() external {

function cancelOutstandingDepositsForExodusMode(uint64 _n, bytes[] memory _depositsPubdata) external {
```

3.23 CVF-23

- **Severity** Minor
- **Category** Readability
- **Status** Opened
- **Source** ZkSync.sol

Recommendation It is not obvious form the code, that op.tokenId <= MAX_FUNGIBLE_TOKEN_ID guarantees that uint16(op.tokenId) wouldn't overflow. Safe cast would be more clear.

Listing 23:

```
require(op.tokenId <= MAX_FUNGIBLE_TOKEN_ID, "mf1");
  withdrawOrStore(uint16(op.tokenId), op.owner, op.amount);
```

```
require(op.tokenId <= MAX_FUNGIBLE_TOKEN_ID, "mf2");
  withdrawOrStore(uint16(op.tokenId), op.target, op.amount);
```

3.24 CVF-24

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** ZkSync.sol

Recommendation This variable should have type "AdditionalZkSync".

Listing 24:

```
address _target = address(additionalZkSync);
```

3.25 CVF-25

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

Description Only the new listing fee token is logged, but not the new listing fee.
Recommendation Consider either adding a listing fee parameter to this event or emit another 'ListingFeeUpdate' event.

Listing 25:

```
emit ListingFeeTokenUpdate(_newListingFeeToken);
```

3.26 CVF-26

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

Description This event is emitted even if nothing were actually changed.

Listing 26:

```
emit ListingFeeTokenUpdate(_newListingFeeToken);

emit ListingFeeUpdate(_newListingFee);

emit ListingCapUpdate(_newListingCap);

emit TreasuryUpdate(_newTreasury);
```

3.27 CVF-27

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

Recommendation A bit mask would be more gas-efficient.

Listing 27:

```
mapping(uint256 => bool) internal securityCouncilApproves;
```

3.28 CVF-28

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

Recommendation This variable would be redundant if a bit mask would be used for 'securityCouncilApproves', as it is quite cheap to count “one” bits in a word.

Listing 28:

```
uint256 internal numberOfApprovalsFromSecurityCouncil;
```

3.29 CVF-29

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

Recommendation A bit mask would be more gas-efficient.

Listing 29:

```
mapping(uint256 => bool) internal securityCouncilApproves;
```

3.30 CVF-30

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

Recommendation Using a bit mask for "securityCouncilApproves" would make this unnecessary, as it is quite cheap to count "one" bits is a word.

Listing 30:

```
uint256 internal numberOfApprovalsFromSecurityCouncil;
```

3.31 CVF-31

- **Severity** Minor
- **Category** Bad naming
- **Status** Opened
- **Source** RegenesisMultisig.sol

Description The meaning of this important storage variable is unclear from its name.
Recommendation Consider adding a documentation comment.

Listing 31:

```
uint256 securityCouncilThreshold;
```

3.32 CVF-32

- **Severity** Minor
- **Category** Flaw
- **Status** Opened
- **Source** RegenesisMultisig.sol

Description There is not range check for the value of the "threshold" argument.
Recommendation Consider checking that it doesn't exceed the total number of security council members.

Listing 32:

```
constructor(uint256 threshold) Ownable(msg.sender) {
```

3.33 CVF-33

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

Recommendation This function should emit some event.

Listing 33:

```
function submitHash(bytes32 _oldRootHash, bytes32 _newRootHash)
    external {
```

3.34 CVF-34

- **Severity** Moderate
- **Category** Flaw
- **Status** Opened
- **Source** RegenesisMultisig.sol

Recommendation There should be some way to revoke the approval in case some issue was found after the approval was made.

Listing 34:

```
function approveHash(bytes32 _oldRootHash, bytes32 _newRootHash)
     external {
```

3.35 CVF-35

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** DeployFactory.sol

Recommendation This argument should have type “TokenGovernance”.

Listing 35:

```
address _finalGovernor
```

3.36 CVF-36

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

Description This event is logged when then number of the security council member that approved cutting the notice period crosses certain thresholds.

Recommendation It would be more logical to emit an even each time a security council member approves the cut, and have the address of the council member, the current number of approves, and the current notice period as event parameters. This would tell users how close the protocol is to cutting the notice period.

Listing 36:

```
/// @notice Notice period changed
event NoticePeriodChange(uint256 newNoticePeriod);
```