ABDK CONSULTING

SMART CONTRACT AUDIT

Matter Labs

ZkSync Phase 2 p.2

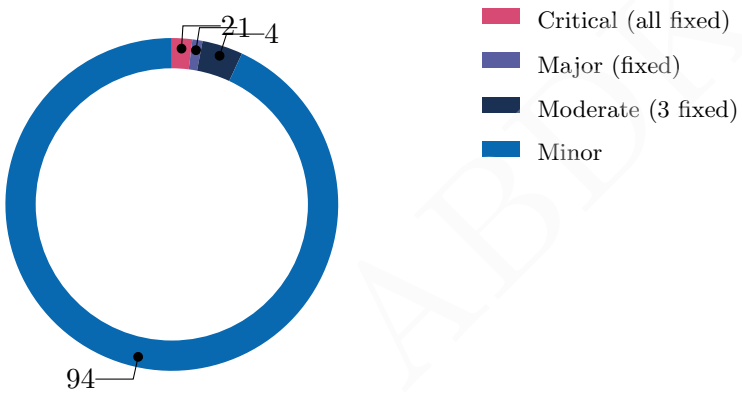
abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

Mikhail Vladimirov and Dmitry Khovratovich

3 February 2021

We have found a number of issues in the code but all major ones were fixed in the subsequent releases. The latest release we have audited was this one.



Findings

ID	Severity	Subject	Status
CVF-1	Minor	No access level for the `EMPTY_STRING_KECCAK`	OPENED
CVF-2	Moderate	Incorrect `publicDataOffset` field location	FIXED
CVF-3	Moderate	Incorrect `blockNumber` and `feeAccount` location	FIXED
CVF-4	Minor	Magic number	OPENED
CVF-5	Minor	No comment for the `upgradeNoticePeriodStarted()` function	FIXED
CVF-6	Minor	No comment for the `upgradePreparationStarted()` function	FIXED
CVF-7	Minor	No comment for the `upgradeCanceled()` function	FIXED
CVF-8	Minor	No comment for the `upgradeFinishes()` function	FIXED
CVF-9	Minor	Lost check in the `isReadyForUpgrade()`	OPENED
CVF-10	Minor	No check for the `_genesisStateHash`	OPENED
CVF-11	Minor	Uninitialized `numberOfPendingWithdrawals_DEPRECATED` variable	OPENED
CVF-12	Minor	Suboptimal check	OPENED
CVF-13	Minor	Incorrect `_tokenAddress` type	FIXED
CVF-14	Minor	No check for `balanceDiff <= _maxAmount, "wtg12"`	OPENED
CVF-15	Minor	Similar variable naming	FIXED
CVF-16	Minor	Multiple `priorityRequests[id]` calculating	OPENED
CVF-17	Minor	Overflow in `balanceToWithdraw += op.amount`	OPENED
CVF-18	Minor	Not validated `_zkSyncAddress`	OPENED
CVF-19	Minor	Not validated `_zkSyncAddress-2`	OPENED

ID	Severity	Subject	Status
CVF-20	Minor	Gas spending	OPENED
CVF-21	Minor	Redundant calculating	OPENED
CVF-22	Minor	Unseparated require statements	OPENED
CVF-23	Minor	Redundant assignment	FIXED
CVF-24	Minor	Overflow in `commitBlocks` operation	OPENED
CVF-25	Minor	Redundant logging	OPENED
CVF-26	Minor	Overflow in the `withdrawOrStore` function	OPENED
CVF-27	Minor	Redundant conversion	OPENED
CVF-28	Minor	Several `_blockExecuteData.storedBlock.blockN` calculating	OPENED
CVF-29	Minor	Redundant `_blockExecuteData.storedBlock.blockN` `<= totalBlocksProofed` check	FIXED
CVF-30	Minor	Incorrect `totalBlocksProofed` naming	FIXED
CVF-31	Minor	Multiply pubdata hashing	OPENED
CVF-32	Minor	Overflow in the `executeOneBlock`	OPENED
CVF-33	Minor	Redundant block logging	OPENED
CVF-34	Minor	No event logging in the `proveBlocks`	FIXED
CVF-35	Moderate	Inefficient data structure	OPENED
CVF-36	Minor	Inefficient assignment	OPENED
CVF-37	Critical	No check for the `_proof.commitments`	FIXED
CVF-38	Minor	Inconsistent function naming	OPENED
CVF-39	Critical	Dangerous conditional statement	FIXED
CVF-40	Major	Suboptimal assignment `totalBlocksProofed = totalBlocksCommitted;`	FIXED
CVF-41	Minor	Unclear condition `Franklin -> ZkSync`	FIXED
CVF-42	Minor	Incorrect comment	OPENED
CVF-43	Minor	Uncommon `uint(-1)` form	OPENED
CVF-44	Minor	Redundant `commitment = commitment & mask;` assignment	OPENED

ID	Severity	Subject	Status
CVF-45	Minor	The redundant `emitDepositCommitEvent` function	OPENED
CVF-46	Minor	Suboptimal `totalBlocksProofed = totalBlocksCommitted;` assign	OPENED
CVF-47	Minor	Redundant `emitFullExitCommitEvent` function	OPENED
CVF-48	Minor	Inefficient `uint32` counter using	FIXED
CVF-49	Minor	The redundant `pubdataOffset CHUNK_BYTES == 0, "fcso2"` check	OPENED
CVF-50	Minor	Redundant `pubdataOffset / CHUNK_SIZE` calculating	FIXED
CVF-51	Minor	Suboptimal bytes allocating	OPENED
CVF-52	Moderate	Line refactoring	FIXED
CVF-53	Minor	Over-complicated `CREATE2 public key` method	OPENED
CVF-54	Minor	Unclear purpose `0` assignment	OPENED
CVF-55	Minor	Expensive hashing	OPENED
CVF-56	Minor	Overflow in the `addPriorityRequest` function	OPENED
CVF-57	Minor	Unclear logging purpose	OPENED
CVF-58	Minor	Dangerous function call	OPENED
CVF-59	Minor	Redundant payable function	OPENED
CVF-60	Minor	Draft code	FIXED
CVF-61	Minor	The magic number `500000`	OPENED
CVF-62	Minor	Redundant variable	OPENED
CVF-63	Minor	Suboptimal assignment `uint256 mask = (~uint256(0)) >> 3;`	FIXED
CVF-64	Minor	Redundant cast	FIXED
CVF-65	Minor	Suboptimal constant passing	OPENED
CVF-66	Minor	Redundant `PendingWithdrawal_DEPRECATED` struct	OPENED
CVF-67	Minor	Suboptimal `totalBlocksVerified` variable	FIXED

ID	Severity	Subject	Status
CVF-68	Minor	Confusing `totalCommittedPriorityRequests` variable name	OPENED
CVF-69	Minor	Incorrect comment	FIXED
CVF-70	Minor	Non-existent `StoredBlockInfo` structure	FIXED
CVF-71	Minor	The comment to non-existent `StoredBlockInfo` structure	FIXED
CVF-72	Minor	Unclear comment	OPENED
CVF-73	Minor	No check for the `callSuccess`	OPENED
CVF-74	Minor	No check for the `callSuccess`-2	OPENED
CVF-75	Minor	Redundant `ETH_WITHDRAWAL_GAS_LIMIT` variable	OPENED
CVF-76	Minor	No check for the returned value	OPENED
CVF-77	Minor	The `Config` contract converting	FIXED
CVF-78	Minor	Small `50 000` limit	OPENED
CVF-79	Minor	No access level for the `ERC20_WITHDRAWAL_GAS_LIMIT` constant	OPENED
CVF-80	Minor	No access level for the `MASS_FULL_EXIT_PERIOD` constant	OPENED
CVF-81	Minor	No access level for the `TIME_TO_WITHDRAW_FUNDS_FROM_FULL_EXI` constant	OPENED
CVF-82	Minor	No access level for the `UPGRADE_NOTICE_PERIOD` constant	OPENED
CVF-83	Minor	No access level for the `COMMIT_TIMESTAMP_NOT_OLDER` constant	OPENED
CVF-84	Minor	No access level for the `COMMIT_TIMESTAMP_APPROXIMATION_DELTA` constant	OPENED
CVF-85	Minor	Redundant `Add` in the variable name	OPENED
CVF-86	Minor	Redundant `queueStartIndex` property	OPENED
CVF-87	Minor	Unclear `queueEndIndex` index purpose	OPENED
CVF-88	Minor	Not emitted `PendingWithdrawalsComplete` event	OPENED

ID	Severity	Subject	Status
CVF-89	Minor	Redundant `queueStartIndex` property	OPENED
CVF-90	Minor	Unseparated `TokenPausedUpdate` event	OPENED
CVF-91	Minor	Unread `address => bool` mapping	OPENED
CVF-92	Minor	Unclear identification	OPENED
CVF-93	Minor	Redundant `"52 constructor() {}"` line	FIXED
CVF-94	Minor	The `because it` typo	OPENED
CVF-95	Minor	Increased gas consumption	OPENED
CVF-96	Minor	Suboptimal slice copying	OPENED
CVF-97	Minor	Suboptimal slice copying-2	OPENED
CVF-98	Minor	Ignored operation type	OPENED
CVF-99	Minor	Unclear `opType` field behavior	FIXED
CVF-100	Minor	Confusing `ChangePubkeyType` name	OPENED
CVF-101	Minor	Unusual `Create2` name	OPENED

Contents

Document properties

Version

Version	Date	Author	Description
0.1	Dec. 25, 2020	D. Khovratovich and M. Vladimirov	Initial Draft
0.2	Dec. 26, 2020	D. Khovratovich and M. Vladimirov	Findings collected
0.3	Dec. 28, 2020	D. Khovratovich and M. Vladimirov	Minor Revision
1.0	Jan. 11, 2021	D. Khovratovich and M. Vladimirov	Release
1.1	Feb. 02, 2021	D. Khovratovich and M. Vladimirov	Comments added

Contact

D. Khovratovich

dmitry@abdkconsulting.com

1 Introduction

We were asked to review the ZKSync smart contracts given in separate files. This is our second 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 20 Nov 2020:
  - Config.sol
  - Events.sol
  - Governance.sol
  - Operations.sol
  - Proxy.sol
  - Storage.sol
  - UpgradeGateKeeper.sol
  - Utils.sol
  - Verifier.sol
- Release 27 Nov 2020:
  - ZkSync.sol

The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

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

1.3 About Customer

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

1.4 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication.

2 Detailed Results

2.1 CVF-1 No access level for the EMPTY_STRING_KECCAK

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

Description There is no access level specified for this constant, so internal access will be used by default.
Recommendation Consider specifying access level explicitly.

Listing 1: No access level for the `EMPTY\_STRING\_KECCAK`

```
28      bytes32 constant EMPTY_STRING_KECCAK = 0
    ↪ xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470
    ↪ ;
```

2.2 CVF-2 Incorrect publicDataOffset field location

- **Severity** Moderate
- **Status** FIXED
- **Category** Suboptimal
- **Source** ZkSync.sol

Description Struct fields are tightly packed but never cross word boundary, so 28 padding bytes will be inserted after this field.
Recommendation Consider moving the field after `ehtWitness`.

Listing 2: Incorrect `publicDataOffset` field location

```
35  uint32 publicDataOffset;
```

2.3 CVF-3 Incorrect blockNumber and feeAccount location

- **Severity** Moderate
- **Status** FIXED
- **Category** Suboptimal
- **Source** ZkSync.sol

Description Struct fields are tightly packed but never cross word boundary, so 24 padding bytes will be inserted after this field. Moving these fields after `onChainOperations` would address this issue.
Recommendation Consider moving the fields after the `onChainOperations`.

Listing 3: Incorrect `blockNumber` and `feeAccount` location

```
41      uint32 blockNumber;
42      uint32 feeAccount;
```

2.4 CVF-4 The magic number

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

Recommendation 16 in this line should be made a named constant.

Listing 4: Magic number

```
63        uint256[16] subproofsLimbs;
```

2.5 CVF-5 No comment for the upgradeNoticePeriodStarted() function

- **Severity** Minor
- **Status** FIXED
- **Category** Documentation
- **Source** ZkSync.sol

Description It should be noted in the comments that access to this function is controlled by the Proxy. Otherwise it looks like a security hole.
Recommendation Add an explicit comment.

Listing 5: No comment for the `upgradeNoticePeriodStarted()` function

```
74        function upgradeNoticePeriodStarted() external override
    ↪ {}
```

2.6 CVF-6 No comment for the upgradePreparationStarted() function

- **Severity** Minor
- **Status** FIXED
- **Category** Documentation
- **Source** ZkSync.sol

Description It should be noted in comments that access to this function is controlled by the Proxy. Otherwise it looks like a security hole.
Recommendation Add an explicit comment.

Listing 6: No comment for the `upgradePreparationStarted()` function

```
77        function upgradePreparationStarted() external override {
```

2.7 CVF-7 No comment for the upgradeCanceled()

function

- **Severity** Minor
- **Status** FIXED
- **Category** Documentation
- **Source** ZkSync.sol

Description It should be noted in comments that access to this function is controlled by the Proxy. Otherwise it looks like a security hole.
Recommendation Add an explicit comment.

Listing 7: No comment for the `upgradeCanceled()` function

```
83      function upgradeCanceled() external override {
```

2.8 CVF-8 No comment for the upgradeFinishes()

function

- **Severity** Minor
- **Status** FIXED
- **Category** Documentation
- **Source** ZkSync.sol

Description It should be noted in comments that access to this function is controlled by the Proxy. Otherwise it looks like a security hole.
Recommendation Add an explicit comment.

Listing 8: No comment for the `upgradeFinishes()` function

```
89      function upgradeFinishes() external override {
```

2.9 CVF-9 Lost check in the isReadyForUpgrade()

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

Description In the previous version it was checked that there is no open priority requests. Why this check is not needed anymore?
Recommendation Perhaps, there should be check.
Client comment We founded that checking that there is no priority requests is not enough, so we changed upgrade logic. Now user have enough time to withdraw funds in case of some evil upgrade due to big NOTICE_PERIOD.

Listing 9: Lost check inisReadyForUpgrade()

```
97      return !exodusMode;
```

2.10 CVF-10 No check for the _genesisStateHash

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

Description Genesis state is passed as a constructor argument, which means that genesis is not guaranteed to be empty.
Client comment This function will never be called on the PROD.

Listing 10: No check for the `_genesisStateHash`

```
116         StoredBlockInfo(0, 0, EMPTY_STRING_KECCAK, 0,
    ↪ _genesisStateHash, bytes32(0));
```

2.11 CVF-11 Uninitialized numberOfPendingWithdrawals_DEPRECATED variable

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

Description The variable is used without being initialized.
Recommendation Initialize the variable.
Client comment This value is deprecated starting from this upgrade.

Listing 11: Uninitialized `numberOfPendingWithdrawals_ DEPRECATED` variable

```
128         require(numberOfPendingWithdrawals_DEPRECATED == 0, "upg4
    ↪ "); // pending withdrawal is not used anymore
```

2.12 CVF-12 Suboptimal check

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

Description The block.timestamp in this line would probably be more appropriate for the check at line 287.
Recommendation Consider using `block.tomestamp`.

Listing 12: Suboptimal check

```
139         0,
```

2.13 CVF-13 Incorrect _tokenAddress type

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

Recommendation The `_tokenAddress` in this line should have type IERC20.

Listing 13: Incorrect `_tokenAddress` type

```
154     address _tokenAddress,
```

2.14 CVF-14 Check absence

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

Description It is never checked that `_amount <= _maxAmount` and for cases where `_to == this`, the balance change doesn't accurately reflect the amount of token transferred.
Recommendation Consider adding explicit check that `_amount <= _maxAmount`.

Listing 14: No check for `balanceDiff <= \_maxAmount, "wtg12"`

```
166       require(balanceDiff <= _maxAmount, "wtg12"); // wtg12 -
    ↪ rollup balance difference (before and after transfer) is
    ↪ bigger than _maxAmount
```

2.15 CVF-15 Similar variable naming

- **Severity** Minor
- **Status** FIXED
- **Category** Documentation
- **Source** ZkSync.sol

Description Names `depositPubdata` and `depositsPubdata` look too similar and get confused.
Recommendation Consider adding underscore to the `depositsPubdata`

Listing 15: Similar variable naming

```
182       bytes memory depositPubdata = depositsPubdata[
    ↪ currentDepositIdx];
```

2.16 CVF-16 Multiple priorityRequests[id] calculating

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

Description The `priorityRequests[id]` is calculated multiple times.
Recommendation Consider calculating once and caching in a local variable.

Listing 16: Multiple `priorityRequests[id]` calculating

```
183         require(Utils.hashBytesToBytes20(depositPubdata) ==
    ↪ priorityRequests[id].hashedPubData, "coe03");
```

2.17 CVF-17 Overflow in balanceToWithdraw += op.amount

- **Severity** Minor (degraded from Major)
- **Status** OPENED
- **Category** Overflow
- **Source** ZkSync.sol

Description Overflow is possible in this line.
Client comment The DepositERC20 function allows deposit only 2**104-1 tokens per call. Overflow is impossible.

Listing 17: Overflow in `balanceToWithdraw += op.amount`

```
188         balancesToWithdraw[packedBalanceKey].balanceToWithdraw +=
    ↪ op.amount;
```

2.18 CVF-18 Not validated _zkSyncAddress

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

Description The _zkSyncAddress is not validated. Probably, not an issue.
Client comment Not an issue.

Listing 18: Not validated `\_zkSyncAddress`

```
206         function depositETH(address _zkSyncAddress) external
    ↪ payable nonReentrant {
```

2.19 CVF-19 Not validated _zkSyncAddress-2

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

Description The _zkSyncAddress is not validated. Probably, not an issue.
Client comment Not a issue.

Listing 19: Not validated `_zkSyncAddress`

```
206     function depositETH(address _zkSyncAddress) external
    ↪ payable nonReentrant {
```

2.20 CVF-20 Gas spending

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

Description This operation is suboptimal, as Solidity will try to preserve the `balanceToWithdraw` field, i.e. read the slot, then use some bit wise operations to set the `gasREserveValue` preserving the `balanceToWithdraw`, then write it back. This is even more suboptimal when current the `balanceToWithdraw` is already non-zero.
Recommendation Consider using the lowest bit for gas reserve the next code:

```
if (balanceToWithdraw == 0)
  balanceToWithdraw = 1;
```

In the rest of the code, just multiply values by two before setting them to the `balanceToWithdraw` and divide by two before using stored the `balanceToWithdraw` value. This will make code simpler and more efficient.
Client comment We will skip due to very small relative profit.

Listing 20: Gas spending

```
273     balancesToWithdraw[packedBalanceKey].gasReserveValue =
    ↪ FILLED_GAS_RESERVE_VALUE;
```

2.21 CVF-21 Redundant calculating

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

Description This doesn't need to be calculated in case the `timestampNoTooBig` is false.
Client comment Not an issue.

Listing 21: Redundant calculating

```
289     bool timestampNotTooBig = _newBlock.timestamp <= block.
    ↪ timestamp + COMMIT_TIMESTAMP_APPROXIMATION_DELTA;
```

2.22 CVF-22 Unseparated require statements

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

Description This checks two different conditions, but outputs the same error message regardless of what condition failed.
Recommendation Consider using two separate require statements.
Client comment Not an issue.

Listing 22: Unseparated require statements

```
290     require(timestampNotTooSmall && timestampNotTooBig, "
    ↪ tms12"); // New block timestamp is not valid
```

2.23 CVF-23 Redundant assignment

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

Description This assignment is redundant, as _lastCommittedBlockData is not used after it.
Recommendation Just use _lastCommittedBlockData instead of lastCommittedBlock.

Listing 23: Redundant assignment

```
323     StoredBlockInfo memory lastCommittedBlock =
    ↪ _lastCommittedBlockData;
```

2.24 CVF-24 Possible overflow in the commitBlocks operation

- **Severity** Minor (degraded from Moderate)
- **Status** OPENED
- **Category** Overflow
- **Source** ZkSync.sol

Description Overflow is possible in this line
Client comment Overflow is impossible.

Listing 24: Overflow in `commitBlocks` operation

```
329         committedPriorityRequests += lastCommittedBlock.
    ↪ priorityOperations;
```

2.25 CVF-25 Redundant logging

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

Description There is no point to log the numbers of intermediary blocks without any additional information.
Recommendation Just logging the number of the final committed block would be enough.

Listing 25: Redundant logging

```
332         emit BlockCommit(lastCommittedBlock.blockNumber);
```

2.26 CVF-26 Possible overflow in the withdrawOrStore

- **Severity** Minor (degraded from Moderate)
- **Status** OPENED
- **Category** Overflow
- **Source** ZkSync.sol

Description Overflow is possible in this line.
Client comment Overflow is impossible.

Listing 26: Overflow in the `withdrawOrStoren`

```
335         totalBlocksCommitted += uint32(_newBlocksData.length);
```

2.27 CVF-27 Redundant conversion

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

Description The double conversion is probably redundant.

Listing 27: Redundant conversion

```
353         address payable toPayable = address(uint160(_recipient))
    ↪ ;
```

2.28 CVF-28 Several _blockExecuteData.storedBlock.blockNumber calculating

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

Description The `_blockExecuteData.storedBlock.blockNumber` is calculated several times in this function.
Recommendation Consider calculating once and caching in a local variable.
Client comment We will skip due to very small relative profit.

Listing 28: Several `_blockExecuteData.storedBlock.blockNumber` calculating

```
381         require(_blockExecuteData.storedBlock.blockNumber ==
    ↪ totalBlocksVerified + _executedBlockIdx + 1, "exe11"); //
    ↪ Execute blocks in order
```

2.29 CVF-29 Redundant check

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

Recommendation It could be checked once in the `executeBlocks` for the whole batch.

Listing 29: `_blockExecuteData.storedBlock.blockNumber <= totalBlocksProofed` check

```
382         require(_blockExecuteData.storedBlock.blockNumber <=
    ↪ totalBlocksProofed, "exe03"); // Can't execute blocks more
    ↪ then committed and proofed currently.
```

2.30 CVF-30 Incorrect totalBlocksProofed naming

- **Severity** Minor
- **Status** FIXED
- **Category** Documentation
- **Source** ZkSync.sol

Recommendation The `totalBlocksProven` would be a more correct name.

Listing 30: Incorrect `totalBlocksProofed` naming

```
382         require(_blockExecuteData.storedBlock.blockNumber <=
    ↪ totalBlocksProofed, "exe03"); // Can't execute blocks more
    ↪ then committed and proofed currently.
```

2.31 CVF-31 Multiply `pubData` hashing

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

Recommendation Passing all pubData for a block as one bytes array would make it possible to hash it all at once.
Client comment Will be fixed in the next versions.

Listing 31: Multiply `pubData` hashing

```
403         pendingOnchainOpsHash = Utils.concatHash(
    ↪ pendingOnchainOpsHash, pubData);
```

2.32 CVF-32 Overflow in the executeOneBlock

- **Severity** Minor
- **Status** OPENED
- **Category** Overflow
- **Source** ZkSync.sol

Description Overflow is possible in this line.
Client comment Overflow is impossible.

Listing 32: Overflow in the `executeOneBlock`

```
416         uint32 nBlocks = uint32(_blocksData.length);
```

2.33 CVF-33 Redundant block logging

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

Description It is redundant to log the number of all intermediary blocks without any additional information.
Recommendation Logging just the final block number in a batch would be enough.

Listing 33: Redundant block logging

```
420         emit BlockVerification(_blocksData[i].storedBlock.
    ↪ blockNumber);
```

2.34 CVF-34 No event logging in the `proveBlocks`

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

Description The `proveBlocks` function should log some event.

Listing 34: No event logging in the `proveBlocks`

```
432         function proofBlocks(
```

2.35 CVF-35 Inefficient data structure

- **Severity** Moderate
- **Status** OPENED
- **Category** Suboptimal
- **Source** ZkSync.sol

Recommendation Passing a single array of structs with two fields would be more efficient than passing two separate arrays. Also this would make length check unnecessary.
Client comment Can be fixed in the next versions.

Listing 35: Inefficient data structure

```
433         StoredBlockInfo[] memory _committedBlocks,
434         uint256[] memory _commitmentIdxs,
```

2.36 CVF-36 Inefficient assignment

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

Recommendation Starting the `totalBlocksProofed` from 1 would makes this `+1` in many places unnecessary
Client comment Can be fixed in the next versions.

Listing 36: Inefficient assignment

```
441         hashStoredBlockInfo(_committedBlocks[i]) ==
    ↪ storedBlockHashes[currentTotalBlocksProofed + 1],
```

2.37 CVF-37 No check for the `_proof.commitments`

- **Severity** Critical
- **Status** FIXED
- **Category** Unclear behavior
- **Source** ZkSync.sol

Description It is not guaranteed that all elements of the `_proof.commitments` will be set here, so some elements could still contain the values passed by the caller. Is this fine?

Listing 37: No check for the `_proof.commitments`

```
446         _proof.commitments[_commitmentIdxs[i]] = uint256(
    ↪ _committedBlocks[i].commitment);
```

2.38 CVF-38 Inconsistent function naming

- **Severity** Minor
- **Status** OPENED
- **Category** Documentation
- **Source** ZkSync.sol

Description Some variables have different names in the implementation of this function, which makes the code difficult to read.
Recommendation Consider using consistent naming.

Listing 38: Inconsistent function naming

```
450         verifier.verifyAggregatedProof(
451         _proof.recursiveInput,
452         _proof.proof,
453         _proof.vkIndexes,
454         _proof.commitments,
455         _proof.subproofsLimbs,
456         true
```

2.39 CVF-39 Dangerous conditional statement

- **Severity** Critical
- **Status** FIXED
- **Category** Suboptimal
- **Source** ZkSync.sol

Description The idea of this conditional statement is to unproved reverted block, while currently it proves non-reverted blocks.
Recommendation Should be < instead of > here.

Listing 39: Dangerous conditional statement

```
484     if (totalBlocksCommitted > totalBlocksProofed) {
```

2.40 CVF-40 Suboptimal assignment totalBlocksProofed = totalBlocksCommitted;

- **Severity** Major
- **Status** FIXED
- **Category** Suboptimal
- **Source** ZkSync.sol

Description This arbitrarily increases the counter of proven blocks, even if no blocks are reverted at all.
Client comment Not an issue after the fix of CVF-39.

Listing 40: Suboptimal assignment `totalBlocksProofed = totalBlocksCommitted;`

```
485     totalBlocksProofed = totalBlocksCommitted;
```

2.41 CVF-41 Unclear name

- **Severity** Minor
- **Status** FIXED
- **Category** Documentation
- **Source** ZkSync.sol

Description Perhaps, there should be `Franklin -> ZkSync`?

Listing 41: Unclear name `Franklin -> ZkSync`

```
510     /// @notice Withdraws token from Franklin to root chain
    ↪ in case of exodus mode. User must provide proof that he
    ↪ owns funds
```

2.42 CVF-42 Incorrect comment

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

Recommendation This should probably be extracted into an utility function.
Client comment Can be fixed in the next versions.

Listing 42: Incorrect comment

```
529         uint256(sha256(abi.encodePacked(_storedBlockInfo.
    ↪ stateHash, _accountId, msg.sender, _tokenId, _amount)));
```

2.43 CVF-43 Uncommon uint(-1) form

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

Description Form `uint(-1)` is more common than `~uint(0)`. The whole mask should be made a named constant. The variable is redundant, as its value is constant and is used only once.
noindent**Client comment** Deprecated part of the code.

Listing 43: Uncommon `uint(-1)` form

```
531         uint256 mask = (~uint256(0)) >> 3;
```

2.44 CVF-44 Redundant assignment

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

Description This assignment is redundant.
Recommendation Just use commitment & mask instead of commitment in the line below.

Listing 44: Redundant commitment = commitment & mask; assignment

```
532         commitment = commitment & mask;
```

2.45 CVF-45 The redundant `emitDepositCommitEvent` function

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

Description this function is more suitable for Operations.sol, where it can be updated timely if the data structure is modified. Also, this function is called in only one place and is trivial, so probably not worth extracting at all.
Client comment Number of variables in the function where this is used is too big so we need to separate this.

Listing 45: The redundant `emitDepositCommitEvent` function

```
597       function emitDepositCommitEvent(uint32 _blockNumber,
    ↪ Operations.Deposit memory depositData) internal {
598       emit DepositCommit(
599       _blockNumber,
600       depositData.accountId,
601       depositData.owner,
602       depositData.tokenId,
603       depositData.amount
604       );
605        }
606
607       function emitFullExitCommitEvent(uint32 _blockNumber,
    ↪ Operations.FullExit memory fullExitData) internal {
608       emit FullExitCommit(
609       _blockNumber,
610       fullExitData.accountId,
611       fullExitData.owner,
612       fullExitData.tokenId,
613       fullExitData.amount
614       );
615        }
```

2.46 CVF-46 Suboptimal totalBlocksProofed = totalBlocksCommitted; assign

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

Description The order of arguments is different from the original struct

Listing 46: Suboptimal `totalBlocksProofed = totalBlocksCommitted;` assign

```
598           emit DepositCommit(
608           emit FullExitCommit(
```

2.47 CVF-47 Redundan temitFullExitCommitEvent function

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

Description The `emitFullExitCommitEvent` function is called in only one place and is trivial, so probably not worth extracting at all.

Listing 47: Redundant `emitFullExitCommitEvent`function

```
607     function emitFullExitCommitEvent(uint32 _blockNumber,
    ↪ Operations.FullExit memory fullExitData) internal {
```

2.48 CVF-48 Inefficient uint32 counter using

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

Description Using `uint32` for counter is less efficient than using `uint256`, as Solidity will truncate the value to 32 bits after many operations.

Listing 48: Inefficient `uint32` counter using

```
639     for (uint32 i = 0; i < _newBlockData.onchainOperations.
    ↪ length; ++i) {
```

2.49 CVF-49 The redundant check

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

Description This check as well as division below would not be necessary in case pubdataOffset would be specified in chunks rather than in bytes.
Client comment Can be fixed in the next versions.

Listing 49: Redundant pubdataOffset % CHUNK_BYTES == 0, "fcso2" check

```
643     require(pubdataOffset % CHUNK_BYTES == 0, "fcso2"); //
    ↪ offsets should be on chunks boundaries
```

2.50 CVF-50 The redundant calculating

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

Description pubdataOffset / CHUNK_SIZE was already calculated in the previous line.
Recommendation Consider calculating once and caching in a local variable.

Listing 50: Redundant pubdataOffset / CHUNK_SIZE calculating

```
645         offsetsCommitment[pubdataOffset / CHUNK_BYTES] = bytes1(0
    ↪ x01);
```

2.51 CVF-51 Suboptimal bytes allocating

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

Description Allocating a new bytes array for a slice just for hashing is suboptimal.
Recommendation Consider hashing the slice in place.
Client comment We will skip due to very small relative profit.

Listing 51: Suboptimal bytes allocating

```
658         bytes memory opPubData = Bytes.slice(pubData,
    ↪ pubdataOffset, PARTIAL_EXIT_BYTES);
```

2.52 CVF-52 Line refactoring

- **Severity** Moderate
- **Status** FIXED
- **Category** Suboptimal
- **Source** ZkSync.sol

Description The 660 line repeats several times.
Recommendation Consider refactoring according to example.

Listing 52: Example

```
if (Deposit) {...}
else if (ChangePubKey) {...}
else {
    bytes memory opPubData;

if (PartialExit) {...}
else if (ForcedExir) {...}
else if (FullExit) {...}
else {revert}

processableOperationsHash =
    Utils.concatHash(
    processableOperationsHash,
    opPubData);
}
```

Listing 53: Line refactoring

```
660     processableOperationsHash = Utils.concatHash(
    ↪ processableOperationsHash, opPubData);
```

2.53 CVF-53 Over-complicated CREATE2 public key method

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

Description CREATE2 public key verification method looks over-complicated. If the idea is to bind a deployed smart contract with a public key, then the smart contract may just implement a public function that returns the hash of its public key, and zkSync may just call this function.

Listing 54: Over-complicated `CREATE2 public key` method

```
736     function verifyChangePubkeyCREATE2(bytes memory
    ↪ _ethWitness, Operations.ChangePubKey memory _changePk)
```

2.54 CVF-54 Unclear purpose 0 assignment

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

Description Why the nonce is always 0?
Client comment We allow use to set this `pubkey` only once so in case zkSync private key is lost this key can't be abused.

Listing 55: Unclear purpose 0 assignment

```
754     return recoveredAddress == _changePk.owner && _changePk.
    ↪ nonce == 0;
```

2.55 CVF-55 Expensive hashing

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

Description Hashing all at once would be cheaper. Is it really necessary to hash iterative?
Client comment Left over from previous versions.

Listing 56: Expensive hashing

```
764     bytes32 hash = sha256(abi.encodePacked(uint256(
    ↪ _newBlockData.blockNumber), uint256(_newBlockData.
    ↪ feeAccount)));
765     hash = sha256(abi.encodePacked(hash, _previousBlock.
    ↪ stateHash));
766     hash = sha256(abi.encodePacked(hash, _newBlockData.
    ↪ newStateHash));
767     hash = sha256(abi.encodePacked(hash, uint256(
    ↪ _newBlockData.timestamp)));
```

2.56 CVF-56 Overflow in the addPriorityRequest function

- **Severity** Minor
- **Status** OPENED
- **Category** Overflow
- **Source** ZkSync.sol

Description Overflow is possible in this line.
Client comment Overflow is impossible here.

Listing 57: Overflow in the `addPriorityRequest` function

```
832     uint64 expirationBlock = uint64(block.number +
    ↪ PRIORITY_EXPIRATION);
```

2.57 CVF-57 Unclear logging purpose

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

Description What is the purpose of logging the caller?

Listing 58: Unclear logging purpose

```
844         emit NewPriorityRequest(msg.sender,
    ↪ nextPriorityRequestId, _opType, _pubData, uint256(
    ↪ expirationBlock));
```

2.58 CVF-58 Dangerous function call

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Proxy.sol

Description This call may change target address, bypassing checks and logic coded in upgradeTarget function.
Recommendation While this isn't necessary an issue, this could be prevented by comparitng target values before and after delegate call and reverting in cae they are different.
Client comment Added the `nonReentrant` modifier to the `upgrade` function in the `ZkSync.sol`.

Listing 59: Dangerous function call

```
82        let result := delegatecall(
```

2.59 CVF-59 Redundant payable function

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Proxy.sol

Description This function is redundant, as the documentation for receive function states: "If no such function exists, but a payable fallback function exists, the fallback function will be called on a plain Ether transfer." So separate receive function does make sense in case its logic differs from the logic of fallback function, i.e. not in our case.

Listing 60: Redundant payable function

```
113     receive () external payable {
```

2.60 CVF-60 Draft code

- **Severity** Minor
- **Status** FIXED
- **Category** Suboptimal
- **Source** Verifier.sol

Description This shouldn't be in PROD code.

Listing 61: Draft code

```
26      if (DUMMY_VERIFIER) {
27          uint oldGasValue = gasleft();
28          uint tmp;
29          while (gasleft() + 500000 > oldGasValue) {
30                  tmp += 1;
31              }
32              return true;
33          }
```

2.61 CVF-61 Magic number 500000

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Verifier.sol

Description This should be a named constant

Listing 62: Magic number 500000

```
29      while (gasleft() + 500000 > oldGasValue) {
```

2.62 CVF-62 Redundant variable

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Verifier.sol

Description This variable is redundant

Listing 63: Redundant variable

```
35          uint256 commitment = _individual_vks_inputs[i];
```

2.63 CVF-63 Suboptimal assignment

- **Severity** Minor
- **Status** FIXED
- **Category** Suboptimal
- **Source** Verifier.sol

Description Form uint(-1) is more common that ũint(0). Also, the whole mask should be a compile-time constant. Also, the variable here is redundant as it has constant value. Also, this assignment should be made only once before the loop.

Listing 64: Suboptimal assignment `uint256 mask = (~uint256(0)) >> 3;`

```
36        uint256 mask = (~uint256(0)) >> 3;
```

2.64 CVF-64 Redundant cast

- **Severity** Minor
- **Status** FIXED
- **Category** Suboptimal
- **Source** Verifier.sol

Description The cast is redundant. Also, it potentially loses information.
Recommendation It would be better to just require, that the commitment is a field element.
Client comment Fixed in barichek-sc-changes-for-v4.

Listing 65: Redundant cast

```
37        _individual_vks_inputs[i] = uint256(commitment) & mask;
```

2.65 CVF-65 Suboptimal constant passing

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Verifier.sol

Description Passing constants to function calls is probably suboptimal

Listing 66: Suboptimal constant passing

```
40        return verify_serialized_proof_with_recursion(
    ↪ _recursiveInput, _proof, VK_TREE_ROOT, VK_MAX_INDEX,
    ↪ _vkIndexes, _individual_vks_inputs, _subproofs_limbs, vk);
```

2.66 CVF-66 Redundant PendingWithdrawal_DEPRECATED struct

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

Description As long as this structure is not used anymore, it should be removed.

Listing 67: Redundant PendingWithdrawal_DEPRECATED struct

```
42      struct PendingWithdrawal_DEPRECATED {
```

2.67 CVF-67 Suboptimal the totalBlocksVerified variable

- **Severity** Minor
- **Status** FIXED
- **Category** Unclear behavior
- **Source** Storage.sol

Description Is it more correct to be the number of executed blocks? There is now another variable for proven blocks, `totalBlocksProofed`

Listing 68: Suboptimal `totalBlocksVerified` variable

```
51      uint32 public totalBlocksVerified;
```

2.68 CVF-68 Confusing totalCommittedPriorityRequests variable name

- **Severity** Minor
- **Status** OPENED
- **Category** Documentation
- **Source** Storage.sol

Description The name is confusing as actually this is not the total number of committed priority requests, but rather the number of committed but not yet executed priority requests.

Listing 69: Confusing `totalCommittedPriorityRequests` variable name

```
117     uint64 public totalCommittedPriorityRequests;
```

2.69 CVF-69 Incorrect comment

- **Severity** Minor
- **Status** FIXED
- **Category** Documentation
- **Source** Storage.sol

Description Despite tech comment, this structure seems to be still used.

Listing 70: Incorrect comment

```
129     /// @Rollup block stored data - not used in current
    ↪ version
```

2.70 CVF-70 Non-existent StoredBlockInfo structure

- **Severity** Minor
- **Status** FIXED
- **Category** Unclear behavior
- **Source** Storage.sol

Description This function uses the `StoredBlockInfo` structure that is claimed to not be used in current version.
Recommendation Either this function should be removed, or the structure should be recognized as still used.

Listing 71: Non-existent `StoredBlockInfo` structure

```
146     function hashStoredBlockInfo(StoredBlockInfo memory
    ↪ _storedBlockInfo) internal pure returns (bytes32) {
```

2.71 CVF-71 The comment to non-existent StoredBlockInfo structure

- **Severity** Minor
- **Status** FIXED
- **Category** Documentation
- **Source** Storage.sol

Description The comment for this mapping refers to the `StoredBlockInfo` structure that is claimed to not be used in current version.
Recommendation Either don't refer to that structure or confirm the structure as still being used.

Listing 72: The comment to non-existent `StoredBlockInfo` structure

```
151     mapping(uint32 => bytes32) public storedBlockHashes;
```

2.72 CVF-72 Unclear comment

- **Severity** Minor
- **Status** OPENED
- **Category** Unclear behavior
- **Source** Storage.sol

Description What is "in one slot" means?
Client comment Deprecated part of the code.

Listing 73: Unclear comment

```
153     /// @notice Stores verified commitments hashed in one
    ↪ slot.
```

2.73 CVF-73 No check for the callSuccess

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Utils.sol

Description This line should be executed only when callSuccess is true.
Client comment We will skip due to very small relative profit.

Listing 74: No check for the `callSuccess`

```
31        bool returnedSuccess = callReturnValueEncoded.length ==
    ↪ 0 || abi.decode(callReturnValueEncoded, (bool));
```

2.74 CVF-74 No check for the callSuccess-2

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Utils.sol

Description This line should be executed only when callSuccess is true.
Client comment We will skip due to very small relative profit.

Listing 75: No check for the `callSuccess-2`

```
48      bool returnedSuccess = callReturnValueEncoded.length == 0
    ↪  || abi.decode(callReturnValueEncoded, (bool));
```

2.75 CVF-75 Redundant ETH_WITHDRAWAL_GAS_LIMIT variable

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Utils.sol

Description This variable is redundant. Assigning the hardcoded value to a local variable is just waste of gas.
Recommendation Either use named constant or hardcoded value.
Client comment Will be fixed in the next versions.

Listing 76: Redundant `ETH_WITHDRAWAL_GAS_LIMIT` variable

```
58        uint256 ETH_WITHDRAWAL_GAS_LIMIT = 10000;
```

2.76 CVF-76 No check for the returned value

- **Severity** Minor
- **Status** OPENED
- **Category**
- **Source** Utils.sol

Description In Solidity `ecrecover` returns zero address on invalid signature, rather then throws. This allows to `sign` anything with zero address.
Recommendation Always check ecrecover returned value and throw in case it is zero.

Listing 77: No check for the returned value

```
79        return ecrecover(_messageHash, signV, signR, signS);
```

2.77 CVF-77 The Config contract converting

- **Severity** Minor
- **Status** FIXED
- **Category** Suboptimal
- **Source** Config.sol

Description This contract could be turned into a library.

Listing 78: The `Config` contract converting

```
8         contract Config {
```

2.78 CVF-78 Small 50 000 limit

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Config.sol

Description The original limit was 250,000, 50,000 limit is quite optimistic. It would not be enough for complicated token contracts that charge transfer fees, pay dividends for holding tokens etc. See the following answer for some estimates: https://ethereum.stackexchange.com/a/72573 If you want default gas limit to be that low, there should be some way for a caller to override the limit to rescue tokens whose transfer doesn't fit into the default limit.
Client comment Decided to fix this in the next upgrades.

Listing 79: Small 50 000 limit

```
11        uint256 constant ERC20_WITHDRAWAL_GAS_LIMIT = 50000;
```

2.79 CVF-79 No access level for the ERC20_WITHDRAWA_GAS_LIMIT constant

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Config.sol

Description Also, no access level specified for this constant, so internal access will be used by default.
Recommendation Consider specifying access level explicitly.

Listing 80: No access level for the `ERC20_WITHDRAWAL_GAS_LIMIT` constant

```
11        uint256 constant ERC20_WITHDRAWAL_GAS_LIMIT = 50000;
    ↪
```

2.80 CVF-80 No access level for the MASS_FULL_EXIT_PERIOD

constant

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Config.sol

Description No access level specified for this constant, so internal access will be used by default.
Recommendation Consider specifying access level explicitly.

Listing 81: No access level for the `MASS_FULL_EXIT_PERIOD` constant

```
77        uint constant MASS_FULL_EXIT_PERIOD = 3 days;
```

2.81 CVF-81 No access level for the TIME_TO_WITHDRAW_FUNDS_FROM_FULL_EXIT constant

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Config.sol

Description No access level specified for this contant, so internal access will be used by default.
Recommendation Consider specifying access level explicitly.

Listing 82: No access level for the `TIME_TO_WITHDRAW_FUNDS_FROM_FULL_EXIT` constant

```
79        ////@dev Reserved time for users to withdraw funds from
    ↪ full exit priority operation in case of an upgrade (in
    ↪ seconds)
```

2.82 CVF-82 No access level for the UPGRADE_NOTICE_PERIOD constant

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Config.sol

Description No access level specified for this constant, so internal access will be used by default.
Recommendation Consider specifying access level explicitly.

Listing 83: No access level for the `UPGRADE_NOTICE_PERIOD` constant

```
84        uint constant UPGRADE_NOTICE_PERIOD =
    ↪ MASS_FULL_EXIT_PERIOD + PRIORITY_EXPIRATION_PERIOD +
    ↪ TIME_TO_WITHDRAW_FUNDS_FROM_FULL_EXIT;
```

2.83 CVF-83 No access level for the COMMIT_TIMESTAMP_NOT_OLDER constant

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Config.sol

Description No access level specified for this constant, so internal access will be used by default.
Recommendation Consider specifying access level explicitly.

Listing 84: No access level for the `COMMIT_TIMESTAMP_NOT_OLDER` constant

```
87        uint constant COMMIT_TIMESTAMP_NOT_OLDER = 8 hours;
```

2.84 CVF-84 No access level for the COMMIT_TIMESTAMP_APPROXIMATION_DELTA constant

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Config.sol

Description No access level specified for this constant, so internal access will be used by default.
Recommendation Consider specifying access level explicitly.

Listing 85: No access level for the `COMMIT_TIMESTAMP_APPROXIMATION_DELTA` constant

```
91        uint constant COMMIT_TIMESTAMP_APPROXIMATION_DELTA = 1
    ↪ minutes;
```

2.85 CVF-85 Redundant "Add" in the variable name

- **Severity** Minor
- **Status** OPENED
- **Category** Documentation
- **Source** Events.sol

Description This event is nowhere emitted. Also, word "Add" in the name is probably redundant. "PendingWithdrawals" would be enough.
Recommendation "PendingWithdrawals" would be enough.

Listing 86: Redundant `Add` in the variable name

```
79        event PendingWithdrawalsAdd(
```

2.86 CVF-86 Redundant queueStartIndex property

- **Severity** Minor
- **Status** OPENED
- **Category** Documentation
- **Source** Events.sol

Description This property is redundant as it could be derived from the previous event.

Listing 87: Redundant `queueStartIndex` property

```
80        uint32 queueStartIndex,
```

2.87 CVF-87 Unclear queueEndIndex index purpose

- **Severity** Minor
- **Status** OPENED
- **Category** Unclear behavior
- **Source** Events.sol

Description It is unclear whether this index is inclusive or not.

Listing 88: Unclear `queueEndIndex` index purpose

```
81        uint32 queueEndIndex
88        uint32 queueEndIndex
```

2.88 CVF-88 Not emitted PendingWithdrawalsComplete event

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

Description This event is nowhere emitted. Also it name incorrect.
Recommendation Consider event naming via nouns, such as just `Withdrawals`.

Listing 89: Not emitted `PendingWithdrawalsComplete` event

```
86        event PendingWithdrawalsComplete(
```

2.89 CVF-89 Redundant queueStartIndex property

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

Description This property is redundant as it could be derived from the previous event.

Listing 90: Redundant `queueStartIndex` property

```
87        uint32 queueStartIndex,
```

2.90 CVF-90 Unseparated TokenPausedUpdate

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

Description It would be more gas efficient to have two separate events: `TokenPause` and `TokenUnpause`.
Client comment We will skip due to very small relative profit.

Listing 91: Unseparated TokenPausedUpdate event

```
29        event TokenPausedUpdate(
```

2.91 CVF-91 Unread mapping

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

Description This mapping is never read in the smart contract.

Listing 92: Unread `address => bool` mapping

```
49        /// @notice Paused tokens list, deposits are impossible
    ↪ to create for paused tokens
```

2.92 CVF-92 Unclear identification

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

Description Also, tokens are identified here by their IDs, while in other places within this smart contract they are identified by their addresses.
Recommendation Probably address could be used here as well. Also, it would be more gas efficient to store "paused" flags in "tokenIds" mapping in some bit above the lowest 16 bits, currently used for token IDs.
Client comment We will skip due to very small relative profit.

Listing 93: Unclear indentification

```
50        mapping(uint16 => bool) public pausedTokens;
```

2.93 CVF-93 Redundant "52 constructor()" line

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

Description This line is redundant.

Listing 94: Redundant "52 constructor() {}" line

```
52        constructor() {}
```

2.94 CVF-94 The "because it" typo

- **Severity** Minor
- **Status** OPENED
- **Category** Documentation
- **Source** Operations.sol

Description Should be "because they are".
Client comment Deprecated part of code.

Listing 95: The `because it` typo

```
92        // We must ignore `accountId` and operation type because
    ↪ it is present in block pubdata but not in priority queue
```

2.95 CVF-95 Increased gas consumption

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Operations.sol

Description The expression at the right is actually constant, so caching it in variable probably increases gas consumption, rather than reduces it.

Listing 96: Increased gas consumption

```
93        uint skipBytes = ACCOUNT_ID_BYTES + OP_TYPE_BYTES;
```

2.96 CVF-96 Suboptimal slice copying

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Operations.sol

Description Copying a slice to a newly allocated bytes array just to hash it is suboptimal.
Recommendation Consider hashing bytes range in place. This will require some assembly, but copying a slice requires some assembly as well. Also, there is extra space after the first occurrence of "skipBytes" that ought to be removed.

Listing 97: Suboptimal slice copying

```
94        bytes memory lhs_trimmed = Bytes.slice(_lhs, skipBytes,
    ↪ PACKED_DEPOSIT_PUBDATA_BYTES - skipBytes);
```

2.97 CVF-97 Suboptimal slice copying

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Operations.sol

Description Copying a slice to a newly allocated bytes array just to hash is suboptimal.
Recommendation Consider hashing bytes range in place. This will require some assembly, but copying a slice requires some assembly as well.

Listing 98: Suboptimal slice copying

```
95        bytes memory rhs_trimmed = Bytes.slice(_rhs, skipBytes,
    ↪ PACKED_DEPOSIT_PUBDATA_BYTES - skipBytes);
```

2.98 CVF-98 Ignored operation type

- **Severity** Minor
- **Status** OPENED
- **Category** Suboptimal
- **Source** Operations.sol

Description Operation type is not ignored here. Probably not an issue.
Client comment Deprecated part of code.

Listing 99: Ignored operation type

```
136     // `amount` is ignored because it is present in block
    ↪ pubdata but not in priority queue
```

2.99 CVF-99 Unclear opType field behavior

- **Severity** Minor
- **Status** FIXED
- **Category** Documentation
- **Source** Operations.sol

Description This field seems to also be included in pubdata but ignored at serialization.
Recommendation Consider adding a comment.

Listing 100: Unclear opType field behavior

```
145         //uint8 opType
```

2.100 CVF-100 Confusing ChangePubkeyType name

- **Severity** Minor
- **Status** OPENED
- **Category** Documentation
- **Source** Operations.sol

Description The name is confusing, as it is unclear what "type" belongs to.
Recommendation Consider renaming to just "PubkeyType" if it belongs to public key, or to "PubkeyChangeType" if it belongs to change pubkey operation in general.

Listing 101: Confusing `ChangePubkeyType` name

```
188     enum ChangePubkeyType {
```

2.101 CVF-101 Unusual Create2 name

- **Severity** Minor
- **Status** OPENED
- **Category** Documentation
- **Source** Operations.sol

Description The corresponding opcode name is usually typed in upper case as "CREATE2".
Recommendation Consider using upper case here as well.

Listing 102: Unusual `Create2` name

```
190     Create2
```