ABDK CONSULTING

SMART CONTRACT AUDIT

ZKSWAP AUDIT

PART 1: SOLIDITY

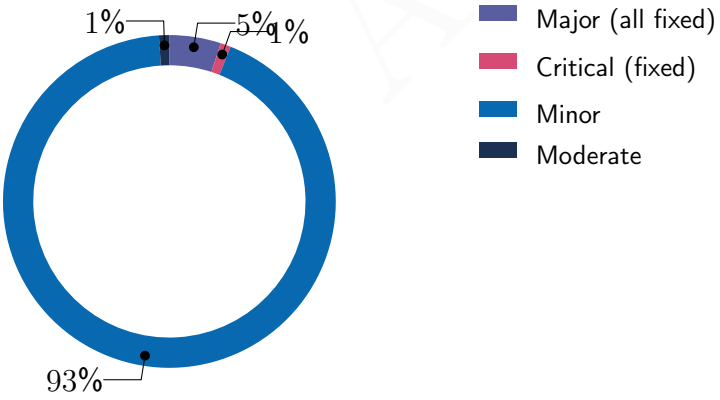
abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
02/23/21

We've been asked to review the smart contracts that form the core of the ZkSwap protocol. We have found a few major issues that affect the security, and all of them were fixed. We have also identified a number of other issues which do not directly affect the security but are important for scalability, readability, or efficiency. Those were left for future releases. Explanations, when necessary, were given by the code developers and are provided in this report as is.

We conclude that to the best of our knowledge, there is no safety or functionality issues left unresolved. The code examined respects the given specifications with some minor exceptions that do not affect the overall sentiment but would be essential for the code's maintainability and readability.



Findings

ID	Severity	Subject	Status
CVF-1	Minor	Unreviewed KeysWithPlonkVerifier contract	Opened
CVF-2	Minor	Unreviewed Verifier contract	Opened
CVF-3	Minor	No access level	Opened
CVF-4	Minor	Dummy functionality	Opened
CVF-5	Minor	Unreviewed isBlockSizeSupportedInternal contract	Opened
CVF-6	Minor	Readability issue	Opened
CVF-7	Minor	Uncommon uint256(-1) assignment	Opened
CVF-8	Minor	Improper type	Opened
CVF-9	Minor	Unreviewed function	Opened
CVF-10	Minor	Unreviewed function	Opened
CVF-11	Minor	Expensive assignment	Opened
CVF-12	Minor	Unreviewed function	Opened
CVF-13	Minor	Unreviewed contract	Opened
CVF-14	Minor	Unreviewed contract	Opened
CVF-15	Minor	Uncommon assignment	Opened
CVF-16	Minor	Improper type	Opened
CVF-17	Minor	Unreviewed function	Opened
CVF-18	Minor	Unreviewed function	Opened
CVF-19	Minor	Expensive assignment	Opened
CVF-20	Minor	Unreviewed function	Opened
CVF-21	Minor	Incorrect access modifier	Opened
CVF-22	Minor	Inefficient function	Opened
CVF-23	Minor	Inefficient concatenation	Opened
CVF-24	Minor	Unreviewed function	Opened
CVF-25	Minor	Out of scope Upgradeable.sol file	Opened
CVF-26	Minor	Unmentioned author of the changes	Opened

ID	Severity	Subject	Status
CVF-27	Minor	Unclear difference	Opened
CVF-28	Minor	Incorrect OnchainCreatePair event naming	Opened
CVF-29	Minor	Unspecific type parameter	Opened
CVF-30	Minor	Not indexed nonce parameter	Opened
CVF-31	Minor	Redundant word "mode"	Opened
CVF-32	Minor	Redundant word "New"	Opened
CVF-33	Minor	Not indexed parameters	Opened
CVF-34	Minor	Redundant word "Add"	Opened
CVF-35	Minor	Redundant derive of the queueStartIndex value	Opened
CVF-36	Minor	Confusing event name	Opened
CVF-37	Minor	Redundant derive of the value-2	Opened
CVF-38	Minor	Suboptimal UpgradeEvents interface placement	Opened
CVF-39	Minor	Intermixing uint256 type	Opened
CVF-40	Minor	Unreviewed file	Opened
CVF-41	Minor	Unreviewed file	Opened
CVF-42	Minor	Incorrect refer	Opened
CVF-43	Minor	Unmentioned author	Opened
CVF-44	Minor	Redundant flag	Opened
CVF-45	Minor	Suboptimal gas usage	Opened
CVF-46	Minor	Suboptimal gas usage	Opened
CVF-47	Minor	Incorrect description	Opened
CVF-48	Minor	Undocumented withdrawalsDataHash field	Opened
CVF-49	Minor	Inefficient mapping	Opened
CVF-50	Minor	Inefficient mapping	Opened
CVF-51	Minor	Inefficient address packing	Opened
CVF-52	Minor	Intermixed type	Opened
CVF-53	Minor	Suboptimal gas usage	Opened

ID	Severity	Subject	Status
CVF-54	Minor	Incorrect zkSyncCommitBlockAddress storage type	Opened
CVF-55	Minor	Incorrect zkSyncExitAddress storage type	Opened
CVF-56	Minor	Incorrect line refer	Opened
CVF-57	Minor	Unmentioned author	Opened
CVF-58	Minor	Improper declaration	Opened
CVF-59	Minor	Missing access level	Opened
CVF-60	Minor	Improper type	Opened
CVF-61	Minor	Improper naming	Opened
CVF-62	Minor	Suboptimal uint16 type	Opened
CVF-63	Minor	Suboptimal number form	Opened
CVF-64	Minor	Improper type	Opened
CVF-65	Minor	Timing	Opened
CVF-66	Minor	Unclear assignment	Opened
CVF-67	Minor	Unclear behavior	Opened
CVF-68	Minor	Readability issue	Opened
CVF-69	Moderate	Suspicious rounding	Ruled out
CVF-70	Minor	Unclear estimate	Opened
CVF-71	Minor	Incorrect uint64 type using	Opened
CVF-72	Minor	Intermixing uint256 type	Opened
CVF-73	Minor	Inconsistent compiler version	Opened
CVF-74	Minor	Out of scope files	Opened
CVF-75	Minor	Suboptimal import	Opened
CVF-76	Minor	Unseparated contracts	Opened
CVF-77	Minor	Interviewed file	Opened
CVF-78	Minor	Interviewed contract	Opened
CVF-79	Minor	Suboptimal formatting	Opened
CVF-80	Minor	Missing event	Opened
CVF-81	Minor	Redundant checks	Opened

ID	Severity	Subject	Status
CVF-82	Minor	Unclear split purpose	Opened
CVF-83	Minor	Improper type	Opened
CVF-84	Minor	Improper type	Opened
CVF-85	Minor	Redundant casts	Opened
CVF-86	Minor	Improper type	Opened
CVF-87	Minor	Unreviewed function	Opened
CVF-88	Minor	Unmentioned author	Opened
CVF-89	Minor	Improper type	Opened
CVF-90	Minor	Not separated ValidatorStatusUpdate event	Opened
CVF-91	Minor	Redundant word	Opened
CVF-92	Minor	Not indexed parameter	Opened
CVF-93	Minor	Improper type	Opened
CVF-94	Minor	Inefficient mapping	Opened
CVF-95	Minor	Specific type missing	Opened
CVF-96	Minor	Redundant line	Opened
CVF-97	Minor	Not checked parameter line	Opened
CVF-98	Minor	Redundant variable	Opened
CVF-99	Minor	Out of scope contract	Opened
CVF-100	Moderate	Missed check	Ruled out
CVF-101	Minor	Unclear constant value	Opened
CVF-102	Minor	Redundant code	Opened
CVF-103	Minor	Redundant word "new"	Opened
CVF-104	Minor	Improper modifier	Opened
CVF-105	Minor	Redundant check	Opened
CVF-106	Minor	Redundant function	Opened
CVF-107	Minor	Out of scope files	Opened
CVF-108	Minor	Unreviewed ReentrancyGuard contact	Opened
CVF-109	Minor	Doubly evaluated expression	Opened

ID	Severity	Subject	Status
CVF-110	Critical	Incorrect updateBalance access modify	Fixed
CVF-111	Minor	Confusing function name	Opened
CVF-112	Major	Improper modifier	Fixed
CVF-113	Moderate	Overflow in the line	Opened
CVF-114	Minor	Redundant toSafeCast.toUint12 call	Opened
CVF-115	Moderate	Length check missing	Ruled out
CVF-116	Minor	Incorrect check placement	Opened
CVF-117	Minor	Redundant assignments	Opened
CVF-118	Moderate	Length check missing	Ruled out
CVF-119	Minor	Predefined arrays lengths	Opened
CVF-120	Minor	Redundant _rootHash argument	Opened
CVF-121	Minor	Redundant _addresses[0] argument	Opened
CVF-122	Minor	Suboptimal check placement	Opened
CVF-123	Minor	Suboptimal array elements placement	Opened
CVF-124	Minor	Unreviewed file	Opened
CVF-125	Minor	Incorrect line zkSync refer-2	Opened
CVF-126	Minor	Not used _CloseAccount contract	Opened
CVF-127	Minor	Unclear comment	Opened
CVF-128	Minor	Constant missing	Opened
CVF-129	Minor	Suboptimal using of the uint8 type	Opened
CVF-130	Minor	Confusing TOKEN_BYTES name	Opened
CVF-131	Minor	Same constant	Opened
CVF-132	Minor	Similar constants	Opened
CVF-133	Minor	Improper name	Opened
CVF-134	Minor	Unused constant	Opened
CVF-135	Minor	Improper modifier	Opened
CVF-136	Moderate	Length check missing	Ruled out
CVF-137	Minor	Inefficient code	Opened

ID	Severity	Subject	Status
CVF-138	Minor	Improper modifier	Opened
CVF-139	Minor	Improper type	Opened
CVF-140	Minor	Suboptimal slice copying	Opened
CVF-141	Minor	Expensive comparison	Opened
CVF-142	Minor	Same structure fields	Opened
CVF-143	Minor	Structure slots	Opened
CVF-144	Minor	Similar function	Opened
CVF-145	Minor	Out of scope files	Opened
CVF-146	Minor	Improper reference	Opened
CVF-147	Minor	Unmentioned author	Opened
CVF-148	Minor	Unreviewed UpgradeableMaster and ReentrancyGuard contracts	Opened
CVF-149	Minor	Incorrect ZkSync contract name	Opened
CVF-150	Minor	Improper definition	Opened
CVF-151	Minor	Not used contract	Opened
CVF-152	Minor	Incorrect createPair behavior	Opened
CVF-153	Moderate	Incorrect maximum number of tokens	Ruled out
CVF-154	Minor	Reordered tokens	Opened
CVF-155	Minor	Redundant checks	Opened
CVF-156	Minor	Unreviewed createPair function	Opened
CVF-157	Minor	Unnecessary call	Opened
CVF-158	Minor	Function similar to createPair function	Opened
CVF-159	Moderate	Not reordered tokens	Ruled out
CVF-160	Minor	Improper type	Opened
CVF-161	Minor	Outdated function name	Opened
CVF-162	Minor	Redundant code line	Opened
CVF-163	Minor	Incorrect variable type	Opened
CVF-164	Minor	Incorrect variable type-2	Opened
CVF-165	Minor	Incorrect variable type-3	Opened

ID	Severity	Subject	Status
CVF-166	Minor	Incorrect variable type-4	Opened
CVF-167	Minor	Improper call	Opened
CVF-168	Critical	Vulnerable setGenesisRootAndAddresses function	Fixed
CVF-169	Minor	Redundant validatePairTokenAddress call	Opened
CVF-170	Minor	Out of scope mint function	Opened
CVF-171	Minor	Out of scope sendERC20 function	Opened
CVF-172	Minor	Out of scope minU32 function	Opened
CVF-173	Minor	Multiple times value calculation	Opened
CVF-174	Minor	Already calculated value	Opened
CVF-175	Minor	Already calculated value	Opened
CVF-176	Minor	Expensive assignment	Opened
CVF-177	Minor	Improper cast	Opened
CVF-178	Minor	Out of scope sendETHNoRevert function	Opened
CVF-179	Minor	Redundant assignment	Opened
CVF-180	Minor	Redundant registry of tokens	Opened
CVF-181	Minor	Suboptimal condition usage	Opened
CVF-182	Minor	Suboptimal code pattern	Opened
CVF-183	Moderate	Overflow	Ruled out
CVF-184	Minor	Out of scope minU64 function	Opened
CVF-185	Minor	Multiple times function call	Opened
CVF-186	Minor	Already calculated value	Opened
CVF-187	Moderate	Overflow	Fixed
CVF-188	Minor	Incorrect depositETH function behavior	Opened
CVF-189	Minor	Incorrect withdrawETH function behavior	Opened
CVF-190	Minor	Non obvious check	Opened
CVF-191	Minor	Complicated code	Opened

ID	Severity	Subject	Status
CVF-192	Minor	Complicated code	Opened
CVF-193	Minor	Suboptimal code placement	Opened
CVF-194	Major	Missing check	Ruled out
CVF-195	Minor	Redundant assignment	Opened
CVF-196	Minor	Out of scope transferFromERC20 function	Opened
CVF-197	Minor	Not supported LP tokens	Opened
CVF-198	Minor	Redundant check	Opened
CVF-199	Minor	Redundant gas usage	Opened
CVF-200	Minor	Check missing	Opened
CVF-201	Minor	Out of scope sub function	Opened
CVF-202	Moderate	Overflow possible	Ruled out
CVF-203	Moderate	Overflow possible	Ruled out
CVF-204	Minor	Incorrect comment	Opened
CVF-205	Minor	Assembly not needed	Opened
CVF-206	Minor	Out of scope files	Opened
CVF-207	Minor	Out of scope ReentrancyGuard conctract	Opened
CVF-208	Minor	Same SafeMath function	Opened
CVF-209	Minor	Improper usage of EMPTY_STRING_KECCAK	Opened
CVF-210	Minor	Improper modifier	Opened
CVF-211	Minor	Suboptimal totalBlocksCommitted incrementing placement	Opened
CVF-212	Minor	Redundant _publicData variable	Opened
CVF-213	Minor	Incorrect collectOnchainOps and createCommittedBlock functions signature	Opened
CVF-214	Minor	Expensive operation	Opened

ID	Severity	Subject	Status
CVF-215	Major	Check missing	Ruled out
CVF-216	Minor	Suboptimal totalBlocksVerified placement	Opened
CVF-217	Minor	Suboptimal Iterating block numbers	Opened
CVF-218	Minor	Redundant check	Opened
CVF-219	Minor	Expensive revert	Opened
CVF-220	Minor	Wrong comment	Opened
CVF-221	Minor	Wrong comment	Opened
CVF-222	Moderate	Missed initialization	Ruled out
CVF-223	Minor	Already calculated expression	Opened
CVF-224	Minor	Over-complicated triggerExodusIfNeeded function	Opened
CVF-225	Minor	Redundant hashing	Opened
CVF-226	Minor	Incorrect check placement	Opened
CVF-227	Moderate	Check missing	Ruled out
CVF-228	Minor	Field names missing	Opened
CVF-229	Minor	Trivial function	Opened
CVF-230	Minor	Incorrect function behavior	Opened
CVF-231	Minor	Variables read several times	Opened
CVF-232	Minor	Redundant usage of memory pointers	Opened
CVF-233	Minor	Complicated code	Opened
CVF-234	Minor	Complicated code	Opened
CVF-235	Minor	Redundant addToPendingWithdrawalsQueue variable	Opened
CVF-236	Minor	Redundant addToPendingWithdrawalsQueue variable	Opened
CVF-237	Minor	Suboptimal copy	Opened

ID	Severity	Subject	Status
CVF-238	Minor	Suboptimal pack of several string literals	Opened
CVF-239	Moderate	Dangerous behavior of the function	Ruled out
CVF-240	Minor	Unnecessary uint256 cast	Opened
CVF-241	Minor	Unclear comment	Opened
CVF-242	Minor	Calculated twice expression	Opened
CVF-243	Minor	Double reading of storage variable	Opened
CVF-244	Minor	Several times reading of storage variable	Opened
CVF-245	Minor	Redundant check	Opened
CVF-246	Minor	Calculated expression twice	Opened
CVF-247	Major	Redundant requests	Ruled out
CVF-248	Minor	Suboptimal contacts splitting	Opened
CVF-249	Minor	Unnecessary assembly	Opened

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	Jan. 18, 2021	D. Khovratovich	Initial Draft
0.2	Jan. 19, 2021	D. Khovratovich	Minor Revision
1.0	Jan. 20, 2021	D. Khovratovich	Release
1.1	Jan. 18, 2021	D. Khovratovich	Initial Draft
2.0	Jan. 19, 2021	D. Khovratovich	New Release
2.1	Feb. 22, 2021	D. Khovratovich	Add client comments
3.0	Feb. 23, 2021	D. Khovratovich	New Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the ZkSwap request. This is an intermediate report devoted to the Solidity files. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

We were given access to the ZkSwap repo, release v0.9. This part of the report is devoted to the smart contracts that run the ZkSwap protocol and verify ZkSwap transactions in zero knowledge. These files are written in Solidity language and are located here. This part provides findings in the following files:

- Config.sol;
- DeployFactory.sol;
- Events.sol;
- Governance.sol;
- Operations.sol;
- PairTokenManager.sol;
- Storage.sol;
- Verifier.sol;
- VerifierExit.sol;
- ZkSync.sol;
- ZkSyncCommitBlock.sol;
- ZkSyncExit.sol

The audit goal was:

- to review the constraints verified by the contracts and to check whether they are both necessary and sufficient for the protocol to operate according to the protocol description;
- to check the code for suboptimal data structures and functions;
- to check whether the documentation and code comments match the code;
- to find scalability issues that could complicate the further development of the project.
- to detect bad code practices;
- to issue recommendations when necessary.

2.1 About ABDK

ABDK Consulting, established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function. The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

3 Detailed Results

3.1 CVF-1 Unreviewed KeysWithPlonkVerifier contract

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Verifier.sol

Description We didn't review this file.

Listing 1: Unreviewed KeysWithPlonkVerifier contract

```
3 import "./KeysWithPlonkVerifier.sol";
```

3.2 CVF-2 Unreviewed Verifier contract

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Verifier.sol

Description The Verifier contract inherits a contract that we didn't review, thus the behavior of Verifier contract cannot be fully checked.

Listing 2: Unreviewed Verifier contract

```
6 Verifier is KeysWithPlonkVerifier {
```

3.3 CVF-3 No access level

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

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

Listing 3: No access level

```
8 bool constant DUMMY_VERIFIER = false;
```

3.4 CVF-4 Dummy functionality

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

Description The contract actually implements two different behaviors: dummy and real verifier.
Recommendation Proper way to do this is to extract IVerifier interface and have two implementations of it: RealVerifier (or just Verifier) and DummyVerifier. This will not pollute production code with dummy functionality like this.

Listing 4: Dummy functionality

```
18 if (DUMMY_VERIFIER) {
       return true;

30 if (DUMMY_VERIFIER) {
       uint oldGasValue = gasleft();
       uint tmp;
       while (gasleft() + 470000 > oldGasValue) {
           tmp += 1;
       }
       return true;
```

3.5 CVF-5 Unreviewed isBlockSizeSupportedInternal contract

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Verifier.sol

Description We didn't review isBlockSizeSupportedInternal function.

Listing 5: Unreviewed isBlockSizeSupportedInternal contract

```
21 return isBlockSizeSupportedInternal(_size);
```

3.6 CVF-6 Readability issue

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

Description This code looks like it is always executed, while it is executed only when DUMMY_VERIFIER is false.
Recommendation Consider putting this code into explicit "else" branch.

Listing 6: Readability issue

```
38 uint256[] memory inputs = new uint256[](1);
   uint256 mask = (~uint256(0)) >> 3;
40 inputs[0] = uint256(_commitment) & mask;
   Proof memory proof = deserialize_proof(inputs, _proof);
   VerificationKey memory vk = getVkBlock(_chunks);
   require(vk.num_inputs == inputs.length);
   return verify(proof, vk);
```

3.7 CVF-7 Uncommon uint256(-1) assignment

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

Description uint256(-1) is more common than ũint256(0).

Listing 7: Uncommon uint256(-1) assignment

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

3.8 CVF-8 Improper type

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

Recommendation Mask should be a named compile-time constant rather than a variable.

Listing 8: Improper type

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

3.9 CVF-9 Unreviewed function

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Verifier.sol

Description We didn't review deserialize_proof function.

Listing 9: Unreviewed function

```
41 Proof memory proof = deserialize_proof(inputs, _proof);
```

3.10 CVF-10 Unreviewed function

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Verifier.sol

Description We didn't review getVKBlock function.

Listing 10: Unreviewed function

```
42 VerificationKey memory vk = getVkBlock(_chunks);
```

3.11 CVF-11 Expensive assignment

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

Recommendation If the goal is to fit the commitment into 253 bits, then the cheapest way is: _commitment » 3, however this will leave higher, rather than lower, 253 bits.

Listing 11: Expensive assignment

```
39 uint256 mask = (~uint256(0)) >> 3;
40 inputs[0] = uint256(_commitment) & mask;
```

3.12 CVF-12 Unreviewed function

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Verifier.sol

Description We didn't review "verify" function.

Listing 12: Unreviewed function

```
44 return verify(proof, vk);
```

3.13 CVF-13 Unreviewed contract

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** VerifierExit.sol

Description We didn't review this file.

Listing 13: Unreviewed contract

```
3 "./KeysWithPlonkVerifier.sol";
```

3.14 CVF-14 Unreviewed contract

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** VerifierExit.sol

Description The VerifierExit contract inherits a contract that we didn't review, thus the behavior of Verifier contract cannot be fully checked.

Listing 14: Unreviewed contract

```
6 VerifierExit is KeysWithPlonkVerifier {
```

3.15 CVF-15 Uncommon assignment

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

Description uint256(-1) is more common than ũint256(0).

Listing 15: Uncommon assignment

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

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

3.16 CVF-16 Improper type

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

Recommendation Mask should be a named compile-time constant rather than a variable.

Listing 16: Improper type

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

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

3.17 CVF-17 Unreviewed function

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** VerifierExit.sol

Description We didn't review deserialize_proof function.

Listing 17: Unreviewed function

```
24 Proof memory proof = deserialize_proof(inputs, _proof);

59 Proof memory proof = deserialize_proof(inputs, _proof);
```

3.18 CVF-18 Unreviewed function

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** VerifierExit.sol

Description We didn't review getVKBlock function.

Listing 18: Unreviewed function

```
25 VerificationKey memory vk = getVkExit();
```

3.19 CVF-19 Expensive assignment

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

Recommendation If the goal is to fit the commitment into 253 bits, then the cheapest way is: _commitment » 3, however this will leave higher, rather than lower, 253 bits.

Listing 19: Expensive assignment

```
22 uint256 mask = (~uint256(0)) >> 3;
   inputs[0] = uint256(commitment) & mask;

57 uint256 mask = (~uint256(0)) >> 3;
   inputs[0] = uint256(commitment) & mask;
```

3.20 CVF-20 Unreviewed function

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** VerifierExit.sol

Description We didn't review "verify" function.

Listing 20: Unreviewed function

```
27 return verify(proof, vk);

62 return verify(proof, vk);
```

3.21 CVF-21 Incorrect access modifier

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

Description This utility function doesn't need to be public.

Listing 21: Incorrect access modifier

```
30 function concatBytes(bytes memory param1, bytes memory param2)
     ↪ public pure returns (bytes memory) {
```

3.22 CVF-22 Inefficient function

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

Description This function is deadly inefficient and is equivalent to abi.encodePacked(param1, param2).

Listing 22: Inefficient function

```
30 function concatBytes(bytes memory param1, bytes memory param2)
     ↪ public pure returns (bytes memory) {
       bytes memory merged = new bytes(param1.length + param2.
         ↪ length);

       uint k = 0;
       for (uint i = 0; i < param1.length; i++) {
           merged[k] = param1[i];
           k++;
       }

       for (uint i = 0; i < param2.length; i++) {
40         merged[k] = param2[i];
           k++;
       }
       return merged;
   }
```

3.23 CVF-23 Inefficient concatenation

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

Description Concatenating a sequence to a concatenation of two other sequences is suboptimal.
Recommendation More efficient solution would be to concatenate three sequences at once like this: abi.encodePacked(_account_data, _pair_data0, _pair_data1)

Listing 23: Inefficient concatenation

```
52 bytes memory _data1 = concatBytes(_account_data, _pair_data0);
   bytes memory _data2 = concatBytes(_data1, _pair_data1);
```

3.24 CVF-24 Unreviewed function

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** VerifierExit.sol

Description We didn't review getVkLpExit function.

Listing 24: Unreviewed function

```
60 VerificationKey memory vk = getVkLpExit();
```

3.25 CVF-25 Out of scope Upgradeable.sol file

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

Description This file is out of scope, so we will not review it.

Listing 25: Out of scope Upgradeable.sol file

```
3 import "./Upgradeable.sol";
```

3.26 CVF-26 Unmentioned author of the changes

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

Description Should the author of the changes be also mentioned here?

Listing 26: Unmentioned author of the changes

```
8 @author Matter Labs
```

3.27 CVF-27 Unclear difference

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

Description The difference between the sender and the owner is unclear.
Recommendation Consider adding more detail into the documentation comment above.

Listing 27: Unclear difference

```
26 address indexed sender,

29 address indexed owner
```

3.28 CVF-28 Incorrect OnchainCreatePair event naming

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

Recommendation Usually, events are named via nouns, such as OnchainPairCreation.

Listing 28: Incorrect OnchainCreatePair event naming

```
32 event OnchainCreatePair(
```

3.29 CVF-29 Unspecific type parameter

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

Description The type of the pair parameter could be made more specific.

Listing 29: Unspecific type parameter

```
36 address pair

102 address pair
```

3.30 CVF-30 Not indexed nonce parameter

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

Description This parameter should probably be indexed.

Listing 30: Not indexed nonce parameter

```
42 uint32 nonce,
```

3.31 CVF-31 Redundant word "mode"

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

Description The name is confusing. Probably, word “mode” is redundant.
Recommendation Just “Exodus” would be enough.

Listing 31: Redundant word "mode"

```
53 event ExodusMode();
```

3.32 CVF-32 Redundant word "New"

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

Description Word "New{ in the event name is redundant.
Recommendation Just "PriorityRequest" would be enough.

Listing 32: Redundant word "New"

```
56 event NewPriorityRequest(
```

3.33 CVF-33 Not indexed parameters

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

Description These parameters should be indexed.

Listing 33: Not indexed parameters

```
57 address sender,
uint64 serialId,
Operations.OpType opType,
```

3.34 CVF-34 Redundant word "Add"

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

Description Word "Add" is redundant in the event name.
Recommendation Just "PendingWithdrawals" would be enough.

Listing 34: Redundant word "Add"

```
84 event PendingWithdrawalsAdd(
```

3.35 CVF-35 Redundant derive of the queueStartIndex value

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

Description As the queueStartIndex value could be derived from the previous events.
Recommendation It would be enough to just log the number of pending withdrawals added.

Listing 35: Redundant derive of the queueStartIndex value

```
85 uint32 queueStartIndex,
   uint32 queueEndIndex
```

3.36 CVF-36 Confusing event name

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

Description Having both "pending" and "complete" is the event name is confusing.
Recommendation Just "CompleteWithdrawals" would be better.

Listing 36: Confusing event name

```
91 event PendingWithdrawalsComplete(
```

3.37 CVF-37 Redundant derive of the value-2

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

Description The queueStartIndex value could be derived from the previous events.
Recommendation It would be enough to just log the number of completed withdrawal requests.

Listing 37: Redundant derive of the value-2

```
92 uint32 queueStartIndex,
   uint32 queueEndIndex
```

3.38 CVF-38 Suboptimal UpgradeEvents interface placement

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

Recommendation This interface should be moved to a separate file named "UpgradeEvents.sol".

Listing 38: Suboptimal UpgradeEvents interface placement

```
108 interface UpgradeEvents {
```

3.39 CVF-39 Intermixing uint256 type

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

Description Intermixing type uint256 with its alias uint makes code harder to read.
Recommendation Consider using consistent type names across the code.

Listing 39: Intermixing uint256 type

```
112 uint indexed versionId,

118 uint indexed versionId,

120 uint noticePeriod // notice period (in seconds)

125 uint indexed versionId

130 uint indexed versionId

135 uint indexed versionId,
```

3.40 CVF-40 Unreviewed file

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

Description We didn't review this file.

Listing 40: Unreviewed file

```
3 import "./IERC20.sol";
```

3.41 CVF-41 Unreviewed file

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

Description This file is our of scope, so we didn't review it.

Listing 41: Unreviewed file

```
10 import "./uniswap/UniswapV2Factory.sol";
```

3.42 CVF-42 Incorrect refer

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

Description This line should refer to zkSwap, rather than zkSync.
Recommendation Probably, some credit to zkSync should also be here.

Listing 42: Incorrect refer

```
12 @title zkSync storage contract
```

3.43 CVF-43 Unmentioned author

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

Description Should the author of the changes be also mentioned here?

Listing 43: Unmentioned author

```
13 @author Matter Labs
```

3.44 CVF-44 Redundant flag

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

Description This flag is redundant, as its value could be derived from the value of upgradePreparationActivationTime storage variable.

Listing 44: Redundant flag

```
22 uint public upgradePreparationActivationTime;
```

3.45 CVF-45 Suboptimal gas usage

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

Description Having a separate field for gas reserve is suboptimal, as the compiler will try to preserve the value of this field when other fields are modified.
Recommendation Consider using high bits of balanceToWithdraw as gas reserve.

Listing 45: Suboptimal gas usage

```
35 uint8 gasReserveValue; // gives user opportunity to fill storage
   ↪ slot with nonzero value
```

3.46 CVF-46 Suboptimal gas usage

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

Recommendation It would be more gas efficient to replace this mapping with a dynamic array. This also would make numberOfPendingWithdrawals storage variable redundant.

Listing 46: Suboptimal gas usage

```
48 mapping(uint32 => PendingWithdrawal) public pendingWithdrawals;
```

3.47 CVF-47 Incorrect description

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

Description There are no such fields in the structure.

Listing 47: Incorrect description

```
59 /// @member validator Block producer

61 /// @member cumulativeOnchainOperations Total number of
   ↪ operations in this and all previous blocks
```

3.48 CVF-48 Undocumented withdrawalsDataHash field

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

Description These fields are not documented.

Listing 48: Undocumented withdrawalsDataHash field

```
70 uint32 chunks;
   bytes32 withdrawalsDataHash; /// can be restricted to 16 bytes
      ↪ to reduce number of required storage slots
```

3.49 CVF-49 Inefficient mapping

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

Recommendation It would be more gas efficient to replace this mapping with a dynamic array. This would also make totalBlocksCommitted storage variable redundant.

Listing 49: Inefficient mapping

```
77 mapping(uint32 => Block) public blocks;
```

3.50 CVF-50 Inefficient mapping

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

Description Keys of these mappings could be packed in a way similar to how keys of balancesToWithdraw mapping are packed.

Listing 50: Inefficient mapping

```
89 mapping(uint32 => mapping(uint16 => bool)) public exited;
90 mapping(uint32 => mapping(uint32 => bool)) public swap_exited;

97 mapping(address => mapping(uint32 => bytes32)) public authFacts;
```

3.51 CVF-51 Inefficient address packing

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

Description It would be more gas efficient to pack address (20 bytes) and nonce (4 bytes) into a single 24-bytes key, rather then use a mapping with two separate keys.

Listing 51: Inefficient address packing

```
97 mapping(address => mapping(uint32 => bytes32)) public authFacts;
```

3.52 CVF-52 Intermixed type

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

Description Type uint256 is intermixed with its alias uint.
Recommendation Consider using consistent type names.

Listing 52: Intermixed type

```
106 uint256 expirationBlock;
```

3.53 CVF-53 Suboptimal gas usage

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

Recommendation It would be more gas efficient to replace this mapping with a dynamic array. This would also make totalOpenPriorityRequests storage variable redundant.

Listing 53: Suboptimal gas usage

```
112 mapping(uint64 => PriorityOperation) public priorityRequests;
```

3.54 CVF-54 Incorrect zkSyncCommitBlockAddress storage type

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

Recommendation This storage variable should have type ZkSyncCommitBlock.

Listing 54: Incorrect zkSyncCommitBlockAddress storage type

```
134 address public zkSyncCommitBlockAddress;
```

3.55 CVF-55 Incorrect zkSyncExitAddress storage type

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

Recommendation This storage variable should have type ZkSyncExit.

Listing 55: Incorrect zkSyncExitAddress storage type

```
135 address public zkSyncExitAddress;
```

3.56 CVF-56 Incorrect line refer

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

Description This line should refer to zkSwap, rather than zkSync.
Recommendation Probably, some credit to zkSync should also be here.

Listing 56: Incorrect line refer

```
4 @title zkSync configuration constants
```

3.57 CVF-57 Unmentioned author

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

Description Should the author of the changes be also mentioned here?

Listing 57: Unmentioned author

```
5 @author Matter Labs
```

3.58 CVF-58 Improper declaration

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

Description As this contract contains only constants, it could be turned into a library.

Listing 58: Improper declaration

```
6 contract Config {
```

3.59 CVF-59 Missing access level

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

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

Listing 59: Missing access level

```
 9  uint256 constant ERC20_WITHDRAWAL_GAS_LIMIT = 250000;

12  uint256 constant ETH_WITHDRAWAL_GAS_LIMIT = 10000;

15  uint8 constant CHUNK_BYTES = 9;

18  uint8 constant ADDRESS_BYTES = 20;

20  uint8 constant PUBKEY_HASH_BYTES = 20;

23  uint8 constant PUBKEY_BYTES = 32;

26  uint8 constant ETH_SIGN_RS_BYTES = 32;

29  uint8 constant SUCCESS_FLAG_BYTES = 1;

32  uint16 constant MAX_AMOUNT_OF_REGISTERED_TOKENS = 128 - 1;

35  uint32 constant MAX_ACCOUNT_ID = (2 ** 24) - 1;

38  uint256 constant BLOCK_PERIOD = 15 seconds;

43  uint256 constant EXPECT_VERIFICATION_IN = 0 hours / BLOCK_PERIOD

45  uint256 constant NOOP_BYTES = 1 * CHUNK_BYTES;
    (...)

55  uint256 constant FULL_EXIT_BYTES = 6 * CHUNK_BYTES;

58  uint256 constant ONCHAIN_WITHDRAWAL_BYTES = 1 + 20 + 2 + 16; //
    (...)

61  uint256 constant CHANGE_PUBKEY_BYTES = 6 * CHUNK_BYTES;

65  uint256 constant PRIORITY_EXPIRATION_PERIOD = 3 days;

68  uint256 constant PRIORITY_EXPIRATION
    (...)
```

3.60 CVF-60 Improper type

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

Description Using uint8 type here doesn't save gas, as all calculations are anyway performed with 256-bit numbers, but may lead to hard to track overflow errors.
Recommendation Consider using uint256 type instead.

Listing 60: Improper type

```
15 uint8 constant CHUNK_BYTES = 9;

18 uint8 constant ADDRESS_BYTES = 20;

20 uint8 constant PUBKEY_HASH_BYTES = 20;

23 uint8 constant PUBKEY_BYTES = 32;

26 uint8 constant ETH_SIGN_RS_BYTES = 32;

29 uint8 constant SUCCESS_FLAG_BYTES = 1;
```

3.61 CVF-61 Improper naming

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

Description There should be "number" word instead of "amount" in the name of the constant, as "amount" is used for uncountables.

Listing 61: Improper naming

```
32 uint16 constant MAX_AMOUNT_OF_REGISTERED_TOKENS = 128 - 1;
```

3.62 CVF-62 Suboptimal uint16 type

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

Description Using uint16 type here doesn't save gas, as all calculations are anyway performed with 256-bit numbers, but may lead to hard to track overflow errors.
Recommendation Consider using uint256 type instead.

Listing 62: Suboptimal uint16 type

```
32 uint16 constant MAX_AMOUNT_OF_REGISTERED_TOKENS = 128 - 1;
```

3.63 CVF-63 Suboptimal number form

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

Recommendation Such numbers are more readable in the hexadecimals form, like 0x7F or 0xFFFFFF.

Listing 63: Suboptimal number form

```
32 uint16 constant MAX_AMOUNT_OF_REGISTERED_TOKENS = 128 - 1;

35 uint32 constant MAX_ACCOUNT_ID = (2 ** 24) - 1;
```

3.64 CVF-64 Improper type

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

Description Using uint32 type here doesn't save gas, as all calculations are anyway performed with 256-bit numbers, but may lead to hard to track overflow errors.
Recommendation Consider using uint256 type instead.

Listing 64: Improper type

```
35 uint32 constant MAX_ACCOUNT_ID = (2 ** 24) - 1;
```

3.65 CVF-65 Timing

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

Description Block period is usually somewhere between 13 and 14 seconds: https://etherscan.io/chart/blocktime

Listing 65: Timing

```
38 uint256 constant BLOCK_PERIOD = 15 seconds;
```

3.66 CVF-66 Unclear assignment

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

Description This constant is zero. Is it really needed? Code, tested with zero value, may behave incorrectly with non-zero value. Also, the division rounds down, while this value should probably be rounded up.

Listing 66: Unclear assignment

```
43 uint256 constant EXPECT_VERIFICATION_IN = 0 hours / BLOCK_PERIOD
   ↪ ;
```

3.67 CVF-67 Unclear behavior

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

Description It is not clear what should happen if any block can be reverted any time.
Recommendation Some clarification is needed in the documentation probably.

Listing 67: Unclear behavior

```
42 /// If set to 0 validator can revert blocks at any time.
```

3.68 CVF-68 Readability issue

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

Recommendation There should be constants OPCODE_BYTES=1, TOKEN_ID_BYTES=2 and AMOUNT_BYTES=16, so the expression could look like: OPCODE_BYTES + ADDRESS_BYTES + TOKEN_ID_BYTES + AMOUNT_BYTES.

Listing 68: Readability issue

```
58 uint256 constant ONCHAIN_WITHDRAWAL_BYTES = 1 + 20 + 2 + 16; //
   ↪ (uint8 addToPendingWithdrawalsQueue, address _to, uint16
   ↪ _tokenId, uint128 _amount)
```

3.69 CVF-69 Suspicious rounding

- **Severity** Moderate
- **Status** Info
- **Category** Unclear behavior
- **Source** Config.sol

Description The division here rounds down, while it should probably round up.
Client Comment Irrelevant. Only 1 sec of difference, which is within block time.

Listing 69: Suspicitious rounding

```
68 uint256 constant PRIORITY_EXPIRATION =
     ↪ PRIORITY_EXPIRATION_PERIOD / BLOCK_PERIOD;
```

3.70 CVF-70 Unclear estimate

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

Description The 750k gas estimate is valid for certain proof format only. Has it remained the same as in the original zkSync?

Listing 70: Unclear estimate

```
72 /// @dev Value based on the assumption of ~750k gas cost of
     ↪ verifying and 5 used storage slots per PriorityOperation
     ↪ structure
```

3.71 CVF-71 Incorrect uint64 type using

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

Description Using uint64 type here doesn't save gas, as all calculations are anyway performed with 256-bit numbers, but may lead to hard to track overflow errors.
Recommendation Consider using uint256 type instead.

Listing 71: Incorrect uint64 type using

```
73 uint64 constant MAX_PRIORITY_REQUESTS_TO_DELETE_IN_VERIFY = 6;
```

3.72 CVF-72 Intermixing uint256 type

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

Description Intermixing uint256 type with its alias uint makes code harder to read.
Recommendation Consider using the same name for this type everywhere.

Listing 72: Intermixing uint256 type

```
76 uint constant MASS_FULL_EXIT_PERIOD = 3 days;

79 uint constant TIME_TO_WITHDRAW_FUNDS_FROM_FULL_EXIT = 2 days;

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

3.73 CVF-73 Inconsistent compiler version

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

Description In Config.sol is was ^0.5.0.
Recommendation Consider using consistent compiler version requirements across the code.

Listing 73: Inconsistent compiler version

```
1 solidity >=0.5.0 <0.7.0;
```

3.74 CVF-74 Out of scope files

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** DeployFactory.sol

Description These files are out of scope, so we don't review them.

Listing 74: Out of scope files

```
4 "./uniswap/UniswapV2Factory.sol";
  "./Proxy.sol";
  "./UpgradeGatekeeper.sol";
```

3.75 CVF-75 Suboptimal import

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

Description It is suboptimal to import the entire contract code which is not used explicitly. An interface file should have been imported instead, and this interface should contain function "setZkSyncAddress"

Listing 75: Suboptimal import

```
4 "./uniswap/UniswapV2Factory.sol";
```

3.76 CVF-76 Unseparated contracts

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

Description These contracts should be passed via interface files , as they are not inherited nor created in this contract.

Listing 76: Unseparated contracts

```
3 "./Governance.sol";

7 "./ZkSync.sol";
  "./Verifier.sol";
  "./VerifierExit.sol";
```

3.77 CVF-77 Unreviewed file

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** DeployFactory.sol

Description We don't have this file, so we don't review it.

Listing 77: Unreviewed file

```
10 import "./TokenInit.sol";
```

3.78 CVF-78 Unreviewed contract

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** DeployFactory.sol

Description We don't have sources for the TokenDeployInit contract, so we cannot fully review the functionality of the DeployFactory contract.

Listing 78: Unreviewed contract

```
12 DeployFactory is TokenDeployInit {
```

3.79 CVF-79 Suboptimal formatting

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

Recommendation Formatting the list of formal argument one argument per line would make code more readable.

Listing 79: Suboptimal formatting

```
31 Governance _govTarget, UniswapV2Factory _pairTarget, address
   ↪ _blockCommit, address _exit, Verifier _verifierTarget,
   ↪ VerifierExit _verifierExitTarget, ZkSync _zkSyncTarget,
bytes32 _genesisRoot, address _firstValidator, address _governor
   ↪ ,

48 Governance _governanceTarget, UniswapV2Factory _pairTarget,
   ↪ address _blockCommit, address _exit, Verifier
   ↪ _verifierTarget, VerifierExit _verifierExitTarget, ZkSync
   ↪ _zksyncTarget,
bytes32 _genesisRoot, address _validator, address _governor
```

3.80 CVF-80 Missing event

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

Description It would be convenient to log the value of _feeAccountAddress in an event, to make it easier to recover it.

Listing 80: Missing event

```
33 address _feeAccountAddress
```

3.81 CVF-81 Redundant checks

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

Description These checks don't make much sense, as non-zero dead addresses could be provided anyway.
Recommendation Consider removing these checks.

Listing 81: Redundant checks

```
35 require(_firstValidator != address(0));
   require(_governor != address(0));
   require(_feeAccountAddress != address(0));
```

3.82 CVF-82 Unclear split purpose

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

Description What is the reason to split the list of addresses into two events? A single event could handle all the addresses and this will be more gas efficient.

Listing 82: Unclear split purpose

```
44 event Addresses(address governance, address zksync, address
     ↪ verifier, address pair, address gatekeeper);
   event AddressesOther(address commitblock, address exit, address
     ↪ verifierexit);
```

3.83 CVF-83 Improper type

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

Description Variable governance should have type Governance.

Listing 83: Improper type

```
52 Proxy governance = new Proxy(address(_governanceTarget), abi.
     ↪ encode(this));
```

3.84 CVF-84 Improper type

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

Description Variable pair should have type UniswapV2Factory.

Listing 84: Improper type

```
53 Proxy pair = new Proxy(address(_pairTarget), abi.encode());
```

3.85 CVF-85 Redundant casts

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

Description Casts to address in abi.encode call are redundant.
Recommendation Concract references could be passed as is.

Listing 85: Redundant casts

```
57 Proxy zkSync = new Proxy(address(_zksyncTarget), abi.encode(
   ↪ address(governance), address(verifier), address(
   ↪ verifierExit), address(pair)));
```

3.86 CVF-86 Improper type

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

Description Variable zkSync should have type ZkSync.

Listing 86: Improper type

```
57 Proxy zkSync = new Proxy(address(_zksyncTarget), abi.encode(
   ↪ address(governance), address(verifier), address(
   ↪ verifierExit), address(pair)));
```

3.87 CVF-87 Unreviewed function

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** DeployFactory.sol

Description We didn't review the getTokens function.

Listing 87: Unreviewed function

```
89 address[] memory tokens = getTokens();
```

3.88 CVF-88 Unmentioned author

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

Description Should the author of the changes be also mentioned here?

Listing 88: Unmentioned author

```
7 @author Matter Labs
```

3.89 CVF-89 Improper type

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

Description The type of the token parameter could be made more specific.

Listing 89: Improper type

```
12 address indexed token,
```

3.90 CVF-90 Not separated ValidatorStatusUpdate event

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

Recommendation It would be cheaper to have two separate events: one for authorizing a new validator and another for revoking validator authorization.

Listing 90: Not separated ValidatorStatusUpdate event

```
22 event ValidatorStatusUpdate(
```

3.91 CVF-91 Redundant word

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

Recommendation Word "address" is redundant in the parameter name. Just "validator" would be enough.

Listing 91: Redundant word

```
23 address indexed validatorAddress,
```

3.92 CVF-92 Not indexed parameter

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

Recommendation This parameter should be indexed.

Listing 92: Not indexed parameter

```
24 bool isActive
```

3.93 CVF-93 Improper type

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

Description Values in this mapping should have more specific type than just address. Something like IERC20.

Listing 93: Improper type

```
34 mapping(uint16 => address) public tokenAddresses;
```

3.94 CVF-94 Inefficient mapping

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

Recommendation It would be more efficient to replace this mapping with a dynamic array. This would also make the totalTokens storage variable unnecessary.

Listing 94: Inefficient mapping

```
34 mapping(uint16 => address) public tokenAddresses;
```

3.95 CVF-95 Specific type missing

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

Recommendation Keys in this mapping should have more specific type than just address. Something like IERC20.

Listing 95: Specific type missing

```
37 mapping(address => uint16) public tokenIds;
```

3.96 CVF-96 Redundant line

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

Description This line is redundant.

Listing 96: Redundant line

```
42 constructor() public {}
```

3.97 CVF-97 Not checked parameter line

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

Description The length of the initializationParameters parameter is not checked, so extra data at the end will be silently ignored. Probably, not an issue.

Listing 97: Not checked parameter line

```
47 function initialize(bytes calldata initializationParameters)
   ↪ external {
```

3.98 CVF-98 Redundant variable

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

Description The _networkGovernor variable is redundant, as the value returned by the abi.decode call may be directly assigned to the networkGovernor storage variable.

Listing 98: Redundant variable

```
48 address _networkGovernor = abi.decode(initializationParameters,
   ↪ (address));

50 networkGovernor = _networkGovernor;
```

3.99 CVF-99 Out of scope contract

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

Description The Proxy contract is out of scope, so we didn't check this claim.

Listing 99: Out of scope contract

```
53 /// @notice Governance contract upgrade. Can be external because
   ↪ Proxy contract intercepts illegal calls of this function.
```

3.100 CVF-100 Missed check

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** Governance.sol

Description This doesn't check that the token being added is not already registered as a pair token, thus it is possible to have two IDs for the same token.

Client Comment Governance ensures that this is not the case.

Listing 100: Missed check

```
71 require(tokenIds[_token] == 0, "gan11"); // token exists
```

3.101 CVF-101 Unclear constant value

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

Description The value of the MAX_AMOUNT_OF_REGISTERED_TOKENS constant is only 127, while the tokens counter and token IDs use uint16 type everywhere, which can hold numbers up to 65535. Why so?

Listing 101: Unclear constant value

```
72 require(totalTokens < MAX_AMOUNT_OF_REGISTERED_TOKENS, "gan12");
     ↪ // no free identifiers for tokens
```

3.102 CVF-102 Redundant code

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

Description These two lines could be merged together like this: uint16 newTokenID = ++totalTokens;

Listing 102: Redundant code

```
74 totalTokens++;
   uint16 newTokenId = totalTokens; // it is not `totalTokens - 1`
     ↪ because tokenId = 0 is reserved for eth
```

3.103 CVF-103 Redundant word "new"

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

Description Word "new" in the name of the variable is redundant.
Recommendation Just "tokenId" would be enough.

Listing 103: Redundant word "new"

```
75 uint16 newTokenId = totalTokens; // it is not `totalTokens - 1`
     ↪ because tokenId = 0 is reserved for eth
```

3.104 CVF-104 Improper modifier

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

Recommendation This function doesn't need to be public and thus be made internal.

Listing 104: Improper modifier

```
95 function requireGovernor(address _address) public view {
```

3.105 CVF-105 Redundant check

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

Description This check is redundant.

Listing 105: Redundant check

```
118 require(tokenId <= MAX_AMOUNT_OF_REGISTERED_TOKENS, "gvs12");
```

3.106 CVF-106 Redundant function

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

Description This function is redundant, as the tokenAddresses mapping is already public.

Listing 106: Redundant function

```
122 function getTokenAddress(uint16 _tokenId) external view returns
    ↪ (address) {
      address tokenAddr = tokenAddresses[_tokenId];
      return tokenAddr;
    }
```

3.107 CVF-107 Out of scope files

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** ZkSyncExit.sol

Description These files are out of scope and thus we didn't review them.

Listing 107: Out of scope files

```
3  "./ReentrancyGuard.sol";
   "./SafeMath.sol";
   "./SafeMathUInt128.sol";
   "./SafeCast.sol";
   "./Utils.sol";

14 "./uniswap/interfaces/IUniswapV2Pair.sol";
```

3.108 CVF-108 Unreviewed ReentrancyGuard contact

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** ZkSyncExit.sol

Description This contract inherits ReentrancyGuard that we didn't review, thus the behavior of this contract cannot be fully checked.

Listing 108: Unreviewed ReentrancyGuard contact

```
16 ZkSyncExit is PairTokenManager, Storage, Config, Events,
     ↪ ReentrancyGuard {
```

3.109 CVF-109 Doubly evaluated expression

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

Description Expression balncesToWithdraw[packedBalanceKey] is evaluated twice.
Recommendation Consider evaluating once and caching in a local variable.

Listing 109: Doubly evaluated expression

```
31 uint128 balance = balancesToWithdraw[packedBalanceKey].
     ↪ balanceToWithdraw;
   balancesToWithdraw[packedBalanceKey].balanceToWithdraw = balance
     ↪ .add(_amount);
```

3.110 CVF-110 Incorrect updateBalance access modify

- **Severity** Critical
- **Status** Fixed
- **Category** Suboptimal
- **Source** ZkSyncExit.sol

Description Anyway may call this function to increase his balance in any token, and then withdraw this increased balance.
Recommendation Consider making this function internal or protecting it in some other way from unauthorized access.

Listing 110: Incorrect updateBalance access modify

```
36 function updateBalance(uint16 _tokenId, uint128 _out) public {
```

3.111 CVF-111 Confusing function name

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

Description The name is confusing, as it suggests that the function doesn't modify blockchain state and only checks something, while actually, the function burns tokens.

Listing 111: Confusing function name

```
42 function checkLpL1Balance(address pair, uint128 _lpL1Amount)
   ↪ public {
```

3.112 CVF-112 Improper modifier

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** ZkSyncExit.sol

Description When called directly, this function may burn caller's LP tokens without giving anything in exchange.
Recommendation Consider making this function internal or protecting it in some other way from improper use.

Listing 112: Improper modifier

```
42 function checkLpL1Balance(address pair, uint128 _lpL1Amount)
   ↪ public {
```

3.113 CVF-113 Overflow in the line

- **Severity** Moderate
- **Status** Opened
- **Category** Overflow
- **Source** ZkSyncExit.sol

Description Overflow is possible here.

Client Comment Can't fix. Account balance on L2 is only 128 bits. No mainstream token would have this problem.

Listing 113: Overflow in the line

```
44 uint128 balance0 = uint128(IUniswapV2Pair(pair).balanceOf(msg.
   ↪ sender));
```

3.114 CVF-114 Redundadnt toSafeCast.toUint12 call

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

Description Call to SafeCast.toUint128 is redundant here as _lpL1Amount is already uint128.

Listing 114: Redundadnt toSafeCast.toUint12 call

```
49 pairmanager.burn(address(pair), msg.sender, SafeCast.toUint128(
   ↪ _lpL1Amount)); //
```

3.115 CVF-115 Length check missing

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** ZkSyncExit.sol

Description There is no length check for _tokenIds array, while it seems that this array is supposed to contain exactly 3 elements.
Recommendation Consider adding explicit check: require(_tokenIDs.length == 3);
Client Comment Not a problem, tx would fail.

Listing 115: Length check missing

```
53 function checkPairAccount(address _pairAccount, uint16[] memory
   ↪ _tokenIds) view internal {
```

3.116 CVF-116 Incorrect check placement

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

Description These checks should be performed inside governance.getTokenAddress function.

Listing 116: Incorrect check placement

```
60 if (_tokenIds[1] != 0) {
       require(_token0 != address(0), "le8");

66 if (_tokenIds[2] != 0) {
       require(_token1 != address(0), "le7");
```

3.117 CVF-117 Redundant assignments

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

Description These assignments are redundant, as governance.getTokenAddress returns zero address for zero token ID.

Listing 117: Redundant assignments

```
63 _token0 = address(0);

69 _token1 = address(1);
```

3.118 CVF-118 Length check missing

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** ZkSyncExit.sol

Description There are no length checks for _accountIds, _addresses, _tokenIds, and _amounts arrays.
Recommendation Consider adding explicit checks.
Client Comment Not a problem, tx would fail.

Listing 118: Length check missing

```
75 function lpExit(bytes32 _rootHash, uint32[] calldata _accountIds
     ↪ , address[] calldata _addresses, uint16[] calldata
     ↪ _tokenIds, uint128[] calldata _amounts, uint256[] calldata
     ↪  _proof) external nonReentrant {
```

3.119 CVF-119 Predefined arrays lengths

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

Description Arguments _accountIds, _addresses, _tokenIds, and _amounts are all dynamic, meaning that their lengths are passed along with their contents. However, it seems that the lengths of these arrays are predefined. This is suboptimal.
Recommendation Consider passing all the data as a single struct instead of four dynamic arrays.

Listing 119: Predefined arrays lengths

```
76 /* data format:
      bytes32 _rootHash
       _owner_id = _accountIds[0]
       _pair_acc_id = _accountIds[1]
80     _owner_addr = _addresses[0]
       _pair_acc_addr = _addresses[1]
       _lp_token_id = _tokenIds[0]
       _token0_id = _tokenIds[1]
       _token1_id = _tokenIds[2]
       _lp_L2_amount = _amounts[0]
       _lp_L1_amount = _amounts[1]
       _balance0 = _amounts[2]
       _balance1 = _amounts[3]
       _out0 = _amounts[4]
90     _out1 = _amounts[5]
   */
```

3.120 CVF-120 Redundant _rootHash argument

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

Description The _rootHash argument is redundant, as its value could be derived.

Listing 120: Redundant _rootHash argument

```
94 require(_rootHash == blocks[totalBlocksVerified].stateRoot, "le1
     ↪ ");
```

3.121 CVF-121 Redundant _addresses[0] argument

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

Description The argument _addresses[0] is redundant, as its value could be derived.

Listing 121: Redundant _addresses[0] argument

```
96 require(msg.sender == _addresses[0], "le2");
```

3.122 CVF-122 Suboptimal check placement

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

Description This check should be performed earlier, before any expensive operations such as those that update blockchain state.

Listing 122: Suboptimal check placement

```
105 require(!swap_exited[_accountId][_pairAccountId], "le3"); //
    ↪ already exited
```

3.123 CVF-123 Suboptimal array elements placement

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

Description Indexed array elements here make code very hard to read.
Recommendation Consider adding comments to them telling their meanings.

Listing 123: Suboptimal array elements placement

```
106 bytes memory _account_data = abi.encodePacked(_rootHash,
    ↪ _accountId, _addresses[0], _amounts[0], _amounts[1]);
bytes memory _pair_data0 = abi.encodePacked(_pairAccountId,
    ↪ _addresses[1], _tokenIds[0], _tokenIds[1], _tokenIds[2]);
bytes memory _pair_data1 = abi.encodePacked(_amounts[2],
    ↪ _amounts[3], _amounts[4], _amounts[5]);
```

3.124 CVF-124 Unreviewed file

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

Description This file is out of scope, so we didn't review it.

Listing 124: Unreviewed file

```
3 import "./Bytes.sol";
```

3.125 CVF-125 Incorrect line zkSync refer-2

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

Description This line should refer to zkSwap, rather than zkSync.
Recommendation Probably, some credit to zkSync should also be here.

Listing 125: Incorrect line zkSync refer-2

```
6 @title zkSync operations tools
```

3.126 CVF-126 Not used _CloseAccount contract

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

Description This constant is actually not used and its name is confusing, as one could think that it has some business beaning. Something like "NotUsed" would be better.

Listing 126: Not used _CloseAccount contract

```
17 _CloseAccount, // used for correct op id offset
```

3.127 CVF-127 Unclear comment

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

Description Why do we need to maintain particular values for operation IDs at all?

Listing 127: Unclear comment

```
17 _CloseAccount, // used for correct op id offset
```

3.128 CVF-128 Constant missing

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

Description Here should probably be a constant for opcode length: OPCODE_BYTES = 1

Listing 128: Constant missing

```
27 // Byte lengths
```

3.129 CVF-129 Suboptimal using of the uint8 type

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

Description Using the uint8 type here does not reduce gas cost but may cause hard to track overflow errors.
Recommendation Consider using the uint256 type instead.

Listing 129: Suboptimal using of the uint8 type

```
29 uint8 constant TOKEN_BYTES = 2;

31 uint8 constant PUBKEY_BYTES = 32;

33 uint8 constant NONCE_BYTES = 4;

35 uint8 constant PUBKEY_HASH_BYTES = 20;

37 uint8 constant ADDRESS_BYTES = 20;

40 uint8 constant FEE_BYTES = 2;

43 uint8 constant ACCOUNT_ID_BYTES = 4;

45 uint8 constant AMOUNT_BYTES = 16;

48 uint8 constant SIGNATURE_BYTES = 64;
```

3.130 CVF-130 Confusing TOKEN_BYTES name

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

Description The name is confusing, as It is not toke, but rather token Id.
Recommendation Consider renaming to "TOKEN_ID_BYTES".

Listing 130: Confusing TOKEN_BYTES name

```
29 uint8 constant TOKEN_BYTES = 2;
```

3.131 CVF-131 Same constant

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

Description Exactly the same constants are already defined in to Config.sol file.
Recommendation Consider defining each constant only once.

Listing 131: Same constant

```
31 uint8 constant PUBKEY_BYTES = 32;

35 uint8 constant PUBKEY_HASH_BYTES = 20;

37 uint8 constant ADDRESS_BYTES = 20;
```

3.132 CVF-132 Similar constants

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

Description These constants look similar to the constants defined in the Config.sol file.
Recommendation Consider placing all similar constants at the same place.

Listing 132: Similar constants

```
29 uint8 constant TOKEN_BYTES = 2;

33 uint8 constant NONCE_BYTES = 4;

40 uint8 constant FEE_BYTES = 2;

43 uint8 constant ACCOUNT_ID_BYTES = 4;

45 uint8 constant AMOUNT_BYTES = 16;

48 uint8 constant SIGNATURE_BYTES = 64;
```

3.133 CVF-133 Improper name

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

Recommendation TOKEN_ID_BYTES might be better name

Listing 133: Improper name

```
29 uint8 constant TOKEN_BYTES = 2;
```

3.134 CVF-134 Unused constant

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

Description This constant is not actually used. Its value looks suspicious as standard Ethereum signature is 65 rather than 64 bytes long.

Listing 134: Unused constant

```
48 uint8 constant SIGNATURE_BYTES = 64;
```

3.135 CVF-135 Improper modifier

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

Description These constants doesn't need to be public, as this library is not supposed to be deployed on its own, but only used internally by other smart contracts.

Listing 135: Improper modifier

```
 58 uint public constant PACKED_DEPOSIT_PUBDATA_BYTES =

102 uint public constant PACKED_FULL_EXIT_PUBDATA_BYTES =

207 uint public constant PACKED_CREATE_PAIR_PUBDATA_BYTES =
```

3.136 CVF-136 Length check missing

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** Operations.sol

Description There is no length check for the _data bytes array.
Recommendation Consider adding explicit check like this: require (_data.length = PACKED_DEPOSIT_PUBDATA_BYTES);

Listing 136: Length check missing

```
 62 function readDepositPubdata(bytes memory _data) internal pure

105 function readFullExitPubdata(bytes memory _data) internal pure

145 function readPartialExitPubdata(bytes memory _data, uint _offset
    ↪ ) internal pure

175 function readChangePubKeyPubdata(bytes memory _data, uint
    ↪ _offset) internal pure

187 function readWithdrawalData(bytes memory _data, uint _offset)
    ↪ internal pure

210 function readCreatePairPubdata(bytes memory _data) internal pure
```

3.137 CVF-137 Inefficien code

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

Description This code is suboptimal. It would be enough to just read two words from the memory like this:
require (_data.length == 42); uint256 a; address b; assembly { a := mload (add (_data, 0x20)) b := mload (add (_data, 0x2a)) } parsed.accountId = uint32 (a » 224); parsed.tokenId = uint16 (a » 208); parsed.amount = uint128 (a » 80); parsed.owner = b;

Listing 137: Inefficien code

```
 66 uint offset = 0;
    (offset, parsed.accountId) = Bytes.readUInt32(_data, offset); //
       ↪ accountId
    (offset, parsed.tokenId) = Bytes.readUInt16(_data, offset);   //
       ↪ tokenId
    (offset, parsed.amount) = Bytes.readUInt128(_data, offset);   //
       ↪ amount
 70 (offset, parsed.owner) = Bytes.readAddress(_data, offset);    //
       ↪ owner

109 uint offset = 0;
110 (offset, parsed.accountId) = Bytes.readUInt32(_data, offset);
       ↪      // accountId
    (offset, parsed.owner) = Bytes.readAddress(_data, offset);
       ↪          // owner
    (offset, parsed.tokenId) = Bytes.readUInt16(_data, offset);
       ↪         // tokenId
    (offset, parsed.amount) = Bytes.readUInt128(_data, offset);
       ↪         // amount

149 uint offset = _offset + ACCOUNT_ID_BYTES;                      //
       ↪ accountId (ignored)
150 (offset, parsed.tokenId) = Bytes.readUInt16(_data, offset); //
       ↪ tokenId
    (offset, parsed.amount) = Bytes.readUInt128(_data, offset); //
       ↪ amount
    offset += FEE_BYTES;                                           //
       ↪ fee (ignored)
    (offset, parsed.owner) = Bytes.readAddress(_data, offset);   //
       ↪ owner

178 uint offset = _offset;
    (offset, parsed.accountId) = Bytes.readUInt32(_data, offset);
       ↪                  // accountId
180 (...)
```

3.138 CVF-138 Improper modifier

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

Description This condition checks correctness of the code rather then correctness of the input, thus it should be "assert" rather then "require".

Listing 138: Improper modifier

```
72  require(offset == PACKED_DEPOSIT_PUBDATA_BYTES, "rdp10"); //
      ↪ reading invalid deposit pubdata size

115 require(offset == PACKED_FULL_EXIT_PUBDATA_BYTES, "rfp10"); //
      ↪ reading invalid full exit pubdata size

219 require(offset == PACKED_CREATE_PAIR_PUBDATA_BYTES, "rcp10"); //
      ↪  reading invalid create pair pubdata size
```

3.139 CVF-139 Improper type

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

Recommendation It should be "uint32(0)" and "uint16" rather then "bytes4(0)" and "bytes2(0)".

Listing 139: Improper type

```
78  bytes4(0),    // accountId (ignored) (update when
      ↪ ACCOUNT_ID_BYTES is changed)

158 bytes4(0),  // accountId (ignored) (update when ACCOUNT_ID_BYTES
      ↪  is changed)

161 bytes2(0),  // fee (ignored)  (update when FEE_BYTES is changed)

224 bytes4(0),      // accountId (ignored) (update when
      ↪ ACCOUNT_ID_BYTES is changed)
```

3.140 CVF-140 Suboptimal slice copying

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

Description It is suboptimal to copy a slice into a newly allocated bytes array just to hash it. **Recommendation** Consider implementing a function to hash a slice in-place like this: function hashSlice (bytes memory data, uint offset, uint length) internal pure returns (bytes32 hash) { require (offset <= data.length); require (length <= data.length - offset); assembly { hash := keccak256 (add (add (data, 0x20), offset), length) } }

Listing 140: Suboptimal slice copying

```
88  bytes memory lhs_trimmed = Bytes.slice(_lhs, ACCOUNT_ID_BYTES,
      ↪ PACKED_DEPOSIT_PUBDATA_BYTES - ACCOUNT_ID_BYTES);
    bytes memory rhs_trimmed = Bytes.slice(_rhs, ACCOUNT_ID_BYTES,
      ↪ PACKED_DEPOSIT_PUBDATA_BYTES - ACCOUNT_ID_BYTES);
90  return keccak256(lhs_trimmed) == keccak256(rhs_trimmed);

235 bytes memory lhs_trimmed = Bytes.slice(_lhs, ACCOUNT_ID_BYTES,
      ↪ PACKED_CREATE_PAIR_PUBDATA_BYTES - ACCOUNT_ID_BYTES);
    bytes memory rhs_trimmed = Bytes.slice(_rhs, ACCOUNT_ID_BYTES,
      ↪ PACKED_CREATE_PAIR_PUBDATA_BYTES - ACCOUNT_ID_BYTES);
    return keccak256(lhs_trimmed) == keccak256(rhs_trimmed);
```

3.141 CVF-141 Expensive comparison

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

Description For very short slices whose length is known at development time it could be cheaper to compare slices directly rather than to compare their hashes.

Listing 141: Expensive comparison

```
90  return keccak256(lhs_trimmed) == keccak256(rhs_trimmed);

237 return keccak256(lhs_trimmed) == keccak256(rhs_trimmed);
```

3.142 CVF-142 Same structure fields

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

Description This structure has exactly the same fields as the Deposit structure, but in a different order. Is this intentional?
Recommendation If it is not, consider using consistent field ordering across different structures.

Listing 142: Same structure fields

```
95  struct FullExit {
        uint32 accountId;
        address owner;
        uint16 tokenId;
        uint128 amount;
100 }
```

3.143 CVF-143 Structure slots

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

Description When a structure is saved in the storage, its fields are packed tightly but never cross slot boundary. So this structure will occupy 3 slots, however if the nonce field would go before the owner field, it would occupy only two slots. Probably not an issue, as this structure is not supposed to be saved in the storage.

Listing 143: Structure slots

```
169 uint32 accountId;
170 bytes20 pubKeyHash;
    address owner;
    uint32 nonce;
```

3.144 CVF-144 Similar function

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

Description This function is similar to the other "readXXX" functions in this file, but returns separate values rather than a structure. Is this intentional?
Recommendation If it is not, consider defining a structure for the values returned by this function.

Listing 144: Similar function

```
187 function readWithdrawalData(bytes memory _data, uint _offset)
    ↪ internal pure
```

3.145 CVF-145 Out of scope files

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

Description These files are out of scope, so we didn't review them.

Listing 145: Out of scope files

```
3  "./ReentrancyGuard.sol";
   "./SafeMath.sol";
   "./SafeMathUInt128.sol";
   "./SafeCast.sol";
   "./Utils.sol";

13 "./Bytes.sol";

16 "./UpgradeableMaster.sol";
   "./uniswap/UniswapV2Factory.sol";
```

3.146 CVF-146 Improper reference

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

Description This line should refer to zkSwap, rather than zkSync. Probably, some credit to zkSync should also be here.

Listing 146: Improper reference

```
21 @title zkSync main contract
```

3.147 CVF-147 Unmentioned author

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

Description Should the author of the changes be also mentioned here?

Listing 147: Unmentioned author

```
22 @author Matter Labs
```

3.148 CVF-148 Unreviewed UpgradeableMaster and ReentrancyGuard contracts

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

Description ZkSync contract inherits contracts UpgradeableMaster and ReentrancyGuard that we didn't review, so the behavior of ZkSync contract cannot be fully checked.

Listing 148: Unreviewed UpgradeableMaster and ReentrancyGuard contracts

```
23 ZkSync is PairTokenManager, UpgradeableMaster, Storage, Config,
    ↪ Events, ReentrancyGuard {
```

3.149 CVF-149 Incorrect ZkSync contract name

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

Recommendation The contract should be renamed to ZkSwap.

Listing 149: Incorrect ZkSync contract name

```
23 contract ZkSync is PairTokenManager, UpgradeableMaster, Storage,
    ↪ Config, Events, ReentrancyGuard {
```

3.150 CVF-150 Improper definition

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

Recommendation This constant should be defined like: bytes32 public constant EMPTY_STRING_KECCAK = keccak256 (""); There was a bug in Solidity compiler (https://github.com/ethereum/solidity/issues/4024) that caused some constant hashes to be reevaluated on every use, but even that bug didn't cause the hash of empty string ot be reevaluated.

Listing 150: Improper definition

```
27 bytes32 public constant EMPTY_STRING_KECCAK = 0
   ↪ xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470
   ↪ ;
```

3.151 CVF-151 Not used contract

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

Description This constant is not used. Is it really necessary? Is it necessary for this constant to be public?

Listing 151: Not used contract

```
27 bytes32 public constant EMPTY_STRING_KECCAK = 0
   ↪ xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470
   ↪ ;
```

3.152 CVF-152 Incorrect createPair behavior

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

Recommendation This function should return the ID of newly created pair and probably the address of it.

Listing 152: Incorrect createPair behavior

```
30 function createPair(address _tokenA, address _tokenB) external {
```

3.153 CVF-153 Incorrect maximum number of tokens

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

Description The functions that create new pair tokens can be called by anybody. The maximum number of base (non-pair) tokens is 128, which allows creating up to 8128 different pair tokens. However, PairTokenManager allows only 1920 pair tokens to be registered. Thus, once at least 63 base tokens are registered, an attacker may quickly exhaust the maximum allowed number of pair tokens, making it impossible to register more pair tokens.
Recommendation Consider either increasing the maximum number of pair tokens to at least 8128, or making only the governance to be able to register pair tokens.
Client Comment Limited to governor.

Listing 153: Incorrect maximum number of tokens

```
30 function createPair(address _tokenA, address _tokenB) external {

51 function createETHPair(address _tokenERC20) external {
```

3.154 CVF-154 Reordered tokens

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

Description Uniswap may reorder tokens in a pair. Probably not an issue.

Listing 154: Reordered tokens

```
39 address pair = pairmanager.createPair(_tokenA, _tokenB);

57 address pair = pairmanager.createPair(address(0), _tokenERC20);
```

3.155 CVF-155 Redundant checks

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

Description This check is redundant, as createPair function does not signal error by returning zero address, but rather just reverts the transaction.

Listing 155: Redundant checks

```
40 require(pair != address(0), "pair is invalid");

58 require(pair != address(0), "pair is invalid");
```

3.156 CVF-156 Unreviewed createPair function

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

Description The implementation of createPair function is out of scope, so we didn't review it.

Listing 156: Unreviewed createPair function

```
39 address pair = pairmanager.createPair(_tokenA, _tokenB);

57 address pair = pairmanager.createPair(address(0), _tokenERC20);
```

3.157 CVF-157 Unnecessary call

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

Description This call would not be necessary, in case addPairToken would return the ID of added pair.

Listing 157: Unnecessary call

```
46     validatePairTokenAddress(pair),

65 validatePairTokenAddress(pair),
```

3.158 CVF-158 Function similar to createPair function

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

Description This function has much in common with createPair function.
Recommendation Consider extracting the common parts into a utility function.

Listing 158: Function similar to createPair function

```
50 //create pair including ETH
   function createETHPair(address _tokenERC20) external {
       requireActive();
       //check _tokenERC20 is registered or not
       uint16 erc20ID = governance.validateTokenAddress(_tokenERC20
          ↪ );

       //create pair
       address pair = pairmanager.createPair(address(0),
          ↪ _tokenERC20);
       require(pair != address(0), "pair is invalid");

60     addPairToken(pair);

       registerCreatePair(
           0,
           erc20ID,
           validatePairTokenAddress(pair),
           pair);
   }
```

3.159 CVF-159 Not reordered tokens

- **Severity** Moderate
- **Status** Info
- **Category** Unclear behavior
- **Source** ZkSync.sol

Description Probably, in case Uniswap has reordered the tokens in the pair, here tokens should be reordered as well.

Client Comment Governor gaurantees order.

Listing 159: Not reordered tokens

```
73 tokenA: _tokenA,
   tokenB: _tokenB,
```

3.160 CVF-160 Improper type

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

Description Type uint is intermixed with its alias uint256 which makes code harder to read.
Recommendation Consider using consistent type names.

Listing 160: Improper type

```
87 function getNoticePeriod() external returns (uint) {
```

3.161 CVF-161 Outdated function name

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

Description This contract is not called Franklin actually, it is an old name of zkSync

Listing 161: Outdated function name

```
120 /// @notice Franklin contract initialization. Can be external
    ↪ because Proxy contract intercepts illegal calls of this
    ↪ function.
```

3.162 CVF-162 Redundant code line

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

Recommendation This line should be probably removed

Listing 162: Redundant code line

```
124 /// _ // FIXME: remove _genesisAccAddress
```

3.163 CVF-163 Incorrect variable type

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

Recommendation Variable type should be Governance instead of just address.

Listing 163: Incorrect variable type

```
130 address _governanceAddress,
```

3.164 CVF-164 Incorrect variable type-2

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

Recommendation Variable type should be Verifier instead of just address.

Listing 164: Incorrect variable type-2

```
131 address _verifierAddress,
```

3.165 CVF-165 Incorrect variable type-3

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

Recommendation Variable type should be VerifierExit instead of just address.

Listing 165: Incorrect variable type-3

```
132 address _verifierExitAddress,
```

3.166 CVF-166 Incorrect variable type-4

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

Recommendation Variable type should be UniswapV2Factory instead of just address.

Listing 166: Incorrect variable type-4

```
133 address _pairManagerAddress
```

3.167 CVF-167 Improper call

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

Recommendation Should be abi.decode(initializationParameters, (Governance, Verifier, VerifierExit, UniswapV2Factory))

Listing 167: Improper call

```
134 ) = abi.decode(initializationParameters, (address, address,
    ↪ address, address));
```

3.168 CVF-168 Vulnerable setGenesisRootAndAddresses function

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

Description The function may be called by anyone to overwrite sensitive information in contract's storage.
Recommendation It should be made callable at most once, or protected in some other way.
Client Comment Can be set only once now.

Listing 168: Vulnerable setGenesisRootAndAddresses function

```
142 function setGenesisRootAndAddresses(bytes32 _genesisRoot,
    ↪ address _zkSyncCommitBlockAddress, address
    ↪ _zkSyncExitAddress) external {
      blocks[0].stateRoot = _genesisRoot;
      zkSyncCommitBlockAddress = _zkSyncCommitBlockAddress;
      zkSyncExitAddress = _zkSyncExitAddress;
    }
```

3.169 CVF-169 Redundant validatePairTokenAddress call

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

Description This call is redundant, as it will always succeed here.

Listing 169: Redundant validatePairTokenAddress call

```
164 validatePairTokenAddress(address(_token));
```

3.170 CVF-170 Out of scope mint function

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

Description The implementation of mint function is out of scope, so we didn't review it.

Listing 170: Out of scope mint function

```
165 pairmanager.mint(address(_token), _to, _amount);
```

3.171 CVF-171 Out of scope sendERC20 function

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

Description The implementation of sendERC20 function is out of scope, so we didn't review it.

Listing 171: Out of scope sendERC20 function

```
167 require(Utils.sendERC20(_token, _to, _amount), "wtg11"); //
    ↪ wtg11 - ERC20 transfer fails
```

3.172 CVF-172 Out of scope minU32 function

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

Description Function minU32 is out of scope, so we didn't review it.

Listing 172: Out of scope minU32 function

```
180 uint32 toProcess = Utils.minU32(_n, numberOfPendingWithdrawals);
```

3.173 CVF-173 Muptiple times value calculation

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

Description The value startIndex + toProcess is calculated at every loop iteration.
Recommendation Consider calculating once and caching in a local variable.

Listing 173: Muptiple times value calculation

```
185 for (uint32 i = startIndex; i < startIndex + toProcess; ++i) {
```

3.174 CVF-174 Already calculated value

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

Description The value pendingWithdrawals[i] was already calculated in the previous line.
Recommendation Consider calculating once and caching in a local variable.

Listing 174: Already calculated value

```
187 address to = pendingWithdrawals[i].to;

189 delete pendingWithdrawals[i];
```

3.175 CVF-175 Already calculated value

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

Description The value balancesToWithdraw[packedBalanceKey] was already calculated few lines above.
Recommendation Consider calculating once and caching in a local variable.

Listing 175: Already calculated value

```
195 balancesToWithdraw[packedBalanceKey].balanceToWithdraw -= amount
    ↪ ;
```

3.176 CVF-176 Expensive assignment

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

Recommendation It would be cheaper to just assign zero to balancesToWithdraw[packedBalanceKey].balanceToWithdraw here, as the subtraction result is always zero.

Listing 176: Expensive assignment

```
195 balancesToWithdraw[packedBalanceKey].balanceToWithdraw -= amount
    ↪ ;
```

3.177 CVF-177 Improper cast

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

Description 'to ' is already of type 'address'

Listing 177: Improper cast

```
198 address payable toPayable = address(uint160(to));
```

3.178 CVF-178 Out of scope sendETHNoRevert function

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

Description Function sendETHNoRevert is out of scope, so we didn't review it.

Listing 178: Out of scope sendETHNoRevert function

```
199 sent = Utils.sendETHNoRevert(toPayable, amount);
```

3.179 CVF-179 Redundant assignment

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

Description The assignment here is redundant, as the assigned value will anyway be overwritten a few lines below.

Listing 179: Redundant assignment

```
201 address tokenAddr = address(0);
```

3.180 CVF-180 Redundant registry of tokens

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

Description There should be a single registry of tokens. Having two registries makes code harder to read, less efficient, and more error prone.

Listing 180: Redundant registry of tokens

```
204 tokenAddr = governance.tokenAddresses(tokenId);

207 tokenAddr = tokenAddresses[tokenId];
```

3.181 CVF-181 Suboptimal condition usage

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

Recommendation As this check should never fail, assert should be used here instead of require.

Listing 181: Suboptimal condition usage

```
210 require(tokenAddr != address(0), "cwt0");
```

3.182 CVF-182 Suboptimal code pattern

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

Recommendation This should be wrapped into a utility function to make code easier to read and less error-prone.

Listing 182: Suboptimal code pattern

```
212 (sent, ) = address(this).call.gas(ERC20_WITHDRAWAL_GAS_LIMIT)(
        abi.encodeWithSignature("withdrawERC20Guarded(address,
            ↪ address,uint128,uint128)", tokenAddr, to, amount,
            ↪ amount)
```

3.183 CVF-183 Overflow

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

Description Overflow is possible here.

Client Comment Can't overflow here because it's just adding back the subtracted amount that failed to withdraw.

Listing 183: Overflow

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

3.184 CVF-184 Out of scope minU64 function

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

Description Function minU64 is out of scope, so we didn't review it.

Listing 184: Out of scope minU64 function

```
232 uint64 toProcess = Utils.minU64(totalOpenPriorityRequests, _n);
```

3.185 CVF-185 Multiple times function call

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

Description The value firstPriorityRequestId + toProcess is calculated at every loop iteration.
Recommendation Consider calculating once and caching in a local variable.

Listing 185: Multiple times function call

```
234 for (uint64 id = firstPriorityRequestId; id <
    ↪ firstPriorityRequestId + toProcess; id++) {
```

3.186 CVF-186 Already calculated value

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

Description The value priorityRequests[id] was already calculated in the previous line.
Recommendation Consider calculating once and caching in a local variable.

Listing 186: Already calculated value

```
236 Operations.Deposit memory op = Operations.readDepositPubdata(
    ↪ priorityRequests[id].pubData);
```

3.187 CVF-187 Overflow

- **Severity** Moderate
- **Status** Fixed
- **Category** Overflow
- **Source** ZkSync.sol

Description Overflow is possible here.

Client Comment Confirmed problem. For now governors shouldn't list tokens with crazy high supply and decimals. Mainstream tokens would be fine.

Listing 187: Overflow

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

3.188 CVF-188 Incorrect depositETH function behavior

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

Description This function allows registering zero-value deposits.
Recommendation Probably not an issue, but consider adding an explicit check that msg.value > 0.

Listing 188: Incorrect depositETH function behavior

```
248 function depositETH(address _franklinAddr) external payable
    ↪ nonReentrant {

265 function depositERC20(IERC20 _token, uint104 _amount, address
    ↪ _franklinAddr) external nonReentrant {
```

3.189 CVF-189 Incorrect withdrawETH function behavior

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

Description This function allows zero-value withdrawals.
Recommendation Probably not an issue, but consider adding explicit an check that _amount > 0.

Listing 189: Incorrect withdrawETH function behavior

```
255 function withdrawETH(uint128 _amount) external nonReentrant {
```

3.190 CVF-190 Non obvious check

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

Description The check that _amount doesn't exceed current user's balance is hidden inside this call which is non-obvious and makes code harder to read.
Recommendation Consider either adding explicit check before the call, or renaming registerWithdraw function to something like "validateAndRegisterWithdrawal".

Listing 190: Non obvious check

```
256 registerWithdrawal(0, _amount, msg.sender);

312 registerWithdrawal(tokenId, withdrawnAmount, msg.sender);
```

3.191 CVF-191 Complicated code

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

Description This is equivalent to: bool success = msg.sender.send (_amount);

Listing 191: Complicated code

```
257 (bool success, ) = msg.sender.call.value(_amount)("");
```

3.192 CVF-192 Complicated code

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

Description These two lines could be rewritten as: require (msg.sender.send (_amount), "fwell");

Listing 192: Complicated code

```
257 (bool success, ) = msg.sender.call.value(_amount)("");
    require(success, "fwe11"); // ETH withdraw failed
```

3.193 CVF-193 Suboptimal code placement

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

Description This code should be extracted into a utility function.

Listing 193: Suboptimal code placement

```
268 // Get token id by its address
    uint16 lpTokenId = tokenIds[address(_token)];
270 uint16 tokenId = 0;
    if (lpTokenId == 0) {
        // This means it is not a pair address
        tokenId = governance.validateTokenAddress(address(_token));
    } else {
        lpTokenId = validatePairTokenAddress(address(_token));
    }

301 uint16 lpTokenId = tokenIds[address(_token)];
    uint16 tokenId = 0;
    if (lpTokenId == 0) {
        // This means it is not a pair address
        tokenId = governance.validateTokenAddress(address(_token));
    } else {
        tokenId = validatePairTokenAddress(address(_token));
    }
```

3.194 CVF-194 Missing check

- **Severity** Major
- **Status** Ruled out
- **Category** Suboptimal
- **Source** ZkSync.sol

Description The fact that lpTokenId here is non-zero doesn't guarantee that the token is not registered within governance as a valid base (non-pair) token, as registered pair token can be later added to the governance, effectively making the token to have two different IDs.
Recommendation Consider either making it impossible for a token to have two IDs, or passing token ID instead of token addresses as a function parameter.
Client Comment Governance will ensure that the conditions are correct.

Listing 194: Missing check

```
274 } else {
288 } else {
306 } else {
```

3.195 CVF-195 Redundant assignment

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

Description This like is redundant, as all necessary checks were already made.

Listing 195: Redundant assignment

```
275 lpTokenId = validatePairTokenAddress(address(_token));
```

3.196 CVF-196 Out of scope transferFromERC20 function

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

Description Function transferFromERC20 is out of scope, so we didn't review it.

Listing 196: Out of scope transferFromERC20 function

```
290 require(Utils.transferFromERC20(_token, msg.sender, address(this
    ↪ ), SafeCast.toUint128(_amount)), "fd012"); // token
    ↪ transfer failed deposit
```

3.197 CVF-197 Not supported LP tokens

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

Description LP tokens are not supported here. Is this intentional?

Listing 197: Not supported LP tokens

```
322 uint16 tokenId;
    if (_token == address(0)) {
        tokenId = 0;
    } else {
        tokenId = governance.validateTokenAddress(_token);
        require(tokenId <= MAX_AMOUNT_OF_REGISTERED_TOKENS, "fee12")
            ↪ ;
    }
```

3.198 CVF-198 Redundant check

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

Description This check is redundant.

Listing 198: Redundant check

```
327 require(tokenId <= MAX_AMOUNT_OF_REGISTERED_TOKENS, "fee12");
```

3.199 CVF-199 Redundant gas usage

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

Description This sets gas reserve value even if it is already set or if balance to withdraw is already non-zero.
Recommendation Consider setting gas resertve value only when necessary.

Listing 199: Redundant gas usage

```
343 balancesToWithdraw[packedBalanceKey].gasReserveValue = 0xff;
```

3.200 CVF-200 Check missing

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

Description Here, the underflow check inside SafeMathUInt128.sub is used to enforce business-level constraint.
Recommendation Consider adding explicit check that _amount doesn't exceed current balance to withdraw.

Listing 200: Check missing

```
380 balancesToWithdraw[packedBalanceKey].balanceToWithdraw = balance
    ↪ .sub(_amount);
```

3.201 CVF-201 Out of scope sub function

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

Description The function "sub" is out of scope and thus was not reviewed.

Listing 201: Out of scope sub function

```
380 balancesToWithdraw[packedBalanceKey].balanceToWithdraw = balance
    ↪ .sub(_amount);
```

3.202 CVF-202 Overflow possible

- **Severity** Moderate
- **Status** Ruled out
- **Category** Overflow
- **Source** ZkSync.sol

Description Overflow is possible here.

Listing 202: Overflow possible

```
405 uint64 nextPriorityRequestId = firstPriorityRequestId +
    ↪ totalOpenPriorityRequests;
```

3.203 CVF-203 Overflow possible

- **Severity** Moderate
- **Status** Ruled out
- **Category** Overflow
- **Source** ZkSync.sol

Description Overflow is possible here.

Listing 203: Overflow possible

```
421 totalOpenPriorityRequests++;
```

3.204 CVF-204 Incorrect comment

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

Description While chaining contracts like this would probably work, the proper way to split lasrge contracts into parts that are deployed separately is to use libraries.

Listing 204: Incorrect comment

```
424 // The contract is too large. Break some functions to
    ↪ zkSyncCommitBlockAddress
```

3.205 CVF-205 Assembly not needed

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

Description Most of this logic could be rewritten in pure Solidity.

Listing 205: Assembly not needed

```
429 assembly {
430     calldatacopy(0, 0, calldatasize())
        let result := delegatecall(gas(), nextAddress, 0,
          ↪ calldatasize(), 0, 0)
        returndatacopy(0, 0, returndatasize())
        switch result
        case 0 {revert(0, returndatasize())}
        default {return (0, returndatasize())}
    }
```

3.206 CVF-206 Out of scope files

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** ZkSyncCommitBlock.sol

Description These files are out of scope, so we didn't review them.

Listing 206: Out of scope files

```
3  "./ReentrancyGuard.sol";
   "./SafeMath.sol";
   "./SafeMathUInt128.sol";
   "./SafeCast.sol";
   "./Utils.sol";

13 "./Bytes.sol";

16 "./uniswap/UniswapV2Factory.sol";
```

3.207 CVF-207 Out of scope ReentrancyGuard conctract

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** ZkSyncCommitBlock.sol

Description This smart contract inherits ReentrancyGuard that is out of scope, so we cannot full check the behavior of this smart contract.

Listing 207: Out of scope ReentrancyGuard conctract

```
21 ZkSyncCommitBlock is PairTokenManager, Storage, Config, Events,
     ↪ ReentrancyGuard {
```

3.208 CVF-208 Same SafeMath function

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

Description The same constant is defined in ZkSync.sol.
Recommendation Consider moving it to a common base contract or to a library used by both contracts.

Listing 208: Same SafeMath function

```
22 using SafeMath for uint256;
```

3.209 CVF-209 Improper usage of EMPTY_STRING_KECCAK

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

Description This constant should be defined like: bytes32 public constant EMPTY_STRING_KECCAK = keccak256 (""); There was a bug in Solidity compiler (https://github.com/ethereum/solidity/issues/4024) that caused some constant hashes to be reevaluated on every use, but even that bug didn't cause the hash of empty string ot be reevaluated.

Listing 209: Improper usage of EMPTY_STRING_KECCAK

```
25 bytes32 public constant EMPTY_STRING_KECCAK = 0
   ↪ xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470
   ↪ ;
```

3.210 CVF-210 Improper modifier

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

Description Is it necessary for this constant to be public?

Listing 210: Improper modifier

```
25 bytes32 public constant EMPTY_STRING_KECCAK = 0
   ↪ xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470
   ↪ ;
```

3.211 CVF-211 Suboptimal totalBlocksCommitted incrementing placement

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

Recommendation Incrementing totalBlocksCommitted before checking _blockNumber would make this "+ 1" unnecessary.

Listing 211: Suboptimal totalBlocksCommitted incrementing placement

```
43 require(_blockNumber == totalBlocksCommitted + 1, "fck11"); //
   ↪ only commit next block

58 totalBlocksCommitted++;
```

3.212 CVF-212 Redundant _publicData variable

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

Recommendation Making _publicData to be "memory" instead of "calldata" would make this variable unnecessary.

Listing 212: Redundant _publicData variable

```
47 bytes memory publicData = _publicData;
```

3.213 CVF-213 Incorrect collectOnchainOps and createCommittedBlock functions signature

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

Recommendation Changing the signatures of the functions collectOnchainOps and createCommittedBlock to accept publicData as "calldata" rather than "memory" reference, would probably reduce gas consumption.

Listing 213: Incorrect collectOnchainOps and createCommittedBlock functions signature

```
53 bytes32 withdrawalsDataHash = collectOnchainOps(_blockNumber,
   ↪ publicData, _ethWitness, _ethWitnessSizes);

57 createCommittedBlock(_blockNumber, _feeAccount, _newBlockInfo
   ↪ [0], publicData, withdrawalsDataHash,
   ↪ nPriorityRequestProcessed);
```

3.214 CVF-214 Expensive operation

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

Recommendation It would be cheaper to directly return the number of collected priority requests from the collectOnchainOps function rather then to derive this value from the changes of storage variables.

Listing 214: Expensive operation

```
55 uint64 nPriorityRequestProcessed =
   ↪ totalCommittedPriorityRequests -
   ↪ prevTotalCommittedPriorityRequests;
```

3.215 CVF-215 Check missing

- **Severity** Major
- **Status** Ruled out
- **Category** Suboptimal
- **Source** ZkSyncCommitBlock.sol

Description It is not checked that _blockNumber <= totalBlocksCommitted, thus one may try to verify not yet committed block.

Client Comment Verify could not succeed in this case

Listing 215: Check missing

```
68 function verifyBlock(uint32 _blockNumber, uint256[] calldata
   ↪ _proof, bytes calldata _withdrawalsData)
```

3.216 CVF-216 Suboptimal totalBlocksVerified placement

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

Description Incrementing totalBlocksVerified before checking _blockNumber would make this "+ 1" unnecessary.

Listing 216: Suboptimal totalBlocksVerified placement

```
72 require(_blockNumber == totalBlocksVerified + 1, "fvk11"); //
   ↪ only verify next block

83 totalBlocksVerified += 1;
```

3.217 CVF-217 Suboptimal Iterating block numbers

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

Recommendation Iterating block numbers down would be more elegant.

Listing 217: Suboptimal Iterating block numbers

```
99 for (uint32 i = totalBlocksCommitted - blocksToRevert + 1; i <=
   ↪ blocksCommited; i++) {
```

3.218 CVF-218 Redundant check

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

Description This check is redundant. If it is here just to ensure storage consistency, then it should be assert instead of require.

Listing 218: Redundant check

```
101 require(revertedBlock.committedAtBlock > 0, "frk11"); // block
    ↪ not found
```

3.219 CVF-219 Expensive revert

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

Recommendation Reverting would be cheaper in case a block would contain the cumulative number of priority operations in this block and all the previous blocks, rather then the number of priority operations just in this block.

Listing 219: Expensive revert

```
103 revertedPriorityRequests += revertedBlock.priorityOperations;
```

3.220 CVF-220 Wrong comment

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

Description The function returns true in case exodus mode was already active, while the documentation comment says, that it returns true in case exodus mode must be entered.

Listing 220: Wrong comment

```
118 /// @return bool flag that is true if the Exodus mode must be
    ↪ entered.

127         return true;
```

3.221 CVF-221 Wrong comment

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

Description This function should probably just revert in case exodus mode may not be entered or is already active.

Listing 221: Wrong comment

```
118 /// @return bool flag that is true if the Exodus mode must be
    ↪ entered.
```

3.222 CVF-222 Missed initialization

- **Severity** Moderate
- **Status** Ruled out
- **Category** Suboptimal
- **Source** ZkSyncCommitBlock.sol

Description Here non-initialized slot of priorityRequests mapping may be accessed.
Recommendation Consider rewriting the expression as: totalOpenPriorityRequests > 0 && block.number >= priorityRequests[firstPriorityRequestId].expirationBlock
Client Comment If 'firstPriorityRequestId' non-initialized, blockExpiration is 0, which is checked and would be denied in the second != 0 check.

Listing 222: Missed initialization

```
120 bool trigger = block.number >= priorityRequests[
    ↪ firstPriorityRequestId].expirationBlock &&
priorityRequests[firstPriorityRequestId].expirationBlock != 0;
```

3.223 CVF-223 Already calculated expression

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

Description Expression priorityRequests[firstPriorityRequestId].expirationBlock was already calculated in the previous line.
Recommendation Consider calculating once and caching in a local variable.

Listing 223: Already calculated expression

```
121 priorityRequests[firstPriorityRequestId].expirationBlock != 0;
```

3.224 CVF-224 Overcomplicated triggerExodusIfNeeded function

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

Description This function is overcomplicated.
Recommendation It could be rewritten in a straightforward way: require (totalOpenPriorityRequests > 0 && block.number >= priorityRequests[firstPriorityRequestId].expirationBlock); exodusMode = true; emit ExodusMode ();

Listing 224: Overcomplicated triggerExodusIfNeeded function

```
119 function triggerExodusIfNeeded() external returns (bool) {
120     bool trigger = block.number >= priorityRequests[
        ↪ firstPriorityRequestId].expirationBlock &&
        priorityRequests[firstPriorityRequestId].expirationBlock !=
        ↪ 0;
        if (trigger) {
            if (!exodusMode) {
                exodusMode = true;
                emit ExodusMode();
            }
            return true;
        } else {
            return false;
130     }
    }
```

3.225 CVF-225 Redundant hashing

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

Description Hashing the pub key again may not be needed

Listing 225: Redundant hashing

```
137 authFacts[msg.sender][_nonce] = keccak256(_pubkey_hash);
```

3.226 CVF-226 Incorrect check placement

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

Description This check is performed two times for each block.
Recommendation Consider moving it into the commitBlock function and do just once per block.

Listing 226: Incorrect check placement

```
152 require(_publicData.length % CHUNK_BYTES == 0, "cbb10"); //
    ↪ Public data size is not multiple of CHUNK_BYTES

196 require(_publicData.length % CHUNK_BYTES == 0, "fcs11"); //
    ↪ pubdata length must be a multiple of CHUNK_BYTES
```

3.227 CVF-227 Check missing

- **Severity** Moderate
- **Status** Ruled out
- **Category** Suboptimal
- **Source** ZkSyncCommitBlock.sol

Description It is not checked that _blocknumber - 1 is a valid number of a committed block.

Client Comment Checked by 'commitBlock'.

Listing 227: Check missing

```
161 blocks[_blockNumber - 1].stateRoot,
```

3.228 CVF-228 Field names missing

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

Recommendation Consider adding field names to make code more readable.

Listing 228: Field names missing

```
166 blocks[_blockNumber] = Block(
        uint32(block.number), // committed at
        _nCommittedPriorityRequests, // number of priority onchain
            ↪ ops in block
        blockChunks,
170     _withdrawalDataHash, // hash of onchain withdrawals data (
            ↪ will be used during checking block withdrawal data in
            ↪ verifyBlock function)
        commitment, // blocks' commitment
        _newRoot // new root
    );
```

3.229 CVF-229 Trivial function

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

Description These functions are trivial and is used only once.
Recommendation Consider inlining their logic.

Listing 229: Trivial function

```
176 function emitDepositCommitEvent(uint32 _blockNumber, Operations.
      ↪ Deposit memory depositData) internal {
      emit DepositCommit(_blockNumber, depositData.accountId,
        ↪ depositData.owner, depositData.tokenId, depositData.
        ↪ amount);
    }

180 function emitFullExitCommitEvent(uint32 _blockNumber, Operations
      ↪ .FullExit memory fullExitData) internal {
      emit FullExitCommit(_blockNumber, fullExitData.accountId,
        ↪ fullExitData.owner, fullExitData.tokenId, fullExitData
        ↪ .amount);
    }

184 function emitCreatePairCommitEvent(uint32 _blockNumber,
      ↪ Operations.CreatePair memory createPairData) internal {
      emit CreatePairCommit(_blockNumber, createPairData.accountId
        ↪ , createPairData.tokenA, createPairData.tokenB,
        ↪ createPairData.tokenPair, createPairData.pair);
    }
```

3.230 CVF-230 Incorrect function behavior

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

Description This function should also return the number of collected priority operations.

Listing 230: Incorrect function behavior

```
194 function collectOnchainOps(uint32 _blockNumber, bytes memory
      ↪ _publicData, bytes memory _ethWitness, uint32[] memory
      ↪ _ethWitnessSizes)
      internal returns (bytes32 withdrawalsDataHash) {
```

3.231 CVF-231 Variables read several times

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

Description Storage variables firstPriorityRequestId and totalCommittedPriorityRequests are read several times.
Recommendation Consider reading once and caching in a local variable.

Listing 231: Variables read several times

```
198 uint64 currentPriorityRequestId = firstPriorityRequestId +
      ↪ totalCommittedPriorityRequests;

316 require(currentPriorityRequestId <= firstPriorityRequestId +
      ↪ totalOpenPriorityRequests, "fcs16"); // fcs16 - excess
      ↪ priority requests in pubdata
totalCommittedPriorityRequests = currentPriorityRequestId -
      ↪ firstPriorityRequestId;
```

3.232 CVF-232 Redundant usage of memory pointers

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

Description There are no benefits from using memory pointers instead of array offsets in the code.
Recommendation Using offsets will make assembly sections unnecessary.

Listing 232: Redundant usage of memory pointers

```
200 uint256 pubDataPtr = 0;
uint256 pubDataStartPtr = 0;
uint256 pubDataEndPtr = 0;
```

3.233 CVF-233 Complicated code

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

Recommendation If array offsets would be used instead of memory pointers, this code could be rewrittes without assembly as: pubDataOffset = 0; pubDataLength = _publicData.length;

Listing 233: Complicated code

```
204 assembly { pubDataStartPtr := add(_publicData, 0x20) }
pubDataPtr = pubDataStartPtr;
pubDataEndPtr = pubDataStartPtr + _publicData.length;
```

3.234 CVF-234 Complicated code

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

Recommendation If array offsets would be used instead of memory pointers, this code could be rewritten without assembly as: Operations.OpType opType = Operations.OpType (uint8 (_publicData [pubDataOffset]));

Listing 234: Complicated code

```
214 Operations.OpType opType;
// read operation type from public data (the first byte per each
   ↪ operation)
assembly {
    opType := shr(0xf8, mload(pubDataPtr))
}
```

3.235 CVF-235 Redundant addToPendingWithdrawalsQueue variable

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

Recommendation This variable is redundant. Just use true instead of it.

Listing 235: Redundant addToPendingWithdrawalsQueue variable

```
254 bool addToPendingWithdrawalsQueue = true;
```

3.236 CVF-236 Redundant addToPendingWithdrawalsQueue variable

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

Recommendation This variable is redundant. Just use false instead of it.

Listing 236: Redundant addToPendingWithdrawalsQueue variable

```
264 bool addToPendingWithdrawalsQueue = false;
```

3.237 CVF-237 Suboptimal copy

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

Description Copying a slice into a new bytes array just to unpack it into a signature is suboptimal.
Recommendation Consider implementing in-place slice into signature unpacking.

Listing 237: Suboptimal copy

```
280 bytes memory currentEthWitness = Bytes.slice(_ethWitness,
    ↪ ethWitnessOffset, _ethWitnessSizes[
    ↪ processedOperationsRequiringEthWitness]);

    bool valid = verifyChangePubkeySignature(currentEthWitness, op.
    ↪ pubKeyHash, op.nonce, op.owner, op.accountId);
```

3.238 CVF-238 Suboptimal pack of several string literals

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

Description It seems that packing several string literals consumes more gas than packing a single pre-concatenated string literal.

Listing 238: Suboptimal pack of several string literals

```
327 bytes memory signedMessage = abi.encodePacked(
        "\x19Ethereum Signed Message:\n152",
        "Register ZKSwap pubkey:\n\n",
330     Bytes.bytesToHexASCIIBytes(abi.encodePacked(_newPkHash)), "\
          ↪ n",
        "nonce: 0x", Bytes.bytesToHexASCIIBytes(Bytes.
          ↪ toBytesFromUInt32(_nonce)), "\n",
        "account id: 0x", Bytes.bytesToHexASCIIBytes(Bytes.
          ↪ toBytesFromUInt32(_accountId)),
        "\n\n",
        "Only sign this message for a trusted client!"
    );
```

3.239 CVF-239 Dangerous behavior of the function

- **Severity** Moderate
- **Status** Ruled out
- **Category** Suboptimal
- **Source** ZkSyncCommitBlock.sol

Description Function recoverAddressFromEthSignature returns the result of ecrecover call as is. Function ecrecover returns zero address for invalid signature. This allows anybody to "sign" anything on behalf of zero address and thus clain ownership of zero address.

Client Comment In the Zk proof system operations that transfer fund to the zero addresses are invalid. So there's really no point to take over ownership of an account with address, nor is it possible to create an account with the zero address.

Listing 239: Dangerous behavior of the function

```
336 address recoveredAddress = Utils.recoverAddressFromEthSignature(
      ↪ _signature, signedMessage);
```

3.240 CVF-240 Unnecessary uint256 cast

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

Recommendation Explicit uint256 casts would not be necessary if abi.encode would be used instead of abi.encodePacked.

Listing 240: Unnecessary uint256 cast

```
355     abi.encodePacked(uint256(_blockNumber), uint256(_feeAccount)
          ↪ )

357 hash = sha256(abi.encodePacked(hash, uint256(_oldRoot)));
    hash = sha256(abi.encodePacked(hash, uint256(_newRoot)));
```

3.241 CVF-241 Unclear comment

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

Description It is unclear what problem comment talks about.

Listing 241: Unclear comment

```
383 // Use "invalid" to make gas estimation work
```

3.242 CVF-242 Calculated twice expression

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

Description Expression priorityRequests[_priorityRequestId] is calculated twice.
Recommendation Consider calculating once and caching in a local variable.

Listing 242: Calculated twice expression

```
394 Operations.OpType priorReqType = priorityRequests[
      ↪ _priorityRequestId].opType;
    bytes memory priorReqPubdata = priorityRequests[
      ↪ _priorityRequestId].pubData;
```

3.243 CVF-243 Double reading of storage variable

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

Description Storage variable numberOfPendingWithdrawals is read twice.
Recommendation Consider reading once and caching in a local variable.

Listing 243: Double reading of storage variable

```
420 uint32 localNumberOfPendingWithdrawals =
    ↪ numberOfPendingWithdrawals;

441 if (numberOfPendingWithdrawals !=
    ↪ localNumberOfPendingWithdrawals) {
```

3.244 CVF-244 Several times reading of storage variable

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

Description Storage variable firstPendingWithdrawalIndex is read several times.
Recommendation Consider reading once and caching in a local variable.

Listing 244: Several times reading of storage variable

```
433     pendingWithdrawals[firstPendingWithdrawalIndex +
        ↪ localNumberOfPendingWithdrawals] = PendingWithdrawal(
        ↪ _to, _tokenId);

442 emit PendingWithdrawalsAdd(firstPendingWithdrawalIndex +
    ↪ numberOfPendingWithdrawals, firstPendingWithdrawalIndex +
    ↪ localNumberOfPendingWithdrawals);
```

3.245 CVF-245 Redundant check

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

Description This check is redundant.

Listing 245: Redundant check

```
452 blocks[totalBlocksVerified + 1].committedAtBlock > 0 &&
```

3.246 CVF-246 Calculated expression twice

- **Severity** Minor
- **Status** Opened
- **Category**
- **Source** ZkSyncCommitBlock.sol

Description Expression totalBlocksVerified + 1 is calculated twice.
Recommendation Consider calculating once and caching in a local variable.

Listing 246: Calculated expression twice

```
452 blocks[totalBlocksVerified + 1].committedAtBlock > 0 &&
    block.number > blocks[totalBlocksVerified + 1].committedAtBlock
    ↪ + EXPECT_VERIFICATION_IN
```

3.247 CVF-247 Redundant requests

- **Severity** Major
- **Status** Ruled out
- **Category** Suboptimal
- **Source** ZkSyncCommitBlock.sol

Description As verifyBlock only tries to delete as many priority requests as were present in the verified block, the number of requests left undeleted due to this limitation will remain undeleted going forward.
Recommendation Consider deleting more requests than was in block, in case the number of priority requests in a block is less than MAX_PRIORITY_REQUESTS_TO_DELETE_IN_VERIFY and there are requests to be deleted remaining from previous the blocks.
Client Comment It is ok to leave these storages undeleted. deleting storage slots cost 5k gas per each slot it's unprofitable to clear too many slots.

Listing 247: Redundant requests

```
467 uint64 numberOfRequestsToClear = Utils.minU64(_number,
    ↪ MAX_PRIORITY_REQUESTS_TO_DELETE_IN_VERIFY);
```

3.248 CVF-248 Suboptimal contacts splitting

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

Recommendation While chaining contracts like this would probably work, the proper way to split large contracts into parts that are deployed separately is to use libraries.

Listing 248: Suboptimal contacts splitting

```
478 // The contract is too large. Break some functions to
    ↪ zkSyncCommitBlockAddress
```

3.249 CVF-249 Unnecessary assembly

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

Description Most of this logic could be rewritten in pure Solidity.

Listing 249: Unnecessary assembly

```
483 assembly {
        calldatacopy(0, 0, calldatasize())
        let result := delegatecall(gas(), nextAddress, 0,
          ↪ calldatasize(), 0, 0)
        returndatacopy(0, 0, returndatasize())
        switch result
        case 0 {revert(0, returndatasize())}
        default {return (0, returndatasize())}
490 }
```

4 References

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