Report

v. 1.0

Customer

zkLink

Smart Contract Audit

zkLink Nova

28th March 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	28.03.24	A. Zveryanskaya	Initial Draft
0.2	28.03.24	A. Zveryanskaya	Minor revision
1.0	28.03.24	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

zkLink is building an aggregated rollup infrastructure, secured by zero-knowledge proofs and multi-chain state synchronization, to aggregate and unify the liquidity from various Layer 1 and Layer 2 networks. zkLink Nova is an Aggregated Layer 3 Rollup zkEVM network built with ZK Stack and zkLink Nexus. Users on Nova have immediate access to the aggregated assets from Ethereum and integrated Ethereum Layer 2 networks. Nova inherits Ethereum's security by achieving multi-chain state synchronization through canonical rollup bridges.

3 Project scope

We were asked to review difference from zksync-era:

- Compared code
- Code with Fixes

And the code for the evm contracts with the provided fixes.

Files:

era/		
Storage.sol	ValidatorTimelock.sol	
era/facets/		
Admin.sol	Base.sol	Executor.sol
Getters.sol	Mailbox.sol	
era/interfaces/		
IAdmin.sol	IExecutor.sol	IGetters.sol
IL2Gateway.sol	IMailbox.sol	IZkLink.sol
era/libraries/		
PriorityQueue.sol		
evm/gateway/arbitrum/		
ArbitrumL1Gateway.sol	ArbitrumL2Gateway.sol	
evm/gateway/ethereum/		
EthereumGateway.sol		
evm/gateway/linea/		
LineaGateway.sol	LineaL1Gateway.sol	LineaL2Gateway.sol
evm/gateway/optimism/		
OptimismGateway.sol	OptimismL1Gateway.sol	OptimismL2Gateway.sol
evm/gateway/scroll/		
ScrollGateway.sol	ScrollL1Gateway.sol	ScrollL2Gateway.sol

evm/gateway/zkpolygon/		
ZkPolygonL1 Gateway.sol	ZkPolygonL2 Gateway.sol	
evm/gateway/zksync/		
ZkSyncL1Gateway.sol	ZkSyncL2Gateway.sol	
evm/gateway/		
BaseGateway.sol	L1BaseGateway.sol	L2BaseGateway.sol
evm/interfaces/		
IArbitrator.sol	IGateway.sol	IL1Gateway.sol
IL2Gateway.sol	IMessageClaimer.sol	IZkLink.sol
evm/interfaces/linea/		
IL2MessageService.sol	IMessageService.sol	
evm/interfaces/optimism/		
IOptimismMessenger.sol		
evm/interfaces/scroll/		
IScrollMessenger.sol		
evm/interfaces/zkpolygon/		
IBridgeMessage Receiver.sol	IZkPolygon.sol	
evm/interfaces/zksync/		
IL2ETHToken.sol	IL2Messenger.sol	IZkSyncL1Gateway.sol
evm/		
Arbitrator.sol	ZkLink.sol	

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 8 major, and a few less important issues. All identified Major issues have been fixed.

	Info	Fixed
Major	0	8
Moderate	3	1
Minor	38	28

Fixed 37 out of 78 issues

6 Major Issues

CVF-2. FIXED

- **Category** Suboptimal
- **Source** ZkPolygonL1Gateway.sol

Recommendation A bit mask would be more efficient if message numbers go in sequence.

```
23 23 /// @dev A mapping L2 batch number => message number => flag
24 24 /// @dev Used to indicate that zkSync L2 -> L1 message was already
      ↪ processed
25 25 mapping(uint256 => mapping(uint256 => bool)) public
      ↪ isMessageFinalized;
```

CVF-3. FIXED

- **Category** Unclear behavior
- **Source** BaseGateway.sol

Recommendation As zero address has the special meaning “not set”, consider explicitly requiring “_remoteGateway” to be not zero.

```
41 41 require(remoteGateway == address(0), "Duplicate␣init␣remote␣gateway"
      ↪ );
42 42 remoteGateway = _remoteGateway;
```

CVF-4. FIXED

- **Category** Unclear behavior
- **Source** ZkLink.sol

Recommendation Unchained version of initializers should be used to avoid double initialization of common base contracts.

```
140 140 __Ownable_init();
141 141 __UUPSUpgradeable_init();
142 142 __ReentrancyGuard_init();
143 143 __Pausable_init();
```

CVF-5. FIXED

- **Category** Unclear behavior
- **Source** ZkLink.sol

Recommendation As zero “gateway” address has the special meaning “not set”, consider explicitly requiring “_gateway” to be not zero.

```
189 189  gateway = _gateway;
190 190  emit InitGateway(_gateway);
```

CVF-6. FIXED

- **Category** Suboptimal
- **Source** Arbitrator.sol

Recommendation Unchained initializers should be used to avoid double initialization of common base contracts.

```
63 63  __Ownable_init();
64 64  __UUPSUpgradeable_init();
65 65  __ReentrancyGuard_init();
```

CVF-7. FIXED

- **Category** Unclear behavior
- **Source** Arbitrator.sol

Recommendation As zero gateway address has the special meaning “not set”, consider explicitly requiring “_gateway” to be not zero.

```
81 81  require(address(primaryChainGateway) == address(0), "Duplicate␣init␣
       ↪ gateway");
82 82  primaryChainGateway = _gateway;
```

CVF-8. FIXED

- **Category** Unclear behavior
- **Source** Admin.sol

Recommendation As zero address has a special meaning “not set”, there should be a check to ensure “_gateway” isn’t zero.

```
24  +require(address(s.gateway) == address(0), "g9");
25  +s.gateway = _gateway;
```

CVF-9. FIXED

- **Category** Suboptimal
- **Source** Mailbox.sol

Description Here all fields of the “SecondaryChain” structure are written into the storage, while only one field was actually changed.

Recommendation Consider writing only the changed field.

185
```
+s.secondaryChains[_secondaryChainGateway] = secondaryChain;
```

345
```
+    s.secondaryChains[_request.gateway] = secondaryChain;
```

7 Moderate Issues

CVF-1. INFO

- **Category** Suboptimal
- **Source** ZkSyncL1Gateway.sol

Recommendation A bit mask would be more efficient.

Client Comment *This contract was deployed on mainnet and it's hard for us to migrate to the recommand structure, so we do not plan to modity it.*

```
19 19 /// @dev A mapping L2 batch number => message number => flag
20 20 /// @dev Used to indicate that zkSync L2 -> L1 message was already
      ↪ processed
21 21 mapping(uint256 => mapping(uint256 => bool)) public
      ↪ isMessageFinalized;
```

CVF-10. INFO

- **Category** Unclear behavior
- **Source** LineaL2Gateway.sol

Description Exact equality check could cause problems in case the fee changes often.

Recommendation Consider allowing "msg.value" to be higher than the sum of "_value" and "coinbaseFee", and refund the surplus.

Client Comment *The msg value send to sendMessage is provided by the validator. The validator will estimate the transaction in advance to ensure that the coinbase fee provided to the linea canonical bridge is correct.*

```
24 24 require(msg.value == _value + coinbaseFee, "Invalid␣value");
```

CVF-11. FIXED

- **Category** Unclear behavior
- **Source** Arbitrator.sol

Description This should be executed only when the status of a secondary chain actually did change.

```
94 94  bytes memory callData = abi.encodeCall(IZkSync.
          ↪ setSecondaryChainGateway, (address(_gateway), _active));
95 95  // Forward fee to send message
96 96  primaryChainGateway.sendMessage{value: msg.value}(0, callData,
          ↪ _adapterParams);
97 97  emit SecondaryChainStatusUpdate(_gateway, _active);
```

CVF-12. INFO

- **Category** Suboptimal
- **Source** Arbitrator.sol

Description Using the “abi.encode” with a single variable-length argument is suboptimal and in theory may produce non-deterministic result.

Recommendation Consider using “abi.encodePacked” instead, or append the hash of “_callData” to a message to be hashed, instead of “_callData” itself. Note that ‘abi.encodePacked‘ should not be used when there are two or more variable length arguments

Client Comment *We do not plan to modify at this version. After the Dencun upgrade, we can store the ‘value‘ and ‘callData‘ in transient value instead of storing their hashes in the messageQueue. We have planned to implement it in the next version.*

```
136 136  bytes32 finalizeMessageHash = keccak256(abi.encode(_value, _callData
            ↪ ));
```

```
153 153  bytes32 finalizeMessageHash = keccak256(abi.encode(_value, _callData
            ↪ ));
```

8 Minor Issues

CVF-13. FIXED

- **Category** Documentation
- **Source** IL2Messenger.sol

Description The semantics of the returned value is unclear.

Recommendation Consider documenting.

```
7  7  function sendToL1(bytes memory _message) external returns (bytes32);
```

CVF-14. FIXED

- **Category** Bad datatype
- **Source** IZkPolygon.sol

Recommendation The type for these fields should be “IERC20”.

```
9  9  address token,
```

```
27 27 address originTokenAddress,
```

CVF-15. FIXED

- **Category** Documentation
- **Source** IL2MessageService.sol

Description This comment is confusing.

Recommendation Consider rephrasing it and adding more details.

```
7  7  /// @notice Returns coinbase fee when sendMessage
```

CVF-16. INFO

- **Category** Bad datatype
- **Source** IGateway.sol

Recommendation The return type should be more specific.

Client Comment *The remote gateway only used in encoding as an address and does not use its interface, so we do not plan to modify it.*

```
6  6  function getRemoteGateway() external view returns (address);
```

CVF-17. INFO

- **Category** Procedural
- **Source** ZkSyncL1Gateway.sol

Description We didn't review these files.

```
4  4  import {IMailbox} from "../../zksync/l1-contracts/zksync/interfaces/
      ↪ IMailbox.sol";
5  5  import {IGetters} from "../../zksync/l1-contracts/zksync/interfaces/
      ↪ IGetters.sol";
```

```
11 11 import {L2Message} from "../../zksync/l1-contracts/zksync/Storage.
      ↪ sol";
12 12 import {UnsafeBytes} from "../../zksync/l1-contracts/common/
      ↪ libraries/UnsafeBytes.sol";
13 13 import {L2_ETH_TOKEN_SYSTEM_CONTRACT_ADDR} from "../../zksync/l1-
      ↪ contracts/common/L2ContractAddresses.sol";
```

CVF-18. FIXED

- **Category** Procedural
- **Source** ZkSyncL1Gateway.sol

Recommendation It is a good practice to put a comment into a empty block to explain why the block is empty.

```
29 29 receive() external payable {}
```

CVF-19. FIXED

- **Category** Documentation
- **Source** ZkSyncL1Gateway.sol

Recommendation Consider adding a comment next to this argument with the argument name to clarify why the transaction origin address is used here.

```
50 50 // solhint-disable-next-line avoid-tx-origin
51 51 tx.origin
```

CVF-20. FIXED

- **Category** Bad datatype
- **Source** ZkSyncL1Gateway.sol

Recommendation The value “108” should be a named constant.

```
111 111 require(_message.length >= 108, "Incorrect␣message␣length");
```

CVF-21. INFO

- **Category** Procedural
- **Source** ZkSyncL2Gateway.sol

Description We didn’t review this file.

```
7 7 import {AddressAliasHelper} from "../../zksync/l1-contracts/vendor/
      ↪ AddressAliasHelper.sol";
```

CVF-22. INFO

- **Category** Bad datatype
- **Source** ZkSyncL2Gateway.sol

Recommendation The type for the “_zkLink” argument should be more specific.

Client Comment *The ‘_zkLink‘ only used to check whether the caller is permitted and does not use its interface, so we do not plan to modify it.*

```
26 26 constructor(address _zkLink) L2BaseGateway(_zkLink) {
```

CVF-23. FIXED

- **Category** Documentation
- **Source** ZkPolygonL1Gateway.sol

Description It is unclear what “default” means for a constant.

Recommendation Consider rephrasing or adding more details.

```
15 15 // Default to true
16 16 bool public constant FORCE_UPDATE_GLOBAL_EXIT_ROOT = true;
```

CVF-24. FIXED

- **Category** Documentation
- **Source** ZkPolygonL2Gateway.sol

Description It is unclear what “default” means for a constant.

Recommendation Consider rephrasing or adding more details.

```
14 14 // Default to true
15 15 bool public constant FORCE_UPDATE_GLOBAL_EXIT_ROOT = true;
```

CVF-25. INFO

- **Category** Bad datatype
- **Source** ZkPolygonL2Gateway.sol

Recommendation The type for the “_zkLink” argument should be more specific.

Client Comment *Same as CVF-22.*

```
23 23 constructor(address _zkLink, IZkPolygon _messageService)
      ↪ L2BaseGateway(_zkLink) {
```

CVF-26. FIXED

- **Category** Documentation
- **Source** ScrollL1Gateway.sol

Recommendation Consider adding a comment next to this argument with the argument name, otherwise it is unclear why the transaction origin address is used here.

```
36 36 tx.origin
```

CVF-27. INFO

- **Category** Bad datatype
- **Source** ScrollL2Gateway.sol

Recommendation The type for this argument should be more specific.

Client Comment *Same as CVF-22.*

```
11 11 address _zkLink,
```

CVF-28. INFO

- **Category** Bad datatype
- **Source** OptimismL2Gateway.sol

Recommendation The type for this argument should be more specific.

Client Comment *Same as CVF-22.*

```
11 11  address _zkLink
```

CVF-29. FIXED

- **Category** Procedural
- **Source** OptimismL2Gateway.sol

Recommendation The address of the optimism messenger should be a named constant or a constructor argument.

```
12 12  ) L2BaseGateway(_zkLink) OptimismGateway(IOptimismMessenger(0
          ↪ x4200000000000000000000000000000000000007)) {
```

CVF-30. INFO

- **Category** Bad datatype
- **Source** LineaL2Gateway.sol

Recommendation The type for this argument should be more specific.

Client Comment *Same as CVF-22.*

```
11 11  address _zkLink,
```

CVF-31. INFO

- **Category** Bad datatype
- **Source** EthereumGateway.sol

Recommendation The type for the "_zkLink" argument should be more specific.

Client Comment *Same as CVF-22.*

```
18 18  constructor(IArbitrator _arbitrator, address _zkLink) L1BaseGateway(
          ↪ _arbitrator) L2BaseGateway(_zkLink) {
```

CVF-32. INFO

- **Category** Procedural
- **Source** ArbitrumL1Gateway.sol

Description We didn't review these files.

```
4  4  import {Inbox, IBridge} from "@arbitrum/nitro-contracts/src/bridge/
   ↪  Inbox.sol";
5  5  import {IOutbox} from "@arbitrum/nitro-contracts/src/bridge/Outbox.
   ↪  sol";
```

CVF-33. INFO

- **Category** Procedural
- **Source** ArbitrumL2Gateway.sol

Description We didn't review this file.

```
4  4  import {ArbSys} from "@arbitrum/nitro-contracts/src/precompiles/
   ↪  ArbSys.sol";
```

CVF-34. INFO

- **Category** Procedural
- **Source** ArbitrumL2Gateway.sol

Description We didn't review this file.

```
6  6  import {AddressAliasHelper} from "../../zksync/l1-contracts/vendor/
   ↪  AddressAliasHelper.sol";
```

CVF-35. INFO

- **Category** Bad datatype
- **Source** ArbitrumL2Gateway.sol

Recommendation The argument type should be more specific.

Client Comment *Same as CVF-22.*

```
20 20  constructor(address _zkLink) L2BaseGateway(_zkLink) {
```

CVF-36. INFO

- **Category** Bad datatype
- **Source** BaseGateway.sol

Recommendation The type for this variable should be more specific.

Client Comment *Same as CVF-16.*

```
11 11  address internal remoteGateway;
```

CVF-37. FIXED

- **Category** Bad naming
- **Source** BaseGateway.sol

Recommendation Events are usually named via nouns, such as “RemoteGateway”.

```
20 20  event SetRemoteGateway(address remoteGateWay);
```

CVF-38. INFO

- **Category** Bad datatype
- **Source** BaseGateway.sol

Recommendation The parameter type should be more specific.

Client Comment *Same as CVF-16.*

```
20 20  event SetRemoteGateway(address remoteGateWay);
```

CVF-39. INFO

- **Category** Bad datatype
- **Source** BaseGateway.sol

Recommendation The return type should be more specific.

Client Comment *Same as CVF-16.*

```
34 34  function getRemoteGateway() external view returns (address) {
```

CVF-40. INFO

- **Category** Bad datatype
- **Source** BaseGateway.sol

Recommendation The argument type should be more specific.

Client Comment *Same as CVF-16.*

```
40 40  function setRemoteGateway(address _remoteGateway) external onlyOwner
          ↪  {
```

CVF-41. INFO

- **Category** Bad datatype
- **Source** L2BaseGateway.sol

Recommendation The type for this variable should be more specific.

Client Comment *Same as CVF-22.*

```
8 8  address public immutable ZKLINK;
```

CVF-42. INFO

- **Category** Bad naming
- **Source** L2BaseGateway.sol

Recommendation Events are usually named via nouns, such as “SentL2GatewayMessage”.

Client Comment *May affect third-party indexing services, we do not plan to modify it*

```
17 17  event L2GatewayMessageSent(uint256 value, bytes callData);
```

CVF-43. FIXED

- **Category** Unclear behavior
- **Source** L2BaseGateway.sol

Recommendation This event is never emitted in this contract. Probably it would be declared elsewhere, e.g. in the “IL2Gateway” interface or in the contract that actually emits it.

```
17 17  event L2GatewayMessageSent(uint256 value, bytes callData);
```

CVF-44. INFO

- **Category** Bad datatype
- **Source** L2BaseGateway.sol

Recommendation The argument type should be more specific.

Client Comment *Same as CVF-22.*

```
25 25 constructor(address _zkLink) {
```

CVF-45. INFO

- **Category** Procedural
- **Source** ZkLink.sol

Description We didn't review these files.

```
10 10 import {AddressAliasHelper} from "./zksync/l1-contracts/vendor/
      ↪ AddressAliasHelper.sol";
```

```
13 13 import {IMailbox, TxStatus} from "./zksync/l1-contracts/zksync/
      ↪ interfaces/IMailbox.sol";
14 14 import {IAdmin} from "./zksync/l1-contracts/zksync/interfaces/IAdmin
      ↪ .sol";
15 15 import {IZkSync} from "./zksync/l1-contracts/zksync/interfaces/
      ↪ IZkSync.sol";
16 16 import {Merkle} from "./zksync/l1-contracts/zksync/libraries/Merkle.
      ↪ sol";
17 17 import {TransactionValidator} from "./zksync/l1-contracts/zksync/
      ↪ libraries/TransactionValidator.sol";
18 18 import {L2Log, L2Message, PubdataPricingMode, FeeParams,
      ↪ SecondaryChainSyncStatus} from "./zksync/l1-contracts/zksync/
      ↪ Storage.sol";
19 19 import {UncheckedMath} from "./zksync/l1-contracts/common/libraries/
      ↪ UncheckedMath.sol";
20 20 import {UnsafeBytes} from "./zksync/l1-contracts/common/libraries/
      ↪ UnsafeBytes.sol";
21 21 import {REQUIRED_L2_GAS_PRICE_PER_PUBDATA, MAX_NEW_FACTORY_DEPS,
      ↪ L1_GAS_PER_PUBDATA_BYTE, L2_L1_LOGS_TREE_DEFAULT_LEAF_HASH}
      ↪ from "./zksync/l1-contracts/zksync/Config.sol";
22 22 import {L2_TO_L1_MESSENGER_SYSTEM_CONTRACT_ADDR,
      ↪ L2_BOOTLOADER_ADDRESS, L2_ETH_TOKEN_SYSTEM_CONTRACT_ADDR} from
      ↪  "./zksync/l1-contracts/common/L2ContractAddresses.sol";
23 23 import {IGetters} from "./zksync/l1-contracts/zksync/interfaces/
      ↪ IGetters.sol";
```

CVF-46. FIXED

- **Category** Procedural
- **Source** ZkLink.sol

Recommendation Consider specifying the typehash as an expression rather than a hard-coded hash value. Solidity compiler is smart enough to precompute hash expressions.

```
39 39 // keccak256("ForwardL2Request(address gateway,bool isContractCall,
      ↪ address sender,uint256 txId,address contractAddressL2,uint256
      ↪ l2Value,bytes32 l2CallDataHash,uint256 l2GasLimit,uint256
      ↪ l2GasPricePerPubdata,bytes32 factoryDepsHash,address
      ↪ refundRecipient)")
40 40 bytes32 public constant FORWARD_REQUEST_TYPE_HASH =
41 41     0
          ↪ xe0aaca1722ef50bb0c9b032e5b16ce2b79fa9f23638835456b27fd6894f8292c
          ↪ ;
```

CVF-47. FIXED

- **Category** Procedural
- **Source** ZkLink.sol

Recommendation The parameter should be indexed.

```
90 90 event InitGateway(IL2Gateway gateway);
```

CVF-48. FIXED

- **Category** Procedural
- **Source** ZkLink.sol

Recommendation The “contractAddress” parameter should be indexed.

```
92 92 event ContractAllowStatusUpdate(address contractAddress, bool
      ↪ isPermit);
```

CVF-49. INFO

- **Category** Suboptimal
- **Source** ZkLink.sol

Recommendation The old value parameters are redundant, as their values could be derived from previous events.

Client Comment *We do not plan to modify it to be consistent with similar events in the zksync era*

```
94  94  event TxGasPriceUpdate(uint256 oldTxGasPrice, uint256 newTxGasPrice)
        ↪ ;
```

```
114 114 event ForwardFeeAllocatorUpdate(address oldAllocator, address
        ↪ newAllocator);
```

CVF-50. FIXED

- **Category** Procedural
- **Source** ZkLink.sol

Recommendation The “validatorAddress” parameter should be indexed.

```
96  96  event ValidatorStatusUpdate(address validatorAddress, bool isActive)
        ↪ ;
```

CVF-51. FIXED

- **Category** Procedural
- **Source** ZkLink.sol

Recommendation The “receiver” parameter should be indexed.

```
108 108 event WithdrawForwardFee(address receiver, uint256 amount);
```

CVF-52. INFO

- **Category** Procedural
- **Source** ZkLink.sol

Recommendation The “to” parameter should be indexed.

Client Comment *‘to‘ is indexed.*

```
112 112 event EthWithdrawalFinalized(address indexed to, uint256 amount);
```

CVF-53. FIXED

- **Category** Procedural
- **Source** ZkLink.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
146 146 function _authorizeUpgrade(address newImplementation) internal
        ↪ override onlyOwner {}
```

CVF-54. FIXED

- **Category** Unclear behavior
- **Source** ZkLink.sol

Description These events are emitted even if nothing actually changed.

```
196 196 emit ContractAllowStatusUpdate(_contractAddress, _permitted);
203 203 emit TxGasPriceUpdate(oldTxGasPrice, _newTxGasPrice);
208 208 emit ValidatorStatusUpdate(_validator, _active);
219 219 emit NewFeeParams(oldFeeParams, _newFeeParams);
227 227 emit ForwardFeeAllocatorUpdate(oldAllocator, _newForwardFeeAllocator
        ↪ );
```

CVF-55. INFO

- **Category** Suboptimal
- **Source** ZkLink.sol

Description The storage slot address of "isEthWithdrawalFinalized[_l2BatchNumber][_l2MessageIndex]" is calculated twice.

Recommendation Consider refactoring to calculate it only once.

Client Comment *We do not plan to modify it and keep it consistent with the zksync era code.*

```
340 340 require(!isEthWithdrawalFinalized[_l2BatchNumber][_l2MessageIndex],
        ↪ "jj");
354 354 isEthWithdrawalFinalized[_l2BatchNumber][_l2MessageIndex] = true;
```

CVF-56. INFO

- **Category** Suboptimal
- **Source** ZkLink.sol

Recommendation Type conversion is redundant.

```
474 474 uint256 batchOverheadETH = uint256(_feeParams.batchOverheadL1Gas) *
        ↪ _l1GasPrice;
```

CVF-57. FIXED

- **Category** Bad datatype
- **Source** ZkLink.sol

Recommendation The value “108” should be a named constant.

```
558 558 require(_message.length == 108, "pm");
```

CVF-58. INFO

- **Category** Procedural
- **Source** Arbitrator.sol

Description We didn’t review these files.

```
11 11 import {IAdmin} from "./zksync/l1-contracts/zksync/interfaces/IAdmin
      ↪ .sol";
12 12 import {IZkSync} from "./zksync/l1-contracts/zksync/interfaces/
      ↪ IZkSync.sol";
13 13 import {FeeParams} from "./zksync/l1-contracts/zksync/Storage.sol";
```

CVF-59. FIXED

- **Category** Procedural
- **Source** Arbitrator.sol

Recommendation The “gateway” parameters should be indexed.

```
38 38 event InitPrimaryChain(IL1Gateway gateway);

40 40 event SecondaryChainStatusUpdate(IL1Gateway gateway, bool isActive);

44 44 event ValidatorStatusUpdate(IL1Gateway gateway, address
      ↪ validatorAddress, bool isActive);

46 46 event NewFeeParams(IL1Gateway gateway, FeeParams newFeeParams);

50 50 event MessageForwarded(IL1Gateway gateway, uint256 value, bytes
      ↪ callData);
```

CVF-60. FIXED

- **Category** Procedural
- **Source** Arbitrator.sol

Recommendation The “relayer” parameter should be indexed.

```
42 42 event RelayerStatusUpdate(address relayer, bool isActive);
```

CVF-61. FIXED

- **Category** Procedural
- **Source** Arbitrator.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
68 68 function _authorizeUpgrade(address newImplementation) internal
      ↪ override onlyOwner {}
```

CVF-62. FIXED

- **Category** Unclear behavior
- **Source** Arbitrator.sol

Description This event is emitted even if nothing actually changed.

```
103 103 emit RelayerStatusUpdate(_relayer, _active);
```

CVF-63. FIXED

- **Category** Suboptimal
- **Source** PriorityQueue.sol

Description The expression "_queue.head + i" is calculated twice.

Recommendation Consider calculating once and reusing.

```
65 +require(_queue.head + i < _queue.tail, "D"); // invalid index
```

```
67 +return _queue.data[_queue.head + i];
```

CVF-64. INFO

- **Category** Procedural
- **Source** IL2Gateway.sol

Description Specifying a particular compiler version makes it harder to upgrade to newer versions.

Recommendation Consider specifying as "^0.8.0". Also relevant for: IZkLink.sol

Client Comment *We do not plan to modify it and keep it consistent with the zksync era code.*

```
2 +pragma solidity 0.8.19;
```

CVF-65. INFO

- **Category** Bad datatype
- **Source** IAdmin.sol

Recommendation The type for the "_gateway" argument should be more specific.

Client Comment *The '_gateway' is only used as an address and does not use its interface, so we do not plan to modify it.*

```
21 +function setSecondaryChainGateway(address _gateway, bool _active)
   ↪ external;
```

```
73 +event SecondaryChainStatusUpdate(address indexed gateway, bool
   ↪ isActive);
```

CVF-66. INFO

- **Category** Bad datatype
- **Source** IGetters.sol

Recommendation The type of the “gateway” arguments should be “IL2Gateway”.

Client Comment *Same as CVF-65.*

```
22 +function getSecondaryChain(address gateway) external view returns (
      ↪ SecondaryChain memory);
```

```
26 +    address gateway,
```

CVF-67. INFO

- **Category** Bad datatype
- **Source** IMailbox.sol

Recommendation The type for the secondary chain gateway arguments should be more specific.

Client Comment *Same as CVF-65.*

```
107 +address gateway;
```

```
230 +address _secondaryChainGateway,
```

```
241 +address _secondaryChainGateway,
```

CVF-68. INFO

- **Category** Bad datatype
- **Source** IMailbox.sol

Recommendation The type for the secondary chain gateway parameters should be more specific.

Client Comment *Same as CVF-65.*

```
275 +    address secondaryChainGateway,
```

```
285 +event SyncBatchRoot(address secondaryChainGateway, uint256
      ↪ batchNumber, uint256 forwardEthAmount);
```

CVF-69. INFO

- **Category** Bad datatype
- **Source** Admin.sol

Recommendation The type for the "_gateway" argument should be more specific.

Client Comment *Same as CVF-65.*

```
30 +function setSecondaryChainGateway(address _gateway, bool _active)
      ↪ external onlyGateway {
```

CVF-70. FIXED

- **Category** Unclear behavior
- **Source** Admin.sol

Description This event is emitted even if nothing actually changed.

```
32 +emit SecondaryChainStatusUpdate(_gateway, _active);
```

CVF-71. INFO

- **Category** Bad datatype
- **Source** Getters.sol

Recommendation The type for the "gateway" arguments should be more "IL2Gateway".

Client Comment *Same as CVF-65.*

```
37 +function getSecondaryChain(address gateway) external view returns (
      ↪ SecondaryChain memory) {
```

```
43 +    address gateway,
```

CVF-72. FIXED

- **Category** Procedural
- **Source** Mailbox.sol

Recommendation Consider specifying the constant value as a hash expression rather than as a hardcoded hash value. Solidity compiler is smart enough to precompute hash expressions.

```
35 +// keccak256("ForwardL2Request(address gateway,bool isContractCall,
      ↪ address sender,uint256 txId,address contractAddressL2,uint256
      ↪ l2Value,bytes32 l2CallDataHash,uint256 l2GasLimit,uint256
      ↪ l2GasPricePerPubdata,bytes32 factoryDepsHash,address
      ↪ refundRecipient)")
36 +bytes32 public constant FORWARD_REQUEST_TYPE_HASH =
37 +0xe0aaca1722ef50bb0c9b032e5b16ce2b79fa9f23638835456b27fd6894f8292c;
```

CVF-73. INFO

- **Category** Bad datatype
- **Source** Mailbox.sol

Recommendation The type for these arguments should be more specific.

Client Comment *Same as CVF-65.*

```
151 +address _secondaryChainGateway,
```

```
191 +address _secondaryChainGateway,
```

CVF-74. INFO

- **Category** Suboptimal
- **Source** Mailbox.sol

Description The expression “s.secondaryChains[_secondaryChainGateway]” is calculated twice.

Recommendation Consider calculating once and reusing.

Client Comment *We do not plan to modify it.*

```
157 +SecondaryChain memory secondaryChain = s.secondaryChains[
      ↪ _secondaryChainGateway];
```

```
185 +s.secondaryChains[_secondaryChainGateway] = secondaryChain;
```

CVF-75. INFO

- **Category** Suboptimal
- **Source** Mailbox.sol

Description The expression "s.secondaryChains[_l2WithdrawReceiver]" is calculated several times.

Recommendation Consider calculating once and reusing.

Client Comment *We do not plan to modify it.*

```
276 +if (s.secondaryChains[_l1WithdrawReceiver].valid) {
277 +    s.secondaryChains[_l1WithdrawReceiver].totalPendingWithdraw =
278 +        s.secondaryChains[_l1WithdrawReceiver].totalPendingWithdraw
    ↪  +
```

CVF-76. FIXED

- **Category** Suboptimal
- **Source** Mailbox.sol

Description The storage slot address of "s.secondaryChains[_l1WithdrawReceiver].totalPendingWithdraw" is calculated twice.

Recommendation Consider using the "+=" operator to make the slot address to be reused.

```
277 +s.secondaryChains[_l1WithdrawReceiver].totalPendingWithdraw =
278 +    s.secondaryChains[_l1WithdrawReceiver].totalPendingWithdraw +
```

CVF-77. INFO

- **Category** Bad datatype
- **Source** Storage.sol

Recommendation The first key type for these mappings should be more specific.

Client Comment *Same as CVF-65.*

```
137 +mapping(address secondaryChainGateway => SecondaryChain)
    ↪ secondaryChains;
138 +mapping(address secondaryChainGateway => mapping(uint256
    ↪ secondaryChainPriorityOpId => SecondaryChainSyncStatus
    ↪ syncStatus)) secondaryChainSyncStatus;
```

CVF-78. INFO

- **Category** Suboptimal
- **Source** Storage.sol

Recommendation It would be more efficient to merge these two mappings into a single mapping whose keys are secondary chain gateways and values are structs of two fields encapsulating values of the original mappings.

Client Comment *We do not plan to modify it.*

```
137 +mapping(address secondaryChainGateway => SecondaryChain)
       ↪ secondaryChains;
138 +mapping(address secondaryChainGateway => mapping(uint256
       ↪ secondaryChainPriorityOpId => SecondaryChainSyncStatus
       ↪ syncStatus)) secondaryChainSyncStatus;
```

ABDK
Consulting

About us

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.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting