Report

v. 1.0

Customer

zkLink

Smart Contract Audit

Solidity

6th February 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	03.02.23	A. Zveryanskaya	Initial Draft
0.2	06.02.23	A. Zveryanskaya	Minor revision
1.0	07.02.23	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 a trading-focused multi-chain L2 network with unified liquidity secured by ZK-Rollups.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

/			
	DeployFactory.sol	Storage.sol	ZkLink.sol
	ZkLinkPeriphery.sol		
token/			
	IZKL.sol		
bridge/			
	ILayerZeroEndpoint.sol	ILayerZeroReceiver.sol	ILayerZeroUser ApplicationConfig.sol
	LayerZeroBridge.sol	LayerZeroStorage.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 1 critical, 20 major, and a few less important issues. All identified Critical issues have been fixed.

Severity	Active	Fixed
Critical	0	1
Major	8	12
Moderate	8	4
Minor	42	41

Fixed 58 out of 116 issues

6 Critical Issues

CVF-41. FIXED

- **Category** Overflow/Underflow
- **Source** ZkLinkPeriphery.sol

Description Overflow is possible here.

Recommendation Consider performing calculations in 256-bit numbers and using safe math.

```
382 amountReceive = amount * (MAX_ACCEPT_FEE_RATE - withdrawFeeRate) /
    ↪ MAX_ACCEPT_FEE_RATE;
```

7 Major Issues

CVF-3. INFO

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

Recommendation This variable should be declared as immutable and should be set in the constructor, rather than in the "initialize" and "upgrade" functions. This would make the whole schema more efficient and less error-prone.

Client Comment *Not a problem, we want individually upgrade periphery logic contract.*

```
address public periphery;
```

CVF-4. FIXED

- **Category** Documentation
- **Source** Storage.sol

Description This comment is incorrect, as "self" is an immutable variable, and immutable variables don't occupy storage space.

```
// self(20 bytes) + totalBlocksSynchronized(4 bytes) + exodusMode(1
   ↪ bytes) stored in the same slot
```

CVF-29. FIXED

- **Category** Suboptimal
- **Source** ZkLinkPeriphery.sol

Recommendation It would be more efficient to pass a single array of structs with five fields, rather than five parallel arrays. This would also make the length checks unnecessary.

```
function addTokens(uint16[] calldata _tokenIdList, address[]
   ↪ calldata _tokenAddressList, uint8[] calldata _decimalsList,
   ↪ bool[] calldata _standardList, uint16[] calldata
   ↪ _mappingTokenList) external {
```

CVF-31. FIXED

- **Category** Suboptimal
- **Source** ZkLinkPeriphery.sol

Description Here the whole “RegisteredToken” structure is read from the storage and then is written back, while only a few fields are actually accessed.

Recommendation Consider using a storage reference to read and update fields in place.

```
RegisteredToken memory rt = tokens[_tokenId];
```

```
    tokens[_tokenId] = rt;
```

CVF-40. INFO

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

Recommendation Should be “<=”.

Client Comment *Not a problem, feeRate of 100% is impossible.*

```
require(withdrawFeeRate < MAX_ACCEPT_FEE_RATE, "H4");
```

CVF-42. INFO

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

Description Ethereum chainID is not guaranteed to fit into 16 bits.

Recommendation Consider using a wider type.

Client Comment *Not a problem, chainId is defined by LayerZero , please see https://layerzero.gitbook.io/docs/technical-reference/mainnet/supported-chain-ids.*

```
function setConfig(uint16 _version, uint16 _chainId, uint
    ↪ _configType, bytes calldata _config) external;
```

```
function forceResumeReceive(uint16 _srcChainId, bytes calldata
    ↪ _srcAddress) external;
```

CVF-45. FIXED

- **Category** Documentation
- **Source** ZkLink.sol

Description The parameters described here don't match with the actual structure fields below.

Recommendation Consider describing the actual fields.

```
/// @param commitmentsInSlot verified commitments in one slot
/// @param commitmentIdx index such that commitmentsInSlot[
   ↪ commitmentIdx] is current block commitment
```

CVF-50. FIXED

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

Description Three external calls are performed here which is redundant for most tokens.

Recommendation Consider maintaining a flag per token telling whether for the token the actual transferred amount could differ from the requested amount, and do additional balance calculations only for those tokens that actually need them.

```
uint256 balanceBefore = _token.balanceOf(address(this));
```

```
_token.transferFrom(msg.sender, address(this), _amount);
uint256 balanceAfter = _token.balanceOf(address(this));
```

CVF-52. FIXED

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

Description Here the whole “RegisteredToken” structure is read into the memory, while only a few fields are actually used.

Recommendation Consider changing “memory” to “storage” here to avoid redundant storage reads.

```
RegisteredToken memory rt = tokens[_tokenId];
```

```
RegisteredToken memory rt = tokens[_tokenId];
```

```
RegisteredToken memory rt = tokens[tokenId];
```

```
RegisteredToken memory rt = tokens[op.tokenId];
```

```
RegisteredToken memory rt = tokens[op.tokenId];
```

```
RegisteredToken memory rt = tokens[op.tokenId];
```

CVF-57. INFO

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

Description The “amount” value logged here could differ from both, the amount debited from the contract and the amount credited to the user.

Recommendation Consider always logging the amount actually debited from the contract.

Client Comment *Not a problem, transferFromERC20 will return the actual debited amount from the contract.*

```
emit Withdrawal(_tokenId, amount);
```

CVF-59. FIXED

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

Description This function should emit some event.

```
function proveBlocks(StoredBlockInfo[] memory _committedBlocks,
   ↪ ProofInput memory _proof) external nonReentrant {
```

CVF-62. INFO

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

Description This ignores higher bits when comparing commitments.

Recommendation Consider explicitly requiring the higher bits to be zero.

Client Comment *Not a problem, the higher 3 bits are erased because the max number bits of circuit can represent is 253, and it’s safe enough to avoid hash conflict.*

```
require(_proof.commitments[i] & INPUT_MASK == uint256(
   ↪ _committedBlocks[i].commitment) & INPUT_MASK, "x1");
```

CVF-67. FIXED

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

Recommendation Consider elaborating more regarding why overflow is not possible. Even if it is not possible due to some business-logic constraints enforced in different part of the code, it would still be better to use safe addition here.

```
    // overflow is impossible
```

```
firstPriorityRequestId += priorityRequestsExecuted;
totalCommittedPriorityRequests -= priorityRequestsExecuted;
totalOpenPriorityRequests -= priorityRequestsExecuted;
```

```
totalBlocksExecuted += nBlocks;
```

CVF-69. FIXED

- **Category** Flaw
- **Source** ZkLink.sol

Description In case a zero tokenId would ever be registered, all token addresses will become registered as well.

Recommendation Consider explicitly checking that “tokenId” is not zero.

377
```
uint16 tokenId = tokenIds[_tokenAddress];
```

CVF-72. FIXED

- **Category** Overflow/Underflow
- **Source** ZkLink.sol

Description Overflow is possible here. Consider using safe addition and safe conversion.

419
```
uint64 expirationBlock = uint64(block.number + PRIORITY_EXPIRATION);
```

434
```
totalOpenPriorityRequests++;
```

CVF-73. FIXED

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

Recommendation Even if overflow is impossible due to business-logic constraints, it would still be better to use safe addition to prevent complicated attacks that use several vulnerabilities.

421
```
// overflow is impossible
uint64 nextPriorityRequestId = firstPriorityRequestId +
   ↪ totalOpenPriorityRequests;
```

447
```
    // overflow is impossible
    totalCommittedPriorityRequests += _lastCommittedBlockData.
       ↪ priorityOperations;
```

521
```
// overflow is impossible
uint64 uncommittedPriorityRequestsOffset = firstPriorityRequestId +
   ↪ totalCommittedPriorityRequests;
```

CVF-78. INFO

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

Recommendation It would be enough to allocate MAX_CHAIN_ID - MIN_CHAIN_ID + 1 elements.

Client Comment *Not a problem, we use index of the array to represent chain id.*

```
onchainOperationPubdataHashs = new bytes32[](MAX_CHAIN_ID + 1); // overflow is impossible
```

CVF-82. INFO

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

Recommendation No data copying is needed here, just copy a reference: processablePubData = opPubData;

Client Comment *Not a problem, because 'opPubData' and 'processablePubData' will be consumed in later concatHash.*

```
processablePubData = Bytes.slice(opPubData, 0, opPubData.length);
```

CVF-109. FIXED

- **Category** Flaw
- **Source** LayerZeroBridge.sol

Description The length of the "contractAddr" argument is not checked against the corresponding destAddressLength value.

Recommendation Consider adding such a check.

```
function setDestination(uint16 dstChainId, bytes calldata contractAddr) external onlyGovernor {
```

CVF-110. INFO

- **Category** Suboptimal
- **Source** LayerZeroBridge.sol

Description There could be blockchains with variable address length, such as bitcoin.

Recommendation Consider somehow supporting such blockchains.

Client Comment *Not a problem, the type of 'contractAddr' is bytes that can support mutlichains.*

```
67 /// @notice Set destination address length
```

8 Moderate Issues

CVF-18. INFO

- **Category** Overflow/Underflow
- **Source** Storage.sol

Description Overflow here may make it impossible th perform a priority operation and thus trigger exodus mode.

Recommendation Consider supporting 256-bits pending balances.

Client Comment *Not a problem, pending amount should not exceed uint128.max and entering exdous mode is expected.*

```
pendingBalances[_packedBalanceKey] = balance.add(_amount);
```

CVF-37. INFO

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

Description There is no check to ensure that tokenId is not zero.

Recommendation Consider adding such a check.

Client Comment *Not a problem, tokenId will be checked later in function '_checkAccept'.*

```
uint16 tokenId = tokenIds[ETH_ADDRESS];
```

CVF-51. INFO

- **Category** Flaw
- **Source** ZkLink.sol

Description The returned value is ignored.

Recommendation Consider explicitly requiring the returned value to be true.

Client Comment *Not a problem, there may be no return value of 'transferFrom'.*

```
_token.transferFrom(msg.sender, address(this), _amount);
```

CVF-54. FIXED

- **Category** Flaw
- **Source** ZkLink.sol

Description This check actually makes DDoS attacks possible, as a malicious user may flood the contract with lots of priority requests effectively preventing normal users from being able to exit. Such possibility could deteriorate trust in the protocol.

Recommendation Consider preventing DDoS attacks in other ways, e.g. by linearly increasing the price of subsequent priority requests, to make a DDoS attack to cost O(n^2), rather than O(n).

Client Comment *Fix, we remove this check, protocol trust is first.*

```
172 require(totalOpenPriorityRequests < MAX_PRIORITY_REQUESTS, "a4");
```

```
393 require(totalOpenPriorityRequests < MAX_PRIORITY_REQUESTS, "e6");
```

CVF-58. INFO

- **Category** Flaw
- **Source** ZkLink.sol

Description The returned value is ignored.

Recommendation Consider explicitly requiring that the returned value is true.

Client Comment *Not a problem, there may be no return value of 'transfer'.*

```
244 _token.transfer(_to, _amount);
```

```
248 _token.transfer(_to, _amount);
```

CVF-61. FIXED

- **Category** Overflow/Underflow
- **Source** ZkLink.sol

Description Overflow is possible here.

```
293 ++currentTotalBlocksProven;
```

CVF-66. FIXED

- **Category** Overflow/Underflow
- **Source** ZkLink.sol

Description Overflow is possible here. It should never happen in case there are no bugs in the protocol, however it would still be better to use safe addition here.

```
revertedPriorityRequests += storedBlockInfo.priorityOperations;
```

CVF-77. FIXED

- **Category** Overflow/Underflow
- **Source** ZkLink.sol

Description Overflow is possible here.

Recommendation Consider using safe math.

```
require(pubdataOffset + 1 < pubData.length, "h1");
```

```
uint64 nextPriorityOpIndex = uncommittedPriorityRequestsOffset +
   ↪ priorityOperationsProcessed;
```

CVF-101. INFO

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

Description It is unclear how the ZRO token holder authorizes the transaction. Simple "approve" wouldn't work here, as it would allow anyone to use approved ZRO tokens to pay for a transaction.

Client Comment *Not a problem, user must approve ZRO to LayerZero protocol if they want to pay lz protocol fee in ZRO token, please see https://github.com/LayerZero-Labs/LayerZero/blob/3fb8f6962c1346eefa7e12f2cd8c299f0cfba944/contracts/Ultra-LightNodeV2.sol#L196.*

```
// @param _zroPaymentAddress - the address of the ZRO token holder
   ↪ who would pay for the transaction
```

CVF-111. INFO

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

Description It is unclear how the “zroPaymentAddress” owner authorizes the transaction. Simple “approve” call is not enough as anyone would be able to use the approved tokens to pay transaction fees.

Client Comment *Same as 101.*

```
ILayerZeroEndpoint(endpoint).send{value:msg.value}(_dstChainId,
    ↪ trustedRemote, payload, params.refundAddress, params.
    ↪ zroPaymentAddress, params.adapterParams);
```

CVF-113. INFO

- **Category** Flaw
- **Source** LayerZeroBridge.sol

Description There is no nonce check, so it is possible to overwrite an already stored failed message by specifying the same nonce again.

Client Comment *Not a problem, nonce will be checked in LayerZero protocol, please see https://github.com/LayerZero-Labs/LayerZero/blob/3fb8f6962c1346eefa7e12f2cd8c299f0cfba944/contracts/Endpoint.sol#L102.*

```
failedMessages[srcChainId][srcAddress][nonce] = keccak256(payload);
```

CVF-114. INFO

- **Category** Suboptimal
- **Source** LayerZeroBridge.sol

Description The nonce value is used only in event parameters. There is no actual nonce check.

Recommendation Consider either adding nonce check or removing nonce.

Client Comment *Same as 113.*

```
function _nonblockingLzReceive(uint16 srcChainId, bytes calldata /**
    ↪ srcAddress**/, uint64 nonce, bytes calldata payload) internal
    ↪ {
```

9 Minor Issues

CVF-1. INFO

- **Category** Procedural
- **Source** Storage.sol

Recommendation We didn't review these files.

Client Comment *These files are copyed from zkSync and all have been audited.*

```
import "./zksync/Operations.sol";
import "./zksync/SafeMath.sol";
import "./zksync/SafeMathUInt128.sol";
import "./zksync/Config.sol";
import "./zksync/Verifier.sol";
```

CVF-2. INFO

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

Recommendation The type of this variable should bbe "ZkLinkPeriphery" or an interface extracted from it.

Client Comment *Not a problem, this variable is just a logic contract address, which only be used in deletegatecall.*

```
address public periphery;
```

CVF-5. FIXED

- **Category** Documentation
- **Source** Storage.sol

Description It is unclear what is the difference between "tokenId" and "srcTokenId".

Recommendation Consider explaining.

```
/// @notice Flag indicates that a user has exited in the exodus mode
    ↪ certain token balance (accountId => subAccountId => tokenId
    ↪ => srcTokenId)
```

CVF-6. INFO

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

Recommendation It would be more efficient to merge these two maps into a single map whose keys are address and nonce, and value are structs of two fields encapsulating the values of the original maps.

Client Comment *Not a problem, the usage of 'authFacts' is much more higher than 'authFactsResetTimer'.*

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

```
mapping(address => mapping(uint32 => uint256)) internal
    ↪ authFactsResetTimer;
```

CVF-7. FIXED

- **Category** Documentation
- **Source** Storage.sol

Description The semantics of keys and values in this mapping is unclear.

Recommendation Consider documenting.

```
mapping(uint16 => mapping(address => mapping(address => uint128)))
    ↪ internal brokerAllowances;
```

CVF-8. FIXED

- **Category** Documentation
- **Source** Storage.sol

Description Despite the comment, this is not a list but rather a set.

Recommendation Consider rephrasing.

```
/// @notice List of permitted validators
mapping(address => bool) public validators;
```

CVF-9. INFO

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

Recommendation The type of this field should be "IERC20".

Client Comment *Not a problem, 'tokenAddress' may be ETH_ADDRESS which represent deposit or withdraw ETH.*

```
address tokenAddress; // the token address
```

CVF-10. FIXED

- **Category** Procedural
- **Source** Storage.sol

Description In ERC-20 the "decimals" property is used by UI to render token amounts in a human-friendly way. Using this property in smart contracts is discouraged.

Recommendation Consider treating all token amounts as integers.

```
uint8 decimals; // the token decimals of layer one
```

CVF-11. FIXED

- **Category** Documentation
- **Source** Storage.sol

Description It is unclear what is a standard token.

Recommendation Consider explaining.

```
bool standard; // if a standard token
```

CVF-12. FIXED

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

Description It is unclear what is a mapping token, and the example doesn't help.

Recommendation Consider elaborating more on this.

```
uint16 mappingTokenId; // eg. USDC -> USD, zero means no mapping token
```

CVF-13. INFO

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

Recommendation The key type should be "IERC20".

Client Comment *Same as 9.*

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

CVF-14. INFO

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

Recommendation The type of this field should be "LayerZeroBridge" or an interface extracted from it.

Client Comment *Not a problem, 'bridge' here is similar 'owner' who has a special authority to call function.*

```
address bridge;
```

CVF-15. INFO

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

Recommendation The key type should be “LayerZeroBridge” or an interface extracted from it.

Client Comment *Same as 14.*

```
mapping(address => uint256) public bridgeIndex;
```

CVF-16. INFO

- **Category** Documentation
- **Source** Storage.sol

Recommendation Should be “block.timestamp predefined variable”.

Client Comment *Not a problem.*

```
uint256 timestamp; // Rollup block timestamp, have the same format as Ethereum block constant
```

CVF-17. INFO

- **Category** Procedural
- **Source** Storage.sol

Description Here a “StoredBlockInfo” struct is repacked in memory before hashing.

Recommendation Consider hashing in-place using an assembly block.

Client Comment *Not a problem, keep same with circuit.*

```
return keccak256(abi.encode(_storedBlockInfo));
```

CVF-19. FIXED

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

Recommendation This variable is redundant. Just use 0×0 memory offset instead, as this function anyway either reverts of terminates the transaction, so it doesn't need to care about preserving memory contents.

```
let ptr := mload(0x40)
```

CVF-20. FIXED

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

Recommendation This variable is redundant, as the "RETURNDATASIZE" opcode is cheaper than an access to a local variable.

```
let size := returndatasize()
```

CVF-21. INFO

- **Category** Procedural
- **Source** ZkLinkPeriphery.sol

Description We didn't review these files.

Client Comment *These files are copyed from zkSync and all have been audited.*

```
import "./zksync/ReentrancyGuard.sol";
import "./zksync/Events.sol";
```

```
import "./zksync/Bytes.sol";
import "./zksync/Utils.sol";
import "./zksync/SafeMath.sol";
import "./zksync/SafeCast.sol";
import "./zksync/IERC20.sol";
```

CVF-22. INFO

- **Category** Suboptimal
- **Source** ZkLinkPeriphery.sol

Description This should be emitted only if exodus mode has not been activated yet.

Client Comment *Not a problem, because there is a 'active' modifier applied to function 'activateExodusMode'.*

```
emit ExodusMode();
```

CVF-23. FIXED

- **Category** Suboptimal
- **Source** ZkLinkPeriphery.sol

Description The expression "firstPriorityRequestId + toProcess" is calculated on every loop iteration.

Recommendation Consider calculating once before the loop.

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

CVF-24. FIXED

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

Description This operation deletes every request, not only Deposit.

Recommendation Consider explaining in the documentation why deleting other requests is okay.

```
delete priorityRequests[id];
```

CVF-25. INFO

- **Category** Suboptimal
- **Source** ZkLinkPeriphery.sol

Recommendation As the length of a "_pubKeyHash" value must always be 20, consider using the "bytes20" type instead of "bytes".

Client Comment *Not a problem, if '_pubkeyHash' is bytes20 and then we need to use abi.encodePacked to convert it to bytes when call keccak256.*

```
function setAuthPubkeyHash(bytes calldata _pubkeyHash, uint32 _nonce ) external active nonReentrant {
```

CVF-26. FIXED

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

Description Some event should be emitted in this case.

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

CVF-27. FIXED

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

Description This check is redundant as it is anyway possible to set a dead governor address.

```
require(_newGovernor != address(0), "H");
```

CVF-28. INFO

- **Category** Procedural
- **Source** ZkLinkPeriphery.sol

Description In ERC-20 the “decimals” property is used by UI to render token amounts in a human-readable way. Using this property in smart contracts is discouraged.

Recommendation Consider treating all token amounts as integers.

Client Comment *Not a problem, token may has different decimals in chains, e.g. USDC decimals is 6 in Ethereum, but 18 in BSC. We define token decimals is 18 in l2, so we need to imcrease decimals when deposit and decrease decimals when withdraw.*

```
143 function addToken(uint16 _tokenId, address _tokenAddress, uint8
      ↪ _decimals, bool _standard, uint16 _mappingTokenId) public
      ↪ onlyGovernor {
```

CVF-30. FIXED

- **Category** Suboptimal
- **Source** ZkLinkPeriphery.sol

Description In case other arrays are longer than the “_tokenIdList” array, the remaining parts of the other arrays are ignored.

Recommendation Consider explicitly requiring all the arrays to be of the same length.

```
172 for (uint i; i < _tokenIdList.length; i++) {
```

CVF-32. FIXED

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

Description This function should return the index of the new bridge.

```
203 function addBridge(address bridge) external onlyGovernor {
```

CVF-33. INFO

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

Description Here a newly added bridge is immediately enabled for both, incoming and outgoing transfers.

Recommendation Consider implementing an ability to add a bridge in a not “all enabled” state.

Client Comment *Not a problem, when a new bridge is consider to used by our protocol, it should be ready for inbound and outbound messages.*

```
enableBridgeTo: true,
enableBridgeFrom: true
```

CVF-34. FIXED

- **Category** Suboptimal
- **Source** ZkLinkPeriphery.sol

Recommendation This event should include the index of the new bridge.

```
emit AddBridge(bridge);
```

CVF-35. FIXED

- **Category** Suboptimal
- **Source** ZkLinkPeriphery.sol

Description The checks “info.bridge == bridge” are redundant.

Recommendation Consider removing them. This would also allow reading only a single field from a “BridgeInfo” structure.

```
return info.bridge == bridge && info.enableBridgeTo;
```

```
return info.bridge == bridge && info.enableBridgeFrom;
```

CVF-36. INFO

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

Description This allows a bridge to update synchronization for any chain.

Recommendation Consider implementing a more fine-grained access control where each bridge is associated with a set of chains the bridge is allows to update synchronization for.

Client Comment *Not a problem, a bridge can send message to zkLink contract in all chains.*

```
require(isBridgeFromEnabled(msg.sender), "C");
```

CVF-38. INFO

- **Category** Suboptimal
- **Source** ZkLinkPeriphery.sol

Description The storage slot of the broker allowance is calculated twice.

Recommendation Consider refactoring to calculate it only once.

Client Comment *Not a problem.*

```
require(brokerAllowance(tokenId, accepter, msg.sender) >= amountSent
   ↪ , "F1");
brokerAllowances[tokenId][accepter][msg.sender] -= amountSent;
```

CVF-39. INFO

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

Description This function always returns true.

Recommendation Consider returning nothing,

Client Comment *Not a problem, similar to 'approve' of ERC20.*

```
function brokerApprove(uint16 tokenId, address spender, uint128
   ↪ amount) external returns (bool) {
```

CVF-43. INFO

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

Description These functions should emit some events, and these events should be declared in this interface.

Client Comment *Not a problem, please see https://layerzero.gitbook.io/docs/evm-uides/interfaces/evm-solidity-interfaces.*

```
function setConfig(uint16 _version, uint16 _chainId, uint
   ↪ _configType, bytes calldata _config) external;
```

```
function setSendVersion(uint16 _version) external;
```

```
function setReceiveVersion(uint16 _version) external;
```

```
function forceResumeReceive(uint16 _srcChainId, bytes calldata
   ↪ _srcAddress) external;
```

CVF-44. INFO

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

Description We didn't review these files.

Client Comment *These files are copyed from zkSync and all have been audited.*

```
import "./zksync/ReentrancyGuard.sol";
```

```
import "./zksync/Events.sol";
import "./zksync/UpgradeableMaster.sol";
import "./zksync/SafeMath.sol";
import "./zksync/SafeMathUInt128.sol";
import "./zksync/SafeCast.sol";
import "./zksync/Utils.sol";
import "./zksync/IERC20.sol";
```

CVF-46. INFO

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

Description There is no explicit check to prevent this function from being called several times.

Recommendation Consider adding such a check.

Client Comment *Not a problem, 'initializeReentrancyGuard' will prevent 'initialize' to be called serveral times.*

```
function initialize(bytes calldata initializationParameters) external onlyDelegateCall {
```

CVF-47. FIXED

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

Description This function is redundant, as a payable fallback function is already implemented.

Recommendation Consider removing this function.

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

CVF-48. FIXED

- **Category** Readability
- **Source** ZkLink.sol

Recommendation It is a good practice to put a comment with the argument name next to boolean literals passed as arguments. This would improve code readability.

```
deposit(ETH_ADDRESS, SafeCast.toUint128(msg.value), _zkLinkAddress, _subAccountId, false);
```

CVF-49. FIXED

- **Category** Documentation
- **Source** ZkLink.sol

Description This comment is confusing. The “token” argument type is “IERC20” and the code deals with “token” as if it were ERC-20 token, while ERC-1155 tokens are not backward compatible with ERC-20, so ERC-1155 token cannot be used with this function.

```
/// when the token(eg. erc777,erc1155) is not a pure erc20 token
```

CVF-53. FIXED

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

Recommendation This assignment should be made in an “else” branch of the conditional statement below.

```
uint16 srcTokenId = _tokenId;
```

CVF-55. FIXED

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

Description As the actual withdrawn amount could differ from the “_amount” argument value, this function should return the actual amount withdrawn.

```
function withdrawPendingBalance(address payable _owner, uint16
   ↪ _tokenId, uint128 _amount) external nonReentrant {
```

CVF-56. FIXED

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

Description The pending balance is potentially being updated twice.

Recommendation Consider refactoring to update it at most once.

```
pendingBalances[packedBalanceKey] = balance - amount; // amount <= balance
```

```
        pendingBalances[packedBalanceKey] = balance - amount1; // amount1 <= balance
```

CVF-60. FIXED

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

Recommendation These two lines could be merged into one: require(hashStoredBlockInfo(_committedBlocks[i]) == storedBlockHashes[++currentTotalBlocksProven], ”x0”);

```
require(hashStoredBlockInfo(_committedBlocks[i]) == storedBlockHashes[currentTotalBlocksProven + 1], "x0");
++currentTotalBlocksProven;
```

CVF-63. INFO

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

Description The mask is redundant here, since it does not guarantee that the result fits the field.

Recommendation Consider removing this operation and replacing it (maybe elsewhere) with a field check.

Client Comment *Same as 62.*

```
require(_proof.commitments[i] & INPUT_MASK == uint256(_committedBlocks[i].commitment) & INPUT_MASK, "x1");
```

CVF-64. FIXED

- **Category** Overflow/Underflow
- **Source** ZkLink.sol

Description Overflow is possible here.

Recommendation Consider using safe conversion.

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

CVF-65. FIXED

- **Category** Overflow/Underflow
- **Source** ZkLink.sol

Description Underflow is possible during subtraction. It should never happen in case the protocol is in a consistent state, however it would still be better to use safe subtraction here.

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

CVF-68. FIXED

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

Recommendation This check should be done at the beginning of the function before actually executing any blocks.

```
require(totalBlocksExecuted <= totalBlocksSynchronized, "d1");
```

CVF-70. FIXED

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

Description This assignment should be made only if “mapping” is false.

```
uint16 targetTokenId = tokenId;
```

CVF-71. INFO

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

Description In ERC-20 the "decimals" property is used by UI to render token amounts in a human-friendly way. Using this property in smart contract is discouraged.

Recommendation Consider treating all token amounts as integers.

Client Comment *Same as 28.*

```
// improve decimals before send to layer two
_amount = improveDecimals(_amount, rt.decimals);
```

```
uint128 amount = recoveryDecimals(op.amount, rt.decimals);
```

```
uint128 amount = recoveryDecimals(op.amount, rt.decimals);
```

```
uint128 amount = recoveryDecimals(op.amount, rt.decimals);
```

CVF-74. INFO

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

Description Here a storage variable is updated on each loop iteration.

Recommendation Consider refactoring to update once after the loop.

Client Comment *Not a problem, forward 'totalCommittedPriorityRequests' because it's will be reused in the next 'commitOneBlock'.*

```
totalCommittedPriorityRequests += _lastCommittedBlockData.
    ↪ priorityOperations;
```

CVF-75. INFO

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

Recommendation This check could be simplified as: require (_newBlock.timestamp - block.timestamp + COMMIT_TIMESTAMP_NOT_OLDER <= COMMIT_TIMESTAMP_NOT_OLDER + COMMIT_TIMESTAMP_APPROXIMATION_DELTA);

Client Comment *Not a problem, current code is easier to understand.*

```
require(block.timestamp.sub(COMMIT_TIMESTAMP_NOT_OLDER) <= _newBlock
   ↪ .timestamp &&
    _newBlock.timestamp <= block.timestamp.add(
       ↪ COMMIT_TIMESTAMP_APPROXIMATION_DELTA), "g3");
```

CVF-76. INFO

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

Description Performing this check on every loop iteration is suboptimal.

Recommendation Consider splitting into two loops: one from MIN_CHAIN_ID to CHAIN_ID - 1 and another from CHAIN_ID + 1 to MAX_CHAIN_ID. Alternatively, consider copying all elements including the element for the current chain, and then restoring the hash for the current chain using a value, cached before the loop.

Client Comment *Not a problem, current code is easier to understand.*

```
if (i != CHAIN_ID) {
```

CVF-79. INFO

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

Recommendation It would be cheaper to update the “chainIndex” incrementally like this: chainIndex «= 1;

Client Comment *Not a problem, we need to calculate the index of chain id in ’ALL_CHAINS’.*

```
uint256 chainIndex = 1 << i - 1; // overflow is impossible
```

```
uint256 chainIndex = 1 << i - 1; // overflow is impossible
```

CVF-80. INFO

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

Recommendation It would be cheaper to do: uint256 remainingChains = ALL_CHAINS » (MIN_CHAIN_ID - 1); for (i = MIN_CHAIN_ID; remainingChains != 0; i++) { if (remainingChains & 0×1 != 0) {...} remainingChains »= 1; }

Client Comment *Not a problem*

```
if (chainIndex & ALL_CHAINS == chainIndex) {
```

```
if (chainIndex & ALL_CHAINS == chainIndex) {
```

CVF-81. INFO

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

Recommendation Using zero instead as the initial value instead of an empty string hash, would make this function unnecessary.

Client Comment *Not a problem, keep same with circuit.*

```
onchainOperationPubdataHashs[i] = EMPTY_STRING_KECCAK;
```

CVF-83. INFO

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

Description The “abi.encodePacked” function is able to concatenate narrow data type (shorter than 32 bytes) without padding, however, by converting values to “uint256”, this functionality is not used.

Recommendation Consider removing convertions to reduce the number of bytes hashed.

Client Comment *Not a problem, keep same with circuit.*

```
commitment = sha256(abi.encodePacked(
        uint256(_newBlockData.blockNumber),
        uint256(_newBlockData.feeAccount),
```

```
        uint256(_newBlockData.timestamp),
```

CVF-84. FIXED

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

Description The function usually returns false on failed verification.

Recommendation Consider returning false here as well.

663
```
revert("l");
```

CVF-85. FIXED

- **Category** Readability
- **Source** ZkLink.sol

Description The fact that the nonce always increases is asserted only in circuits.

Recommendation Consider making this fact explicit in the documentation to make the code more readable.

696
```
// This type of change pubkey can be done only once
return recoveredAddress == _changePk.owner && _changePk.nonce == 0;
```

CVF-86. INFO

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

Description This line appears in the code twice.

Recommendation Consider refactoring to avoid code duplication.

Client Comment *Not a problem.*

754
```
    withdrawOrStore(op.tokenId, rt.tokenAddress, rt.standard, op.
       ↪ owner, amount);
```

760
```
withdrawOrStore(op.tokenId, rt.tokenAddress, rt.standard, op.owner,
   ↪ amount);
```

CVF-87. INFO

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

Recommendation An actual address might be more useful here.

Client Comment *Not a problem, tokenId cost less gas than tokenAddress in log.*

```
emit Withdrawal(_tokenId, _amount);
```

CVF-88. INFO

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

Recommendation The decimals factor could be precomputed for a tokens. No need to calculate it on every deposit.

Client Comment *Not a problem, current code is easier to understand.*

```
return _amount.mul(SafeCast.toUint128(10**(TOKEN_DECIMALS_OF_LAYER2
   ↪ - _decimals)));
```

CVF-89. FIXED

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

Description Rounding down here may drop some dust.

Recommendation Consider leaving the dust at the account.

```
return _amount.div(SafeCast.toUint128(10**(TOKEN_DECIMALS_OF_LAYER2
   ↪ - _decimals)));
```

CVF-90. INFO

- **Category** Procedural
- **Source** DeployFactory.sol

Description We didn’t review these files.

Client Comment *These files are copyed from zkSync and all have been audited.*

```
import "./zksync/Proxy.sol";
import "./zksync/UpgradeGatekeeper.sol";
import "./zksync/Verifier.sol";
```

CVF-91. FIXED

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

Recommendation Thew type of the “_periphery” argument would be “ZkLinkPeriphery” or an interface extracted from it.

```
constructor(Verifier _verifierTarget, ZkLink _zkLinkTarget, address
   ↪ _periphery, uint32 _blockNumber, uint256 _timestamp, bytes32
   ↪ _stateHash, bytes32 _commitment, bytes32 _syncHash, address
   ↪ _firstValidator, address _governor, address _feeAccountAddress
   ↪ ) {
```

CVF-92. FIXED

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

Recommendation The parameter types should be more specific.

```
event Addresses(address verifier, address zkLink, address gatekeeper
   ↪ );
```

CVF-93. FIXED

- **Category** Suboptimal
- **Source** DeployFactory.sol

Description The expression “ZkLinkPeriphery(address(zkLink))” is coded twice.

Recommendation Consider refactoring to avoid code duplication.

63
```
ZkLinkPeriphery(address(zkLink)).setValidator(_validator, true);
ZkLinkPeriphery(address(zkLink)).changeGovernor(_governor);
```

CVF-94. INFO

- **Category** Suboptimal
- **Source** LayerZeroStorage.sol

Recommendation These variables should be declared as immutable.

Client Comment *Not a problem, they can be initialized only once.*

17
```
address public networkGovernor;
```

19
```
address public endpoint;
```

CVF-95. FIXED

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

Recommendation The type of this variable should be “ILayerZeroEndpoint”.

19
```
address public endpoint;
```

CVF-96. FIXED

- **Category** Documentation
- **Source** LayerZeroStorage.sol

Description The semantics of keys and values in this mapping is unclear.

Recommendation Consider documenting.

```
mapping(uint16 => mapping(bytes => mapping(uint64 => bytes32)))
  ↪ public failedMessages;
```

CVF-97. FIXED

- **Category** Suboptimal
- **Source** LayerZeroStorage.sol

Recommendation The chain ID parameters should be indexed.

```
event UpdateDestination(uint16 lzChainId, bytes destination);
event UpdateDestinationAddressLength(uint16 lzChainId, uint8
  ↪ addressLength);
```

```
event MessageFailed(uint16 srcChainId, bytes srcAddress, uint64
  ↪ nonce, bytes payload);
event SendZKL(uint16 dstChainId, uint64 nonce, address sender, bytes
  ↪  receiver, uint amount);
event ReceiveZKL(uint16 srcChainId, uint64 nonce, address receiver,
  ↪ uint amount);
event SendSynchronizationProgress(uint16 dstChainId, uint64 nonce,
  ↪ bytes32 syncHash, uint progress);
event ReceiveSynchronizationProgress(uint16 srcChainId, uint64 nonce
  ↪ , bytes32 syncHash, uint progress);
```

CVF-98. INFO

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

Recommendation Events are usually named via nouns, such as “Destination”, “DestinationAddressLength” etc.

Client Comment *Not a problem.*

```
event UpdateDestination(uint16 lzChainId, bytes destination);
event UpdateDestinationAddressLength(uint16 lzChainId, uint8 addressLength);
event UpdateAPP(APP app, address contractAddress);
event MessageFailed(uint16 srcChainId, bytes srcAddress, uint64 nonce, bytes payload);
event SendZKL(uint16 dstChainId, uint64 nonce, address sender, bytes receiver, uint amount);
event ReceiveZKL(uint16 srcChainId, uint64 nonce, address receiver, uint amount);
event SendSynchronizationProgress(uint16 dstChainId, uint64 nonce, bytes32 syncHash, uint progress);
event ReceiveSynchronizationProgress(uint16 srcChainId, uint64 nonce, bytes32 syncHash, uint progress);
```

CVF-99. FIXED

- **Category** Suboptimal
- **Source** LayerZeroStorage.sol

Recommendation The “app” parameter should be indexed.

```
event UpdateAPP(APP app, address contractAddress);
```

CVF-100. INFO

- **Category** Documentation
- **Source** ILayerZeroReceiver.sol

Description The description is confusing.

Recommendation Consider rephrasing.

Client Comment *Not a problem, this file is copyed from LayerZero.*

```
// @param _payload - the signed payload is the UA bytes has encoded
    ↪ to be sent
```

CVF-102. INFO

- **Category** Flaw
- **Source** ILayerZeroEndpoint.sol

Description Ethereum chain ID is not guaranteed to fit into 16 bits.

Recommendation Consider using a wider type.

Client Comment *Not a problem, the chain ids are custom defined by LayerZero.*

```
function send(uint16 _dstChainId, bytes calldata _destination, bytes
   ↪  calldata _payload, address payable _refundAddress, address
   ↪ _zroPaymentAddress, bytes calldata _adapterParams) external
   ↪ payable;
```

```
function receivePayload(uint16 _srcChainId, bytes calldata
   ↪ _srcAddress, address _dstAddress, uint64 _nonce, uint
   ↪ _gasLimit, bytes calldata _payload) external;
```

```
function getInboundNonce(uint16 _srcChainId, bytes calldata
   ↪ _srcAddress) external view returns (uint64);
```

```
function getOutboundNonce(uint16 _dstChainId, address _srcAddress)
   ↪ external view returns (uint64);
```

```
function estimateFees(uint16 _dstChainId, address _userApplication,
   ↪ bytes calldata _payload, bool _payInZRO, bytes calldata
   ↪ _adapterParam) external view returns (uint nativeFee, uint
   ↪ zroFee);
```

```
function getChainId() external view returns (uint16);
```

```
function retryPayload(uint16 _srcChainId, bytes calldata _srcAddress
   ↪ , bytes calldata _payload) external;
```

```
function hasStoredPayload(uint16 _srcChainId, bytes calldata
   ↪ _srcAddress) external view returns (bool);
```

```
function getConfig(uint16 _version, uint16 _chainId, address
   ↪ _userApplication, uint _configType) external view returns (
   ↪ bytes memory);
```

CVF-103. INFO

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

Description As some part of the passed ether could be refunded, this function should return the actual ether amount used.

Client Comment *Not a problem, this file is copyed from LayerZero.*

```
function send(uint16 _dstChainId, bytes calldata _destination, bytes
   ↪  calldata _payload, address payable _refundAddress, address
   ↪ _zroPaymentAddress, bytes calldata _adapterParams) external
   ↪ payable;
```

CVF-104. INFO

- **Category** Documentation
- **Source** ILayerZeroEndpoint.sol

Description The returned values are not documented. Their number format is unclear.

Recommendation Consider documenting.

Client Comment *Not a problem, this file is copyed from LayerZero.*

```
function estimateFees(uint16 _dstChainId, address _userApplication,
   ↪ bytes calldata _payload, bool _payInZRO, bytes calldata
   ↪ _adapterParam) external view returns (uint nativeFee, uint
   ↪ zroFee);
```

CVF-105. INFO

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

Recommendation The types of the returned values could be more specific.

Client Comment *Not a problem, this file is copyed from LayerZero.*

```
function getSendLibraryAddress(address _userApplication) external
   ↪ view returns (address);
```

```
function getReceiveLibraryAddress(address _userApplication) external
   ↪  view returns (address);
```

CVF-106. FIXED

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

Recommendation The type of the "_endpoint" argument should be "ILayerZeroEndpoint".

```
function initialize(address _governor, address _endpoint) public
    ↪ initializer {
```

CVF-107. FIXED

- **Category** Procedural
- **Source** LayerZeroBridge.sol

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

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

CVF-108. INFO

- **Category** Suboptimal
- **Source** LayerZeroBridge.sol

Description This function does noting and it is not declared as virtual.

Recommendation Consider removing it.

Client Comment *Not a problem, we must override this function and add onlyGovernor modifier because the default implemention do nothing about authorize.*

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

CVF-112. FIXED

- **Category** Suboptimal
- **Source** LayerZeroBridge.sol

Recommendation The length check is redundant as the hash check supercedes the length check.

```
require(trustedRemote.length > 0 && srcAddress.length ==
   ↪ trustedRemote.length && keccak256(trustedRemote) == keccak256(
   ↪ srcAddress), "Invalid src");
```

CVF-115. INFO

- **Category** Procedural
- **Source** LayerZeroBridge.sol

Recommendation This interface should be moved to a separate file named “IZkLink.sol”.

Client Comment *Not a problem.*

```
interface IZkLink {
```

CVF-116. FIXED

- **Category** Documentation
- **Source** LayerZeroBridge.sol

Description The semantics of the progress values is unclear.

Recommendation Consider documenting.

```
function getSynchronizedProgress(StoredBlockInfo memory block)
   ↪ external view returns (uint256 progress);
```

```
function receiveSynchronizationProgress(bytes32 syncHash, uint256
   ↪ progress) external;
```

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