Linea Rollup Update

Date	December 2024
Auditors	Rai Yang, Vladislav Yaroshuk

1 Executive Summary

This report presents the results of our engagement with **Linea** to review **Linea Rollup Update**.

The review was conducted over two weeks, from **November 25, 2024** to **December 6, 2024**, by **Rai Yang** and **Vladislav Yaroshuk**. A total of 20 person-days were spent.

We reviewed the updates implemented between the current commit and the previous audit commit. The key updates include:

High priority changes

- Adjust Blob Submission and Finalization Events to be State Reconstruction compatible.
- Create granular roles for contracts V2: - does not include TokenBridge.
- Granular Roles for TokenBridge V2.
- Remove finalizeBlocksWithoutProof.
- Allow any address to finalize blocks if no finalization happened in the last 6 months.

Medium priority changes

- Bump solidity version to 0.8.26 across the repository.
- Return better error when verifier fails.
- Optimize message and new token creation hashing.

Low priority changes

- Apply Diligence Findings From Last Audit Round.
- Initialize-L2-MinimumFee.
- Ratelimiter ignores reset when affected amount is zero.
- Add leaf index check in sparse merkle tree verifier.
- Cleanup errors and interfaces.
- Complete contracts recommendation.

2 Scope

Our review focused on the difference between commit hash `b17e7c79b5647e47c175c6367dea30c3f1c66738` and commit hash `adb097aff4d7d32da843b16c9e1c1b21eecbf955`. The list of files in scope can be found in the Appendix.

2.1 Objectives

Together with the **Linea** team, we identified the following priorities for our review:

1. Correctness of the implementation, consistent with the intended functionality and without unintended edge cases.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.

3 Security Specification

This section describes, **from a security perspective**, the expected behavior of the system under audit. It is not a substitute for documentation. The purpose of this section is to identify specific security properties that were validated by the audit team.

3.1 Actors

The relevant actors with their respective abilities that are changed are listed below:

- Operator: Submits block data either by blobs or compressed data to L1 contract depending on the gas cost.
- Fallback Operator: Will be granted Operator role if blocks hasn't been finalized for 6 month, can't renounce himself.
- Security Council: Grants and revokes roles.
- Custom roles which can be set up in the `PauseManager` contract during initialization with the access to pause and unpause different functionality.

3.2 Trust Model

In any system, it's important to identify what trust is expected/required between various actors. For this audit, we established the trust model remains unchanged from the previous versions

3.3 Security Properties

The following is a non-exhaustive list of security properties that were reviewed in this audit:

- EIP-4844 blob submission, validation and finalization is correct and sound. Particularly the removed block numbers in the blob/calldata submission are verified in the proof in the finalization.
- Proof verification is sound (public input generation).
- Storage layout is not broken.
- Changes to the Shnarf for data submission cardinality is correct and sound.
- Efficient hashing for data submission, public input, Shnarf and last finalized state is correct.
- All the roles are not compromised and are being operated correctly.

4 Findings

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

4.1 Hardcoded `GENESIS_SHNARF` Is Incompatible With New Networks Medium Acknowledged

Description

In the `initialize` function of the `LineaRollup` contract, the genesis `shnarf` (`GENESIS_SHNARF`) is hard-coded, it cannot work with new networks that has a different genesis `shnarf` without modifying the contract, thus restricts the contract's adaptability and interoperability across different networks.

Examples

contracts/contracts/LineaRollup.sol:L143-L143

```
currentFinalizedShnarf = GENESIS_SHNARF;
```

contracts/contracts/LineaRollup.sol:L398-L414

```
function _computeShnarf(
  bytes32 _parentShnarf,
  bytes32 _snarkHash,
  bytes32 _finalStateRootHash,
  bytes32 _dataEvaluationPoint,
  bytes32 _dataEvaluationClaim
) internal pure returns (bytes32 shnarf) {
  assembly {
    let mPtr := mload(0x40)
    mstore(mPtr, _parentShnarf)
    mstore(add(mPtr, 0x20), _snarkHash)
    mstore(add(mPtr, 0x40), _finalStateRootHash)
    mstore(add(mPtr, 0x60), _dataEvaluationPoint)
    mstore(add(mPtr, 0x80), _dataEvaluationClaim)
    shnarf := keccak256(mPtr, 0xA0)
  }
}
```

Recommendation

Remove the hard-coded genesis `shnarf`, compute it from a parameter passed in the `initialize` function (`_initializationData.initialStateRootHash`).

4.2 Missing Validation for Fallback Operator Address Medium

Description

In the `initialize` function of the `LineaRollup` contract, there is no validation for fallback operator address(`_initializationData.fallbackOperator`) to be non zero. As a result, the fall back operator would fail to work in case of Linea stops submitting blobs and finalizing for 6 months.

Examples

contracts/contracts/LineaRollup.sol:L135

```
fallbackOperator = _initializationData.fallbackOperator;
```

Recommendation

Add the missing non-zero validation for fallback operator address.

4.3 Missing Validation for `chainID` in `TokenBridge` Contract Medium

Description

In the `initialize` function of the `TokenBridge` contract, the source `chainID` (`_initializationData.sourceChainId`) and target chain ID (`_initializationData.targetChainId`) of the bridge is not validated to be distinct and neither is set to zero. As a result, incorrect chain ID

will be set or identical chain IDs for the source and target chains, which fundamentally compromises the functionality of the bridge by allowing for the possibility of erroneous or unintended bridge operations.

Examples

contracts/contracts/tokenBridge/TokenBridge.sol:L160-L161

```
sourceChainId = _initializationData.sourceChainId;
targetChainId = _initializationData.targetChainId;
```

Recommendation

Add the validation for source and target chain ID to ensure they are distinct and non-zero.

4.4 Missing Validation for `defaultAdmin` in Contract Initialization Medium

Description

In the `initialize` function of both the `LineaRollup` and `TokenBridge` contract, the `DEFAULT_ADMIN_ROLE` role of the contract is granted to `_initializationData.defaultAdmin`, which is presumed to be security council account. However there is no validation checks to ensure that `_initializationData.defaultAdmin` is not a zero address. The absence of such validation could potentially result in the contracts being initialized without a designated Admin, compromising the permission management system within these contracts and leaving the contracts vulnerable to unauthorized access and manipulation.

Examples

contracts/contracts/LineaRollup.sol:L129

```
_grantRole(DEFAULT_ADMIN_ROLE, _initializationData.defaultAdmin);
```

contracts/contracts/tokenBridge/TokenBridge.sol:L155

```
_grantRole(DEFAULT_ADMIN_ROLE, _initializationData.defaultAdmin);
```

Recommendation

Add validation of non zero address for `_initializationData.defaultAdmin` in the `initialize` function for both the `LineaRollup` and `TokenBridge` contracts.

4.5 Ability to Pause and Unpause Using `UNUSED` Pause Type Minor

Description

The `pauseByType` and `unPauseByType` functions allow pausing and unpausing functionality by specifying a `PauseType` value. However, the `UNUSED` pause type, which is intended as a default value, can still be used in these functions. This creates a potential issue where someone could unintentionally pause or unpause using the `UNUSED` type and the transaction will successfully pass, which can make the authorised role think that the execution has been paused with the `GENERAL` type, but it will have no effect to the system.

Examples

contracts/contracts/lib/PauseManager.sol:L110-L136

```
/**
 * @notice Pauses functionality by specific type.
 * @dev Requires the role mapped in `_pauseTypeRoles` for the pauseType.
 * @param _pauseType The pause type value.
 */
function pauseByType(PauseType _pauseType) external onlyRole(_pauseTypeRoles[_pauseType]) {
  if (isPaused(_pauseType)) {
    revert IsPaused(_pauseType);
  }

  _pauseTypeStatusesBitMap |= 1 << uint256(_pauseType);
  emit Paused(_msgSender(), _pauseType);
}

/**
 * @notice Unpauses functionality by specific type.
 * @dev Requires the role mapped in `_unPauseTypeRoles` for the pauseType.
 * @param _pauseType The pause type value.
 */
function unPauseByType(PauseType _pauseType) external onlyRole(_unPauseTypeRoles[_pauseType]) {
  if (!isPaused(_pauseType)) {
    revert IsNotPaused(_pauseType);
  }

  _pauseTypeStatusesBitMap &= ~(1 << uint256(_pauseType));
  emit UnPaused(_msgSender(), _pauseType);
}
```

Recommendation

Add validation to the `pauseByType` and `unPauseByType` functions to prevent the use of the `UNUSED` pause type.

4.6 Redundant Validation of Block Number in Finalization Minor

Description

In the `_finalizeBlocks` function of the `LineaRollup` contract, the latest block number (`_finalizationData.endBlockNumber`) in the finalization is validated that it's greater than the last finalized block number (`_lastFinalizedBlock`) to ensure the block number sequence is correct during the finalization process. However the block number sequence is already verified in the proof along with the state in the finalization. Therefore this validation to compare the latest and last finalized block numbers is unnecessary.

Examples

contracts/contracts/LineaRollup.sol:L507-L509

```
if (_finalizationData.endBlockNumber <= _lastFinalizedBlock) {
  revert FinalBlockNumberLessThanOrEqualToLastFinalizedBlock(_finalizationData.endBlockNumber, _lastFinalizedBlock);
}
```

contracts/contracts/LineaRollup.sol:L486-L494

```
uint256 publicInput = _computePublicInput(
  _finalizationData,
  lastFinalizedShnarf,
  finalShnarf,
  lastFinalizedBlockNumber,
  _finalizationData.endBlockNumber
);

_verifyProof(publicInput, _proofType, _aggregatedProof);
```

Recommendation

Remove the validation to compare the latest and last finalized block numbers in `_finalizeBlocks` .

4.7 Redundant Parameter in the _computePublicInput Minor

Description

In the function `_computePublicInput` of the `LineaRollup` contract, the parameter `_endBlockNumber` is redundant as it's included in the parameter `_finalizationData` , the function can load the parameter from `_finalizationData` directly.

Examples

contracts/contracts/LineaRollup.sol:L683-L689

```
function _computePublicInput(
  FinalizationDataV3 calldata _finalizationData,
  bytes32 _lastFinalizedShnarf,
  bytes32 _finalShnarf,
  uint256 _lastFinalizedBlockNumber,
  uint256 _endBlockNumber
) private pure returns (uint256 publicInput) {
```

contracts/contracts/interfaces/l1/ILineaRollup.sol:L102-L115

```
struct FinalizationDataV3 {
  bytes32 parentStateRootHash;
  uint256 endBlockNumber;
  ShnarfData shnarfData;
  uint256 lastFinalizedTimestamp;
  uint256 finalTimestamp;
  bytes32 lastFinalizedL1RollingHash;
  bytes32 l1RollingHash;
  uint256 lastFinalizedL1RollingHashMessageNumber;
  uint256 l1RollingHashMessageNumber;
  uint256 l2MerkleTreesDepth;
  bytes32[] l2MerkleRoots;
  bytes l2MessagingBlocksOffsets;
}
```

Recommendation

Remove the parameter `_endBlockNumber` and load it from `_finalizationData` inside the function using `calldatacopy`

4.8 Redundant `Initializable` Import

Description

The `LineaRollup` contract redundantly imports `Initializable` from OpenZeppelin:

contracts/contracts/LineaRollup.sol:L4

```
import { Initializable } from "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
```

However, `Initializable` is already inherited through `AccessControlUpgradeable` , which includes the `Initializable` contract in its own inheritance hierarchy:

```
abstract contract AccessControlUpgradeable is Initializable, ContextUpgradeable, IAccessControlUpgradeable, ERC165Upgradeable
```

The `PermissionsManager` contract, `ZkEvmV2` contract, `PauseManager` contract, `L1MessageServiceV1` contract, `L2MessageManagerV1` contract, `RateLimiter` contract, `TokenBridge` contract are also redundantly imports `Initializable` contract:

contracts/contracts/lib/PermissionsManager.sol:L4-L14

```
import { Initializable } from "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
import { AccessControlUpgradeable } from "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
import { IGenericErrors } from "../interfaces/IGenericErrors.sol";
import { IPermissionsManager } from "../interfaces/IPermissionsManager.sol";

/**
 * @title Contract to manage permissions initialization.
 * @author ConsenSys Software Inc.
 * @custom:security-contact security-report@linea.build
 */
abstract contract PermissionsManager is Initializable, AccessControlUpgradeable, IPermissionsManager, IGenericErrors {
```

contracts/contracts/tokenBridge/TokenBridge.sol:L14

```
import { Initializable } from "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
```

contracts/contracts/ZkEvmV2.sol:L4

```
import { Initializable } from "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
```

contracts/contracts/messageService/lib/RateLimiter.sol:L4

```
import { Initializable } from "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
```

contracts/contracts/messageService/l1/v1/L1MessageServiceV1.sol:L4

```
import { Initializable } from "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
```

contracts/contracts/messageService/l2/v1/L2MessageServiceV1.sol:L4

```
import { Initializable } from "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
```

This duplicate import is unnecessary and increases code complexity without providing additional functionality.

Recommendation

We recommend removing the redundant `Initializable` import, this cleanup will reduce unnecessary imports, improving code clarity and maintainability. The contract will still compile correctly as the `Initializable` features are inherited through `AccessControlUpgradeable` or other contracts.

4.9 Use of `abi.encodeWithSelector` Instead of `abi.encodeCall`

Description

The `_verifyProof` function uses `abi.encodeWithSelector` to encode the function call data for the `call` operation with the verifier contract. While this approach is functional, it lacks the compile-time type safety provided by `abi.encodeCall`, introduced in Solidity version `0.8.11`. The absence of type checking increases the risk of encoding errors due to mismatches between the expected and actual types, which could lead to runtime failures.

Examples

contracts/contracts/ZkEvmV2.sol:L51-L53

```
(bool callSuccess, bytes memory result) = verifierToUse.call(
  abi.encodeWithSelector(IPlonkVerifier.Verify.selector, _proof, publicInput)
);
```

Recommendation

Replace `abi.encodeWithSelector` with `abi.encodeCall` to leverage compile-time type checking and adhere to Solidity best practices:

```
(bool callSuccess, bytes memory result) = verifierToUse.call(
    abi.encodeCall(IPlonkVerifier.Verify, (_proof, publicInput))
);
```

4.10 Lack of Role Reconfiguration Mechanism in `PauseManager`

Description

The `__PauseManager_init` function is used to initialize the pause and unpause roles during `PauseManager` contract deployment. However, the design does not include a mechanism to reconfigure or update these roles after initialization, the initialization function is the only one which can modify `_pauseTypeRoles` and `_unPauseTypeRoles` mappings. If a role is compromised, there is no way to revoke or reassign it, potentially allowing malicious actors to pause or unpause the system indefinitely, as well as if roles are

configured incorrectly during initialization or not configured at all, for example if the pause role has been configured and unpause role hasn't, the only recourse is to redeploy the contract, which is costly and operationally challenging.

Examples

contracts/contracts/lib/PauseManager.sol:L13-L18

```
abstract contract PauseManager is Initializable, IPauseManager, AccessControlUpgradeable {
  /// @notice This is used to pause all pausable functions.
  bytes32 public constant PAUSE_ALL_ROLE = keccak256("PAUSE_ALL_ROLE");

  /// @notice This is used to unpause all unpausable functions.
  bytes32 public constant UNPAUSE_ALL_ROLE = keccak256("UNPAUSE_ALL_ROLE");
```

contracts/contracts/lib/PauseManager.sol:L61-L80

```
/**
 * @notice Initializes the pause manager with the given pause and unpause roles.
 * @dev This function is called during contract initialization to set up the pause and unpause roles.
 * @param _pauseTypeRoleAssignments An array of PauseTypeRole structs defining the pause types and their associated roles.
 * @param _unpauseTypeRoleAssignments An array of PauseTypeRole structs defining the unpause types and their associated roles.
 */
function __PauseManager_init(
  PauseTypeRole[] calldata _pauseTypeRoleAssignments,
  PauseTypeRole[] calldata _unpauseTypeRoleAssignments
) internal onlyInitializing {
  for (uint256 i; i < _pauseTypeRoleAssignments.length; i++) {
    _pauseTypeRoles[_pauseTypeRoleAssignments[i].pauseType] = _pauseTypeRoleAssignments[i].role;
    emit PauseTypeRoleSet(_pauseTypeRoleAssignments[i].pauseType, _pauseTypeRoleAssignments[i].role);
  }

  for (uint256 i; i < _unpauseTypeRoleAssignments.length; i++) {
    _unPauseTypeRoles[_unpauseTypeRoleAssignments[i].pauseType] = _unpauseTypeRoleAssignments[i].role;
    emit UnPauseTypeRoleSet(_unpauseTypeRoleAssignments[i].pauseType, _unpauseTypeRoleAssignments[i].role);
  }
}
```

Recommendation

We recommend reviewing the architecture and adding configuration functions if neccesary, for example:

```
function updatePauseRole(uint256 pauseType, bytes32 newRole) external onlyRole(PAUSE_ALL_ROLE) {
    require(newRole != bytes32(0), "Invalid role");
    _pauseTypeRoles[pauseType] = newRole;
    emit PauseTypeRoleSet(pauseType, newRole);
}

function updateUnpauseRole(uint256 pauseType, bytes32 newRole) external onlyRole(UNPAUSE_ALL_ROLE)  {
    require(newRole != bytes32(0), "Invalid role");
    _unPauseTypeRoles[pauseType] = newRole;
    emit UnPauseTypeRoleSet(pauseType, newRole);
}
```

This would allow for dynamic role reconfiguration while maintaining security through appropriate access controls.

4.11 Lack of Restriction for Overwriting Existing Verifier Addresses in `setVerifierAddress`

Description

The `setVerifierAddress` function allows the role with `VERIFIER_SETTER_ROLE` to update the verifier address for a given proof type to set it, while with the `unsetVerifierAddress` function the `VERIFIER_UNSETTER_ROLE` has an access to unset this value. While the access for management of setter and unsetter function is designed to be separated, there are no restrictions for `VERIFIER_SETTER_ROLE` preventing the overwriting of existing verifier addresses with arbitrary values, such as dead addresses (`0xdead`) and by doing that successfully unsetting the verifier contract. Separating roles enhance security of the protocol, as if the `VERIFIER_SETTER_ROLE` is compromised, it shouldn't be able to unset the verifier blocking all of the execution, but with the current design the setter role can unset values to other dead addresses, only except for zero address, making the `VERIFIER_UNSETTER_ROLE` role and `unsetVerifierAddress` function almost redundant, and `VERIFIER_SETTER_ROLE` role overpowered.

Examples

contracts/contracts/LineaRollup.sol:L30-L34

```
/// @notice The role required to set/add  proof verifiers by type.
bytes32 public constant VERIFIER_SETTER_ROLE = keccak256("VERIFIER_SETTER_ROLE");

/// @notice The role required to set/remove  proof verifiers by type.
bytes32 public constant VERIFIER_UNSETTER_ROLE = keccak256("VERIFIER_UNSETTER_ROLE");
```

contracts/contracts/LineaRollup.sol:L189-L204

```
/**
 * @notice Adds or updates the verifier contract address for a proof type.
 * @dev VERIFIER_SETTER_ROLE is required to execute.
 * @param _newVerifierAddress The address for the verifier contract.
 * @param _proofType The proof type being set/updated.
 */
function setVerifierAddress(address _newVerifierAddress, uint256 _proofType) external onlyRole(VERIFIER_SETTER_ROLE) {
  if (_newVerifierAddress == address(0)) {
    revert ZeroAddressNotAllowed();
  }

  emit VerifierAddressChanged(_newVerifierAddress, _proofType, msg.sender, verifiers[_proofType]);

  verifiers[_proofType] = _newVerifierAddress;
}
```

contracts/contracts/LineaRollup.sol:L229-L238

```
/**
 * @notice Unset the verifier contract address for a proof type.
 * @dev VERIFIER_UNSETTER_ROLE is required to execute.
 * @param _proofType The proof type being set/updated.
 */
function unsetVerifierAddress(uint256 _proofType) external onlyRole(VERIFIER_UNSETTER_ROLE) {
  emit VerifierAddressChanged(address(0), _proofType, msg.sender, verifiers[_proofType]);

  delete verifiers[_proofType];
}
```

Recommendation

We recommend reviewing current roles, and if needed restricting the ability to overwrite existing verifier addresses to cases where the current address is unset (`address(0)`). For example:

```
if (verifiers[_proofType] != address(0)) {
    revert("Cannot overwrite existing verifier address");
}
```

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
contracts/contracts/LineaRollup.sol	`7a894ebf3c265ec95c30fd103cd36f81094b49b0`
contracts/contracts/ZkEvmV2.sol	`331303e4817f53dfc562a8576d3a0a4df1be5e72`
contracts/contracts/interfaces/IGenericErrors.sol	`18b23eff87ee67397ba14c605d4c9032b4c92f9f`
contracts/contracts/interfaces/IMessageService.sol	`55a04ced3aa0fbc64df5723a10e27f7bef0d4532`
contracts/contracts/interfaces/IPauseManager.sol	`f98f815e0b287660c184371c27119a583fe21f02`
contracts/contracts/interfaces/IPermissionsManager.sol	`194ee54bf6aab3b7d27c7077c368e13c773bf888`
contracts/contracts/interfaces/IRateLimiter.sol	`35eaa8b555e3d6a6af9223a1f834d6d50a65336e`
contracts/contracts/interfaces/l1/IL1MessageManager.sol	`aa292bdfc1f97dcc22038b3b3eaedbdd969a4ff9`
contracts/contracts/interfaces/l1/IL1MessageManagerV1.sol	`2bb90a7c41abefe1e40c39fbc1f522dee69486d1`
contracts/contracts/interfaces/l1/IL1MessageService.sol	`7b662cb6a858ee4f30fec25300c094680fed221f`
contracts/contracts/interfaces/l1/ILineaRollup.sol	`7ae593a1580e5db2e73f7f58298e7ae7bc96987f`
contracts/contracts/interfaces/l1/IPlonkVerifier.sol	`641a36cf60115831e77705c4b16783b0255ff917`
contracts/contracts/interfaces/l1/IZkEvmV2.sol	`dd1d4a9fe0ea51b3dc692b8f9fa761a2f0069d92`
contracts/contracts/interfaces/l2/IL2MessageManager.sol	`2da34026dc6365bd7be5fc3b96540536538a51ad`
contracts/contracts/interfaces/l2/IL2MessageManagerV1.sol	`437fb3c7195fb0207cd688ba5823eecac4479e76`
contracts/contracts/interfaces/l2/IL2MessageServiceV1.sol	`facddf5d0218a48f490c5eb3ca83be2c27d6453e`
contracts/contracts/interfaces/tools/IRecoverFunds.sol	`f89f0106cab16caefeea3c171fa462e62a7f0bdf`
contracts/contracts/lib/L2MessageServicePauseManager.sol	`623db409585f55e0dd8e896d89f6f4999c9d4978`
contracts/contracts/lib/LineaRollupPauseManager.sol	`4ec42e07c7e018612dbafe9299c50560078646b3`

File	SHA-1 hash
contracts/contracts/lib/Mimc.sol	`856e99a7545e99472afd9325b0f7eadc08d60161`
contracts/contracts/lib/PauseManager.sol	`b296ed26d1ac2f204dfc47e99e8b75c50971f67a`
contracts/contracts/lib/PermissionsManager.sol	`033839cd54c37a497ef89330a2e2d72c11735073`
contracts/contracts/lib/SparseMerkleProof.sol	`6b7dfac8eac9751dbebbb412debbe90c29310889`
contracts/contracts/lib/TokenBridgePauseManager.sol	`2feddbeca5e7d68747845c91c808a23dd1c03033`
contracts/contracts/lib/Utils.sol	`e135639686ae118a11f6a888dff41dc1a5930e46`
contracts/contracts/messageService/MessageServiceBase.sol	`671d90ec76b19ad5ae5f77cba0253b657efc67e2`
contracts/contracts/messageService/l1/L1MessageManager.sol	`1871ef73ccf7a9cc97fd4f499c6b6344f6b07044`
contracts/contracts/messageService/l1/L1MessageService.sol	`fc0614ebfb585a198c423032da28679b2ec5213e`
contracts/contracts/messageService/l1/TransientStorageReentrancyGuardUpgradeable.sol	`1b53ca2fd5bb3fc73528dc44ef6571c4af43fa87`
contracts/contracts/messageService/l1/v1/L1MessageManagerV1.sol	`783a7d6252389c8bf56b69dbe419362991e6aad7`
contracts/contracts/messageService/l1/v1/L1MessageServiceV1.sol	`0252df1c5d26f62a7e92260bde86258ed5b71089`
contracts/contracts/messageService/l2/L2MessageManager.sol	`bbb518b840193744b31a2e48e80d91e9ecd65f91`
contracts/contracts/messageService/l2/L2MessageService.sol	`650739fb35afdc23d8674d7fafe81ad33d355788`
contracts/contracts/messageService/l2/v1/L2MessageManagerV1.sol	`4378c1c86af94a9fae3281378a8c1ebf20da4e46`
contracts/contracts/messageService/l2/v1/L2MessageServiceV1.sol	`9aba0872221f92aad2f9a55110a4d8a5273eaa70`
contracts/contracts/messageService/lib/MessageHashing.sol	`dda45a57a2d69b7d2440268f71b235135d1424bf`
contracts/contracts/messageService/lib/RateLimiter.sol	`d663a362609adcf25337e26f51e89df1cba57d0f`
contracts/contracts/messageService/lib/SparseMerkleTreeVerifier.sol	`03575245f18630c3ed07b3f0ffd11b55dfaa01c7`
contracts/contracts/messageService/lib/TimeLock.sol	`4c841f496a82760960ee0a9edd461a75e77441ae`
contracts/contracts/messageService/lib/TransientStorageHelpers.sol	`aa99759259b16636999e38befd51b28aa8fb0728`
contracts/contracts/tokenBridge/BridgedToken.sol	`7a4f73f0acb2a3c21f3e1bd79fdf9897b241bd2b`
contracts/contracts/tokenBridge/CustomBridgedToken.sol	`c6955bf390214a34a3b8a6695966a5e0aa3acfc2`
contracts/contracts/tokenBridge/TokenBridge.sol	`5d7f050fc72154effaa498966751b188dffa16f4`
contracts/contracts/tokenBridge/interfaces/ITokenBridge.sol	`02de301c1249e89749830ce315456c1a2cded6e1`
contracts/contracts/tokenBridge/lib/StorageFiller39.sol	`aaa5dc0cf4ad750b280f7da908497e59c4c8332f`

Appendix 2 - Disclosure

Consensys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via Consensys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any third party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any third party by virtue of publishing these Reports.

A.2.1 Purpose of Reports

The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

A.2.2 Links to Other Web Sites from This Web Site

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Consensys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that Consensys and CD are not responsible for the content or operation of such Web sites, and that Consensys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that Consensys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. Consensys and CD assumes no responsibility for the use of third-party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

A.2.3 Timeliness of Content

The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice unless indicated otherwise, by Consensys and CD.