Linea Tokens Audit Report

Prepared by Cyfrin

Version 2.3

Lead Auditors

Immeas

Chinmay

July 31, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

The Linea Token is an ERC-20 deployed by the Linea team on L1 (Ethereum) and bridged to L2 (Linea chain) for distribution via `TokenAirdrop`. Bridged tokens are seeded into the airdrop contract, which computes allocations based on three soulbound factor tokens, `PRIMARY_FACTOR_ADDRESS`, `PRIMARY_CONDITIONAL_MULTIPLIER_ADDRESS`, and `SECONDARY_FACTOR_ADDRESS`. Going forward, Linea Token will also serve as the on-chain governance token across the Linea ecosystem.

4.1 Actors and Roles

- **Actors:**
  - **Linea team:** Deploys and initializes all contracts. Holds the `MINTER_ROLE` on L1, bridges minted tokens to L2, and funds the `TokenAirdrop`.
  - **Canonical Token Bridge:** Authorized to mint and burn the L2 token (`L2LineaToken`) when users bridge tokens between L1 and L2.
  - **Message Service:** The cross-chain messaging contract (configured in both `LineaToken` and `L2LineaToken`) that relays total-supply sync calls.
  - **Users:** Hold soulbound factor tokens to claim their airdrop, receive Linea Tokens, and (in future) participate in governance voting.
- **Roles:**
  - **Owner** (`Ownable2Step` on `TokenAirdrop`): Can withdraw unclaimed tokens from the airdrop contract once the claim period ends.
  - **DEFAULT_ADMIN_ROLE** (`AccessControlUpgradeable` on L1 & L2 tokens): Can grant or revoke `MINTER_ROLE` (and other admin roles).
  - **MINTER_ROLE** (L1 `LineaToken`): Authorized to mint new Linea Tokens on Ethereum.

4.2 Key Components

- **TokenAirdrop:**
  - Uses three soulbound tokens to calculate each account's allocation:
    1. **Primary factor** (`PRIMARY_FACTOR_ADDRESS`): base balance (18 decimals)
    2. **Conditional multiplier** (`PRIMARY_CONDITIONAL_MULTIPLIER_ADDRESS`): multiplier balance (9 decimals), applied to the primary balance via × `multiplier` ÷ `DENOMINATOR`
    3. **Secondary factor** (`SECONDARY_FACTOR_ADDRESS`): additional balance (18 decimals)
  - Allows anyone to call `claim()`; owner can `withdraw()` all unclaimed tokens after `CLAIM_END`.
- **LineaToken (L1):**
  - Upgradeable ERC-20 with 18 decimals, burnable, permit-enabled.
  - `MINTER_ROLE` can mint, initial supply is bridged to L2.
  - `syncTotalSupplyToL2()` is permissionless and sends `(block.timestamp, totalSupply)` via the configured `IMessageService`.
  - Uses a proxy admin pattern where the proxy admin will be a chain specific TimelockController contract.
- **L2LineaToken (L2):**
  - Upgradeable ERC-20 with 18 decimals, permit (`ERC20PermitUpgradeable`), voting (`ERC20VotesUpgradeable`), and cross-chain sync (`MessageServiceBase`).
  - Only the canonical token bridge may call `mint/burn`.
  - Only the message service and the authorized remote sender (the L1 token) may call `syncTotalSupplyFromL1`; enforces monotonic increase of `l1LineaTokenTotalSupplySyncTime`.
  - Note: both the permit and the votes extensions consume the same nonce mapping, so an off-chain client must sequence permit and delegate calls carefully to avoid nonce conflicts if both are issued in the same block.
  - Uses a proxy admin pattern where the proxy admin will be a chain specific TimelockController contract.

4.3 Centralization Risks

All privileged roles (`DEFAULT_ADMIN_ROLE` and `MINTER_ROLE`) on `LineaToken` are initially held by the security council. For stronger separation of duties, consider splitting these roles across distinct keys or multisig wallets so that minting and admin operations require independent approvals. Additionally, because both the L1 and L2 token contracts are upgradeable under the same admin framework, any compromise of these admin wallets would enable unauthorized contract upgrades. Ensure those keys are secured and managed under strict operational security.

5 Audit Scope

```
src/airdrops/TokenAirdrop.sol
src/L1/LineaToken.sol
src/L2/L2LineaToken.sol
```

6 Executive Summary

Over the course of 3 days, the Cyfrin team conducted an audit on the Linea Tokens smart contracts provided by Linea. In this period, a total of 5 issues were found.

The codebase is very well-structured and designed with security in mind. Our audit surfaced only five informational findings, largely around token-supply caps, parameter-naming inconsistencies, and a few best-practice recommendations.

The test suite is thorough, covering all functionality and revert paths.

During our review, the team also merged fixes from two prior audits. All updates, including those addressing the Cyfrin audit findings, are contained in PR#20. We validated the final merge commit (`91036da`) and found it safe.

The bytecode in the `exported-artifacts` folder was confirmed to match a fresh compile using the settings in `foundry.toml`. Finally the deployed bytecode of the `LineaToken` implementation on Ethereum and `L2LineaToken` implementation on Linea was confirmed to match the bytecode commited in `exported-artifacts`. Both two implementations pointed to by the proxy at `0x1789e0043623282d5dcc7f213d703c6d8bafbb04` (ethereum), (linea).

Summary

Project Name	Linea Tokens
Repository	linea-tokens
Commit	44640f0965a5...
Fix Commit	91036daf2331...
Audit Timeline	Jul 25th - Jul 29th, 2025
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	0
Informational	5
Gas Optimizations	0
Total Issues	5

Summary of Findings

[I-1] Mismatched total supply cap between L1 and L2 tokens	Acknowledged
[I-2] Parameter name mismatch between L2LineaToken interface and implementation	Resolved
[I-3] Prevent accidental ownership and admin renouncement	Resolved
[I-4] Consider implementing emergency pause mechanism for user facing calls	Acknowledged
[I-5] Unused AccessControl in `L2LineaToken`	Acknowledged

7 Findings

7.1 Informational

7.1.1 Mismatched total supply cap between L1 and L2 tokens

Description: OpenZeppelin's `ERC20VotesUpgradeable` enforces:

```
function _maxSupply() internal view virtual returns (uint256) {
    return type(uint208).max;
}
```

to keep vote-checkpoint values within 208 bits. As a result, any L1 total supply above `2^208  1` would be valid on L1, as the standard ERC20 implementation uses `type(uint256).max`, but invalid on L2. However, since `type(uint208).max` is astronomically larger than any realistic token issuance, this is extremely unlikely in practice.

If strict symmetry is preferred, consider enforcing the same `uint208` cap on L1, via `ERC20CappedUpgradeable` or a manual `require(totalSupply() + mintAmount <= type(uint208).max)` in `mint()`, so both chains' supply limits are aligned.

Linea: Acknowledged.

7.1.2 Parameter name mismatch between L2LineaToken interface and implementation

Description: There is a discrepancy in parameter naming between the `IL2LineaToken::syncTotalSupplyFromL1` interface and its implementation, `L2LineaToken::syncTotalSupplyFromL1`. The interface uses the names (`_l1BlockTimestamp`, `_l1TotalSupply`), whereas the implementation employs more verbose names (`_l1LineaTokenTotalSupplySyncTime`, `_l1LineaTokenSupply`) that mirror the contract's state variables. This mismatch in wording can lead to confusion when reading documentation or generating bindings, even though the ABI remains compatible.

Consider using `_l1LineaTokenTotalSupplySyncTime` and `_l1LineaTokenSupply` in both the interface and its NatSpec comments so they align with the implementation's state fields and maintain clear, consistent documentation.

Linea: Fixed in PR#17, commit `1296069`

Cyfrin: Verified. Parameters now renamed in interface and corresponding nat-spec.

7.1.3 Prevent accidental ownership and admin renouncement

Description: The inherited `renounceOwnership()` and `AccessControlUpgradeable`'s `renounceRole(DEFAULT_ADMIN_ROLE, msg.sender)` both allow the last authority to remove themselves, potentially leaving the contract permanently ownerless or admin-less—blocking critical functions like `withdraw()` or role-protected operations.

Consider override `renounceOwnership()` in `TokenAirdrop` to always revert, and similarly override `renounceRole` to prevent `DEFAULT_ADMIN_ROLE` from being renounced.

Linea: Fixed in PR#19, commits `babc8ca` and `a302e77`

Cyfrin: Verified. `renounceOwnership` overriden and reverts.

7.1.4 Consider implementing emergency pause mechanism for user facing calls

Description: Both `TokenAirdrop` and `LineaToken` expose critical operations that, once live, cannot be halted in the event of an unforeseen bug or exploit:

- `TokenAirdrop::claim` Without a pausable guard, any mis-calculation or malicious behavior in the "factor" tokens (e.g. a faulty `balanceOf` or overflow/rounding exploit) could irreversibly drain or lock the airdrop pool.
- `LineaToken::syncTotalSupplyToL2` This function bridges on-chain state to L2. If an L2 upgrade introduces a bug, or the message service changes fee semantics, repeated calls could fail or corrupt cross-chain state without any ability to stop them.

Consider integrating OpenZeppelin's `Pausable` (`Upgradeable`) so that the owner/admin can halt pause the contracts in case of any critical issues.

Linea: Acknowledged. This is intentional to provide users access to their tokens at all times.

7.1.5 Unused AccessControl in `L2LineaToken`

Description: `L2LineaToken` inherits `AccessControlUpgradeable` and grants `DEFAULT_ADMIN_ROLE` on initialization, but none of its functions (`mint`, `burn`, `syncTotalSupplyFromL1`) are protected by role checks. As a result, the AccessControl machinery isn't actually enforcing any permissions. Consider removing `AccessControlUpgradeable`.

Linea: Acknowledged. Intentionally left in so that it is not forgotten in the future.