



SMART CONTRACT AUDIT REPORT

for

LayerZero-Bridged AVA Token



Prepared By: Xiaomi Huang

PeckShield
September 30, 2026

Document Properties

Client	AVA
Title	Smart Contract Audit Report
Target	LayerZero-Bridged AVA Token
Version	1.0
Author	Xuxian Jiang
Auditors	Matthew Jiang, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	September 30, 2026	Xuxian Jiang	Final Release
1.0-rc	September 29, 2026	Xuxian Jiang	Release Candidate #1

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Xiaomi Huang
Email	contact@peckshield.com

Contents

1	Introduction	4
1.1	About AVA	4
1.2	About PeckShield	5
1.3	Methodology	5
1.4	Disclaimer	6
2	Findings	8
2.1	Summary	8
2.2	Key Findings	9
3	ERC20 Compliance Checks	10
4	Detailed Results	13
4.1	Incorrect decodeAdapterParams() Logic in LzLib	13
4.2	Trust Issue of Admin Keys	15
5	Conclusion	18
	References	19

1 | Introduction

Given the opportunity to review the design document and related source code of the LayerZero-based AVA token contract, we outline in the report our systematic method to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistency between smart contract code and the documentation, and provide additional suggestions or recommendations for improvement. Our results show that the given version of the smart contract can be further improved due to the presence of certain issues related to ERC20-compliance, security, or performance. This document outlines our audit results.

1.1 About AVA

The AVA token powers the AVA ecosystem and is used to upgrade Smart memberships, make payments, and distribute rewards. The native AVA token includes a 10-year period specification of the inflationary model per year with a max allowable tokens to mint within a year. This audit covers the LayerZero-based AVA token contract for the needed cross-chain transfers. The basic information of the audited contracts is as follows:

Table 1.1: Basic Information of LayerZero-Bridged AVA Token

Item	Description
Client	AVA
Website	https://www.avafoundation.org/
Type	ERC20 Token Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	September 30, 2026

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit. Note this audit only covers the following two contracts: `contracts/token/oft/v2/fee/OFTWithFee.sol` and `contracts/token/oft/v2/fee/ProxyOFTWithFee.sol`.

- <https://github.com/LayerZero-Labs/solidity-examples.git> (9b1a8d5)

And here are their live addresses deployed on Ethereum and BNB Chain.

- ETH-ProxyOFTWithFee <https://etherscan.io/address/0xd9483EA7214FCfd89B4Fb8f513B544920E315A52>
- BNB-OFTWithFee <https://bscscan.com/address/0xd9483EA7214FCfd89B4Fb8f513B544920E315A52>

1.2 About PeckShield

PeckShield Inc. [6] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystem by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (<https://t.me/peckshield>), Twitter (<http://twitter.com/peckshield>), or Email (contact@peckshield.com).

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [5]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk;

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

We perform the audit according to the following procedures:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- ERC20 Compliance Checks: We then manually check whether the implementation logic of the audited smart contract(s) follows the standard ERC20 specification and other best practices.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Table 1.2: Vulnerability Severity Classification

Impact				
High	Critical	High	Medium	
Medium	High	Medium	Low	
Low	Medium	Low	Low	
		High	Medium	Low
		Likelihood		

To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

Table 1.3: The Full List of Check Items

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead of Transfer
	Costly Loop
	(Unsafe) Use of Untrusted Libraries
	(Unsafe) Use of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
	Approve / TransferFrom Race Condition
ERC20 Compliance Checks	Compliance Checks (Section 3)
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

2 | Findings

2.1 Summary

Here is a summary of our findings after analyzing the LayerZero-based AVA token contract. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logics, examine system operations, and place ERC20-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	0	
Low	2	
Informational	0	
Total	2	

Moreover, we explicitly evaluate whether the given contracts follow the standard ERC20 specification and other known best practices, and validate its compatibility with other similar ERC20 tokens and current DeFi protocols. The detailed ERC20 compliance checks are reported in Section 3. After that, we examine a few identified issues of varying severities that need to be brought up and paid more attention to. (The findings are categorized in the above table.) Additional information can be found in the next subsection, and the detailed discussions are in Section 4.

2.2 Key Findings

Overall, no ERC20 compliance issue was found and our detailed checklist can be found in Section 3. While there is no critical or high severity issue, the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 2 low-severity vulnerabilities.

Table 2.1: Key LayerZero-Bridged AVA Token Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Low	Incorrect decodeAdapterParams() Logic in LzLib	Business Logic	Not Applicable
PVE-002	Low	Trust Issue Of Admin Keys	Security Features	Mitigated

Besides recommending specific countermeasures to mitigate the above issue(s), we also emphasize that it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms need to kick in at the very moment when the contracts are being deployed in mainnet. Please refer to Section 3 for our detailed compliance checks and Section 4 for elaboration of reported issues.

3 | ERC20 Compliance Checks

The ERC20 specification defines a list of API functions (and relevant events) that each token contract is expected to implement (and emit). The failure to meet these requirements means the token contract cannot be considered to be ERC20-compliant. Naturally, as the first step of our audit, we examine the list of API functions defined by the ERC20 specification and validate whether there exist any inconsistency or incompatibility in the implementation or the inherent business logic of the audited contract(s).

Table 3.1: Basic `View-only` Functions Defined in The ERC20 Specification

Item	Description	Status
name()	Is declared as a public view function	✓
	Returns a string, for example "Tether USD"	✓
symbol()	Is declared as a public view function	✓
	Returns the symbol by which the token contract should be known, for example "USDT". It is usually 3 or 4 characters in length	✓
decimals()	Is declared as a public view function	✓
	Returns decimals, which refers to how divisible a token can be, from 0 (not at all divisible) to 18 (pretty much continuous) and even higher if required	✓
totalSupply()	Is declared as a public view function	✓
	Returns the number of total supplied tokens, including the total minted tokens (minus the total burned tokens) ever since the deployment	✓
balanceOf()	Is declared as a public view function	✓
	Anyone can query any address' balance, as all data on the blockchain is public	✓
allowance()	Is declared as a public view function	✓
	Returns the amount which the spender is still allowed to withdraw from the owner	✓

Our analysis shows that there is no ERC20 inconsistency or incompatibility issue found in the audited AVA token contract. In the surrounding two tables, we outline the respective list of basic [view-only](#) functions (Table 3.1) and key state-changing functions (Table 3.2) according to the widely-adopted ERC20 specification.

Table 3.2: Key State-Changing Functions Defined in The ERC20 Specification

Item	Description	Status
transfer()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token transfer status	✓
	Reverts if the caller does not have enough tokens to spend	✓
	Allows zero amount transfers	✓
	Emits Transfer() event when tokens are transferred successfully (include 0 amount transfers)	✓
	Reverts while transferring to zero address	✓
transferFrom()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token transfer status	✓
	Reverts if the spender does not have enough token allowances to spend	✓
	Updates the spender's token allowances when tokens are transferred successfully	✓
	Reverts if the from address does not have enough tokens to spend	✓
	Allows zero amount transfers	✓
	Emits Transfer() event when tokens are transferred successfully (include 0 amount transfers)	✓
	Reverts while transferring from zero address	✓
	Reverts while transferring to zero address	✓
approve()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token approval status	✓
	Emits Approval() event when tokens are approved successfully	✓
	Reverts while approving to zero address	✓
Transfer() event	Is emitted when tokens are transferred, including zero value transfers	✓
	Is emitted with the from address set to <i>address(0x0)</i> when new tokens are generated	✓
Approval() event	Is emitted on any successful call to approve()	✓

In addition, we perform a further examination on certain features that are permitted by the ERC20 specification or even further extended in follow-up refinements and enhancements, but not required for implementation. These features are generally helpful, but may also impact or bring certain incompatibility with current DeFi protocols. Therefore, we consider it is important to highlight them as well. This list is shown in Table 3.3.

Table 3.3: Additional `opt-in` Features Examined in Our Audit

Feature	Description	Opt-in
Deflationary	Part of the tokens are burned or transferred as fee while on <code>transfer()/transferFrom()</code> calls	—
Rebasing	The <code>balanceOf()</code> function returns a re-based balance instead of the actual stored amount of tokens owned by the specific address	—
Pausable	The token contract allows the owner or privileged users to pause the token transfers and other operations	—
Whitelistable	The token contract allows the owner or privileged users to whitelist a specific address such that only token transfers and other operations related to that address are allowed	—
Mintable	The token contract allows the owner or privileged users to mint tokens to a specific address	✓
Burnable	The token contract allows the owner or privileged users to burn tokens of a specific address	✓

4 | Detailed Results

4.1 Incorrect decodeAdapterParams() Logic in LzLib

- ID: PVE-001
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: LzLib
- Category: Business Logic [4]
- CWE subcategory: CWE-841 [2]

Description

The audited LayerZero-based AVA token contract has a built-in library LzLib. This library is used to build adapter parameters used in LayerZero-based cross-chain communication. In the process of examining this library contract, we notice a helper function to decode adapter parameters is improperly implemented.

To elaborate, we show below the implementation of this related helper function, i.e., `decodeAdapterParams()`. The logic is rather straightforward in validating the size of the given input bytes and extracting the transaction type. Based on the extracted transaction type, it further decodes type-specific additional parameters, e.g., `airdropAmount` and `airdropAddress` for the transaction type 2. Our analysis shows that the decoded `airdropAddress` (for the transaction type 2) is obtained with the offset 86 (line 76), which should be corrected as 98. Further, since it is an address type, we need to obtain the result in the form of `byte32` as `airdropAddressBytes` and then make a type cast to `address` by `airdropAddress = payable(address(uint160(uint256(airdropAddressBytes))))`.

```

55     function decodeAdapterParams(bytes memory _adapterParams)
56         internal
57         pure
58         returns (
59             uint16 txType,
60             uint uaGas,
61             uint airdropAmount,
62             address payable airdropAddress
63         )

```

```

64     {
65         require(_adapterParams.length == 34 _adapterParams.length > 66, "Invalid
            adapterParams");
66         assembly {
67             txType := mload(add(_adapterParams, 2))
68             uaGas := mload(add(_adapterParams, 34))
69         }
70         require(txType == 1 txType == 2, "Unsupported txType");
71         require(uaGas > 0, "Gas too low");
72
73         if (txType == 2) {
74             assembly {
75                 airdropAmount := mload(add(_adapterParams, 66))
76                 airdropAddress := mload(add(_adapterParams, 86))
77             }
78         }
79     }

```

Listing 4.1: LzLib::decodeAdapterParams()

```

37     function buildAirdropAdapterParams(uint _uaGas, AirdropParams memory _params)
            internal pure returns (bytes memory) {
38         require(_params.airdropAmount > 0, "Airdrop amount must be greater than 0");
39         require(_params.airdropAddress != bytes32(0x0), "Airdrop address must be set");
40
41         // txType 2
42         // bytes [2      32      32      bytes[]      ]
43         // fields [txType extraGas dstNativeAmt dstNativeAddress]
44         return abi.encodePacked(uint16(2), _uaGas, _params.airdropAmount, _params.
            airdropAddress);
45     }

```

Listing 4.2: LzLib::buildAirdropAdapterParams()

Recommendation Revisit the above routine to properly decode the airdropAddress parameter for the transaction type 2.

Status The issue has been resolved as it is not currently used in either deployed contract.

4.2 Trust Issue of Admin Keys

- ID: PVE-002
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: Multiple Contracts
- Category: Security Features [3]
- CWE subcategory: CWE-287 [1]

Description

In the audited token contract, there is a privileged admin account `owner` that plays a critical role in regulating the token-wide operations (e.g., configure parameters, manage peers, set fees, and execute privileged ops). In the following, we show the representative function potentially affected by this privilege.

```

27     function setDefaultFeeBp(uint16 _feeBp) public virtual onlyOwner {
28         require(_feeBp <= BP_DENOMINATOR, "Fee: fee bp must be <= BP_DENOMINATOR");
29         defaultFeeBp = _feeBp;
30         emit SetDefaultFeeBp(defaultFeeBp);
31     }
32
33     function setFeeBp(uint16 _dstChainId, bool _enabled, uint16 _feeBp) public virtual
        onlyOwner {
34         require(_feeBp <= BP_DENOMINATOR, "Fee: fee bp must be <= BP_DENOMINATOR");
35         chainIdToFeeBps[_dstChainId] = FeeConfig(_feeBp, _enabled);
36         emit SetFeeBp(_dstChainId, _enabled, _feeBp);
37     }
38
39     function setFeeOwner(address _feeOwner) public virtual onlyOwner {
40         require(_feeOwner != address(0x0), "Fee: feeOwner cannot be 0x");
41         feeOwner = _feeOwner;
42         emit SetFeeOwner(_feeOwner);
43     }

```

Listing 4.3: Example Privileged Operations in Fee

```

115     function setConfig(
116         uint16 _version,
117         uint16 _chainId,
118         uint _configType,
119         bytes calldata _config
120     ) external override onlyOwner {
121         lzEndpoint.setConfig(_version, _chainId, _configType, _config);
122     }
123
124     function setSendVersion(uint16 _version) external override onlyOwner {
125         lzEndpoint.setSendVersion(_version);
126     }

```

```

127
128     function setReceiveVersion(uint16 _version) external override onlyOwner {
129         lzEndpoint.setReceiveVersion(_version);
130     }
131
132     function forceResumeReceive(uint16 _srcChainId, bytes calldata _srcAddress) external
133         override onlyOwner {
134         lzEndpoint.forceResumeReceive(_srcChainId, _srcAddress);}
135
136     // _path = abi.encodePacked(remoteAddress, localAddress)
137     // this function set the trusted path for the cross-chain communication
138     function setTrustedRemote(uint16 _remoteChainId, bytes calldata _path) external
139         onlyOwner {
140         trustedRemoteLookup[_remoteChainId] = _path;
141         emit SetTrustedRemote(_remoteChainId, _path);}
142
143     function setTrustedRemoteAddress(uint16 _remoteChainId, bytes calldata
144         _remoteAddress) external onlyOwner {
145         trustedRemoteLookup[_remoteChainId] = abi.encodePacked(_remoteAddress, address(
146             this));
147         emit SetTrustedRemoteAddress(_remoteChainId, _remoteAddress);}
148
149     function setPrecrime(address _precrime) external onlyOwner {
150         precrime = _precrime;
151         emit SetPrecrime(_precrime);}
152
153     function setMinDstGas(
154         uint16 _dstChainId,
155         uint16 _packetType,
156         uint _minGas) external onlyOwner {
157         minDstGasLookup[_dstChainId][_packetType] = _minGas;
158         emit SetMinDstGas(_dstChainId, _packetType, _minGas);}
159
160     // if the size is 0, it means default size limit
161     function setPayloadSizeLimit(uint16 _dstChainId, uint _size) external onlyOwner {
162         payloadSizeLimitLookup[_dstChainId] = _size;}

```

Listing 4.4: Example Privileged Operation in LzApp

We emphasize that the privilege assignment may be necessary and consistent with the protocol design. However, it would be worrisome if the privileged account is not governed by a DAO-like structure. Note that a compromised account would allow the new owner to modify a number of sensitive system parameters, which may directly undermine the assumption of the token design.

Recommendation Promptly transfer the privileged account to the intended DAO-like governance contract. All changed to privileged operations may need to be mediated with necessary timelocks. Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

Status The issue has been mitigated as the team makes use of a multi-sig account as the

owner. In particular, on Ethereum, the owner account is 3-of-4 Safe, deployed at <https://etherscan.io/address/0x58653987Ff3837ADBE6383F670f6935fcDE521b0>. And on BNB Chain, the owner account is 3-of-5 Safe, deployed at <https://bscscan.com/address/0xb7954632368474f297d6c056b9c3af9957FbE446>.



5 | Conclusion

In this security audit, we have examined the design and implementation of LayerZero-based AVA token. During our audit, we first checked all respects related to the compatibility of the ERC20 specification and other known ERC20 pitfalls/vulnerabilities. We then proceeded to examine other areas such as coding practices and business logics. Overall, although no critical level vulnerabilities were discovered, we identified several issues that need to be promptly addressed. In the meantime, as disclaimed in Section 1.4, we appreciate any constructive feedbacks or suggestions about our findings, procedures, audit scope, etc.



References

- [1] MITRE. CWE-287: Improper Authentication. <https://cwe.mitre.org/data/definitions/287.html>.
- [2] MITRE. CWE-841: Improper Enforcement of Behavioral Workflow. <https://cwe.mitre.org/data/definitions/841.html>.
- [3] MITRE. CWE CATEGORY: 7PK - Security Features. <https://cwe.mitre.org/data/definitions/254.html>.
- [4] MITRE. CWE CATEGORY: Business Logic Errors. <https://cwe.mitre.org/data/definitions/840.html>.
- [5] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.
- [6] PeckShield. PeckShield Inc. <https://www.peckshield.com>.