

[etherauthority.io](https://etherauthority.io)  
[audit@etherauthority.io](mailto:audit@etherauthority.io)

# SMART CONTRACT

Security Audit Report

**Project:** **Osaka Protocol**  
**Website:** [osaka.win](https://osaka.win)  
**Platform:** **Base Chain Network**  
**Language:** **Solidity**  
**Date:** **June 10th, 2024**

# Table of contents

Introduction .....	4
Project Background .....	4
Audit Scope .....	5
Code Audit History .....	6
Severity Definitions .....	6
Claimed Smart Contract Features .....	7
Audit Summary .....	8
Technical Quick Stats .....	9
Business Risk Analysis .....	10
Code Quality .....	11
Documentation .....	11
Use of Dependencies .....	11
AS-IS overview .....	12
Audit Findings .....	15
Conclusion .....	18
Our Methodology .....	19
Disclaimers .....	21
Appendix	
• Code Flow Diagram .....	22
• Slither Results Log .....	23
• Solidity static analysis .....	26
• Solhint Linter .....	27

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

# Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the Osaka Protocol (OSAK) smart contract from osaka.win was audited extensively. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on June 10th, 2024.

**The purpose of this audit was to address the following:**

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

# Project Background

## Website Details



- A decentralized finance initiative that aims to create an environment of equality and shared responsibility, promoting the concept that ownership percentage equates to one's level of responsibility and influence.
- Each holder's power is proportionate to their ownership stake, It aims to cultivate a thriving ecosystem driven by community collaboration and true decentralization.
- Osaka Protocol is a platform that focuses on creating a secure and decentralized ecosystem for digital asset management and transactions.
- It emphasizes user privacy, security, and efficient digital asset handling through its various protocols and features.

## Code Details

- This contract suite provides a comprehensive implementation for cross-chain token transfers with support for fees and ERC20 token standards. The key functionalities include:
  - Cross-chain token transfers with fee handling.
  - Estimating transfer and call fees.
  - Event logging for transparency and auditing.
  - Integration with LayerZero for cross-chain communication.
- This setup is particularly useful for decentralized finance (DeFi) applications where cross-chain token transfers and interoperability are critical.

## Audit scope

Name	<b>Code Review and Security Analysis Report for Osaka Protocol (OSAK) Smart Contract</b>
Platform	<b>Base Chain Network</b>
Language	<b>Solidity</b>
File	OFTWithFee.sol
Smart Contract Code	<a href="#">0xbFd5206962267c7b4b4A8B3D76AC2E1b2A5c4d5e</a>
Audit Date	June 10th,2024
Audit Result	<b>Passed</b>

# Code Audit History



0  
Total Findings

0  
Critical






0  
High

0  
Medium

0  
Low

0  
Informational

## Severity Definitions

0		<b>Critical</b>	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
0		<b>High</b>	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. Public access is crucial.
0		<b>Medium</b>	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens loss
0		<b>Low</b>	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
0		<b>Lowest / Informational / Best Practice</b>	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

## Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
<p><b>Tokenomics:</b></p> <ul style="list-style-type: none"><li>• Name: Osaka Protocol</li><li>• Symbol: OSAK</li><li>• Decimals: 18</li><li>• Shared Decimals: 4</li><li>• Default Fee BP: 10</li></ul>	<p><b>YES, This is valid.</b></p>
<p><b>Owner Specifications:</b></p> <ul style="list-style-type: none"><li>• Update generic config for LayerZero user Application.</li><li>• Update the send/receive version.</li><li>• Set the trusted path for cross-chain communication.</li><li>• Set the precrime address.</li><li>• Set the payload size limit.</li><li>• Set default fee bp.</li><li>• Set the fee of the owner.</li><li>• The current owner can transfer ownership of the contract to a new account.</li><li>• Deleting ownership will leave the contract without an owner, removing any owner-only functionality.</li></ul>	<p><b>YES, This is valid.</b></p> <p><b>We advise renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.</b></p>

# Audit Summary

According to the standard audit assessment, the Customer's solidity-based smart contracts are **"Well Secured"**. Also, these contracts contain owner control, which does not make them fully decentralized.

Unsecured

Poor Secured

Secured

Well Secured

You are here



We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

**We found 0 critical, 0 high, 0 medium, 0 low, and 0 very low-level issues.**

**Investor Advice:** A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.



## Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	"Out of Gas" Issue	Passed
	High consumption 'for/while' loop	Passed
	High consumption 'storage' storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Passed
	"Short Address" Attack	Passed
	"Double Spend" Attack	Passed

Overall Audit Result: **PASSED**

# Business Risk Analysis

Category	Result
● Buy Tax	0%
● Sell Tax	0%
● Cannot Buy	No
● Cannot Sell	No
● Max Tax	0%
● Modify Tax	No
● Fee Check	Not Detected
● Is Honeypot	Not Detected
● Trading Cooldown	Not Detected
● Can Pause Trade?	Not Detected
● Pause Transfer?	No
● Max Tax?	No
● Is it Anti-whale?	Not Detected
● Is Anti-bot?	Not Detected
● Is it a Blacklist?	No
● Blacklist Check	No
● Can Mint?	No
● Is it a Proxy Contract?	Yes
● Is it used Open Source?	Yes
● External Call Risk?	No
● Balance Modifiable?	No
● Can Take Ownership?	No
● Ownership Renounce?	No
● Hidden Owner?	Not Detected
● Self Destruction?	Not Detected
● Auditor Confidence	High

**Overall Audit Result: PASSED**

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

**Email: [audit@EtherAuthority.io](mailto:audit@EtherAuthority.io)**

## Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in the Osaka Protocol are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the Osaka Protocol.

The EtherAuthority team has not provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

## Documentation

We were given an Osaka Protocol (OSAK) smart contract code in the form of a [basescan](#) web link.

As mentioned above, code parts are well commented on. And the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

-

## Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure are based on well-known industry standard open-source projects.

Apart from libraries, its functions are used in external smart contract calls.

# AS-IS overview

## OFTWithFee.sol

### Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	circulating supply	read	Passed	No Issue
3	token	read	Passed	No Issue
4	debitFrom	internal	Passed	No Issue
5	creditTo	internal	Passed	No Issue
6	transferFrom	internal	Passed	No Issue
7	ld2sdRate	internal	Passed	No Issue
8	sendFrom	write	Passed	No Issue
9	sendAndCall	write	Passed	No Issue
10	supportsInterface	read	Passed	No Issue
11	estimateSendFee	read	Passed	No Issue
12	estimateSendAndCallFee	read	Passed	No Issue
13	circulatingSupply	read	Passed	No Issue
14	token	read	Passed	No Issue
15	transferFrom	read	Passed	No Issue
16	name	read	Passed	No Issue
17	symbol	read	Passed	No Issue
18	decimals	read	Passed	No Issue
19	totalSupply	read	Passed	No Issue
20	balanceOf	read	Passed	No Issue
21	transfer	write	Passed	No Issue
22	allowance	read	Passed	No Issue
23	approve	write	Passed	No Issue
24	transferFrom	write	Passed	No Issue
25	increaseAllowance	write	Passed	No Issue
26	decreaseAllowance	write	Passed	No Issue
27	transfer	internal	Passed	No Issue
28	mint	internal	Passed	No Issue
29	burn	internal	Passed	No Issue
30	approve	internal	Passed	No Issue
31	spendAllowance	internal	Passed	No Issue
32	beforeTokenTransfer	internal	Passed	No Issue
33	afterTokenTransfer	internal	Passed	No Issue
34	callOnOFTReceived	write	Passed	No Issue
35	estimateSendFee	internal	Passed	No Issue
36	estimateSendAndCallFee	internal	Passed	No Issue
37	nonblockingLzReceive	internal	Passed	No Issue
38	send	internal	Passed	No Issue
39	sendAck	internal	Passed	No Issue
40	sendAndCall	internal	Passed	No Issue
41	sendAndCallAck	internal	Passed	No Issue

42	isContract	internal	Passed	No Issue
43	ld2sd	internal	Passed	No Issue
44	sd2ld	internal	Passed	No Issue
45	removeDust	internal	Passed	No Issue
46	encodeSendPayload	internal	Passed	No Issue
47	decodeSendPayload	internal	Passed	No Issue
48	encodeSendAndCallPayload	internal	Passed	No Issue
49	decodeSendAndCallPayload	read	Passed	No Issue
50	addressToBytes32	internal	Passed	No Issue
51	_debitFrom	internal	Passed	No Issue
52	_creditTo	internal	Passed	No Issue
53	transferFrom	internal	Passed	No Issue
54	ld2sdRate	internal	Passed	No Issue
55	setDefaultFeeBp	write	access only owner	No Issue
56	setFeeBp	write	access only owner	No Issue
57	setFeeOwner	write	access only owner	No Issue
58	quoteOFTFee	read	Passed	No Issue
59	_payOFTFee	internal	Passed	No Issue
60	transferFrom	internal	Passed	No Issue
61	_blockingLzReceive	internal	Passed	No Issue
62	storeFailedMessage	internal	Passed	No Issue
63	nonblockingLzReceive	write	Passed	No Issue
64	_nonblockingLzReceive	internal	Passed	No Issue
65	retryMessage	write	Passed	No Issue
66	lzReceive	write	Passed	No Issue
67	_blockingLzReceive	internal	Passed	No Issue
68	_lzSend	internal	Passed	No Issue
69	_checkGasLimit	internal	Passed	No Issue
70	_getGasLimit	internal	Passed	No Issue
71	checkPayloadSize	internal	Passed	No Issue
72	getConfig	external	Passed	No Issue
73	setConfig	external	access only owner	No Issue
74	setSendVersion	external	access only owner	No Issue
75	setReceiveVersion	external	access only owner	No Issue
76	forceResumeReceive	external	access only owner	No Issue
77	setTrustedRemote	external	access only owner	No Issue
78	setTrustedRemoteAddress	external	access only owner	No Issue
79	getTrustedRemoteAddress	external	Passed	No Issue

80	setPrecrime	external	access only owner	No Issue
81	setMinDstGas	external	access only owner	No Issue
82	setPayloadSizeLimit	external	access only owner	No Issue
83	isTrustedRemote	external	Passed	No Issue
84	onlyOwner	modifier	Passed	No Issue
85	owner	r	Passed	No Issue
86	_checkOwner	internal	Passed	No Issue
87	renounceOwnership	write	access only owner	No Issue
88	transferOwnership	write	access only owner	No Issue
89	_transferOwnership	internal	Passed	No Issue

# Audit Findings

## Critical Severity

No Critical severity vulnerabilities were found.

## High Severity

No High severity vulnerabilities were found.

## Medium

No Medium-severity vulnerabilities were found.

## Low

No Low-severity vulnerabilities were found.

## Very Low / Informational / Best practices:

No Very-Low-severity vulnerabilities were found.

# Centralization

This smart contract has some functions that can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble.

Centralized

Decentralized

You are here



The following are Admin functions:

## LzApp.sol

- setConfig: The owner can create a generic config for the LayerZero user Application.
- setSendVersion: The owner can update the send version.
- setReceiveVersion: The owner can update the receive version.
- forceResumeReceive: The owner can force resume receiver ID.
- setTrustedRemote: The owner can set the trusted path for the cross-chain communication.
- setTrustedRemoteAddress: The owner can set the trusted path for the cross-chain communication address.
- setPrecrime: The owner can set the precrime address.
- setMinDstGas: The owner can set a minimum DST gas.
- setPayloadSizeLimit: The owner can set the payload size limit.

## Fee.sol

- setDefaultFeeBp: The owner can set default fee bp.
- setFeeBp: The owner can set fee bp.
- setFeeOwner: The owner can set the fee of the owner.



## Ownable.sol

- `renounceOwnership`: Deleting ownership will leave the contract without an owner, removing any owner-only functionality.
- `transferOwnership`: Current owner can transfer ownership of the contract to a new account.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

## Conclusion

We were given a contract code in the form of a [basescan](#) web link. And we have used all possible tests based on given objects as files. We observed no issue in the smart contracts. So, **it's good to go for the production.**

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **“Well Secured”**.

# Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

## **Manual Code Review:**

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

## **Vulnerability Analysis:**

Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

## **Documenting Results:**

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

## **Suggested Solutions:**

We search for immediate mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

# Disclaimers

## EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

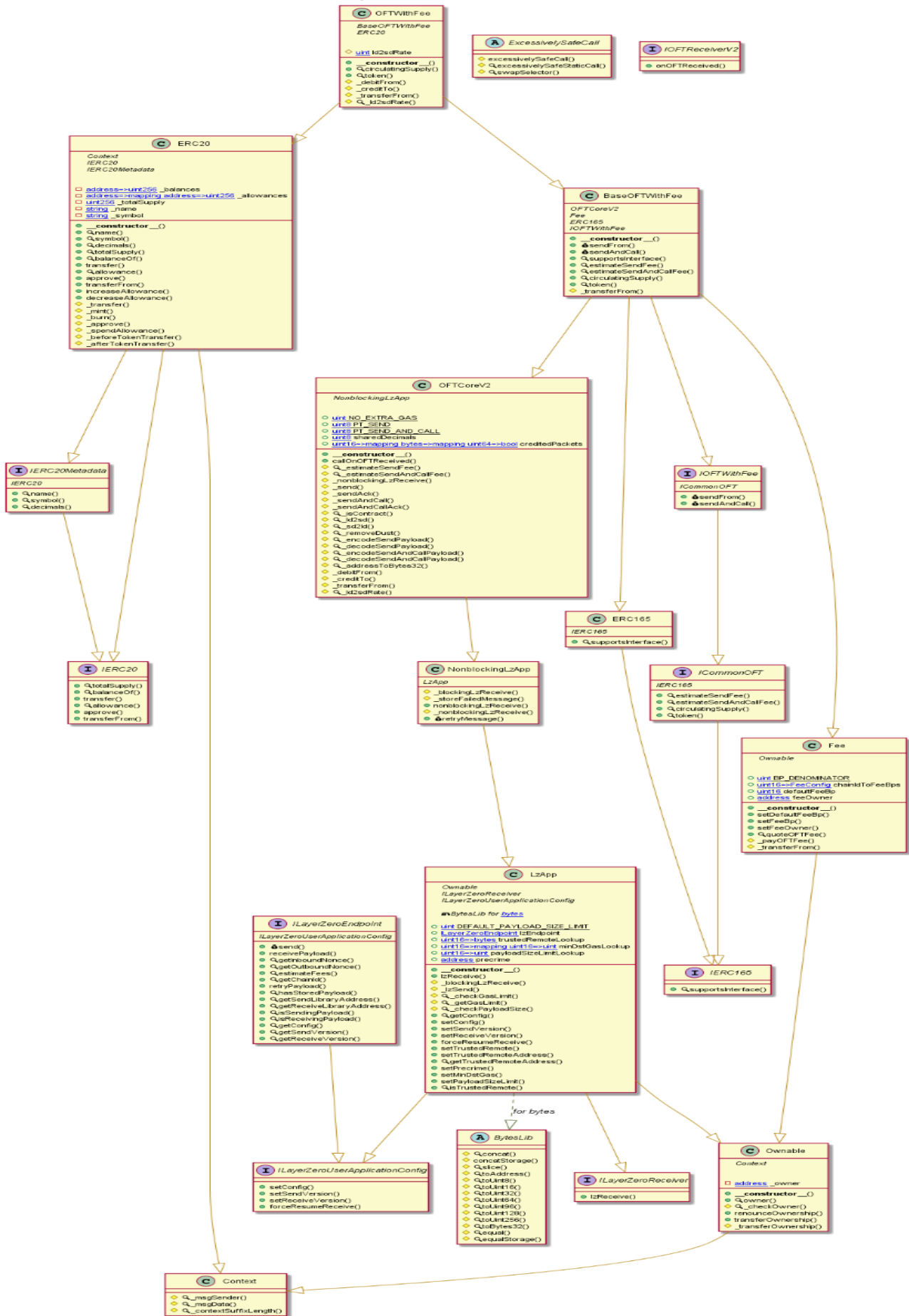
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

## Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

# Appendix

## Code Flow Diagram - Osaka Protocol (OSAK)



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: [audit@EtherAuthority.io](mailto:audit@EtherAuthority.io)

## Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

### OFTWithFee.sol

```
INFO:Detectors:
OFTWithFee.constructor(string,string,uint8,address)._name (OFTWithFee.sol#1399) shadows:
  - ERC20._name (OFTWithFee.sol#652) (state variable)
OFTWithFee.constructor(string,string,uint8,address)._symbol (OFTWithFee.sol#1399) shadows:
  - ERC20._symbol (OFTWithFee.sol#653) (state variable)
OFTWithFee.constructor(string,string,uint8,address).decimals (OFTWithFee.sol#1400) shadows:
  - ERC20.decimals() (OFTWithFee.sol#668-670) (function)
  - IERC20Metadata.decimals() (OFTWithFee.sol#27) (function)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Reentrancy in OFTCoreV2._send(address,uint16,bytes32,uint256,address,address,bytes)
(OFTWithFee.sol#1164-1183):
  External calls:
  -
  _lzSend(_dstChainId,lzPayload,_refundAddress,_zroPaymentAddress,_adapterParams,msg.value)
(OFTWithFee.sol#1180)
    - lzEndpoint.send{value:
_nativeFee}(_dstChainId,trustedRemote,_payload,_refundAddress,_zroPaymentAddress,_adapter
Params) (OFTWithFee.sol#848)
  Event emitted after the call(s):
  - SendToChain(_dstChainId,_from,_toAddress,amount) (OFTWithFee.sol#1182)
Reentrancy in
OFTCoreV2._sendAndCall(address,uint16,bytes32,uint256,bytes,uint64,address,address,bytes)
(OFTWithFee.sol#1202-1223):
  External calls:
  -
  _lzSend(_dstChainId,lzPayload,_refundAddress,_zroPaymentAddress,_adapterParams,msg.value)
(OFTWithFee.sol#1220)
    - lzEndpoint.send{value:
_nativeFee}(_dstChainId,trustedRemote,_payload,_refundAddress,_zroPaymentAddress,_adapter
```

Params) (OFTWithFee.sol#848)

Event emitted after the call(s):

- SendToChain(\_dstChainId,\_from,\_toAddress,amount) (OFTWithFee.sol#1222)

Reference:

<https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3>

INFO:Detectors:

Pragma version^0.8.0 (OFTWithFee.sol#2) allows old versions

solc-0.8.0 is not recommended for deployment

Reference:

<https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity>

INFO:Detectors:

Parameter

BaseOFTWithFee.sendFrom(address,uint16,bytes32,uint256,uint256,ICommonOFT.LzCallParams).\_from (OFTWithFee.sol#1363) is not in mixedCase

Parameter

BaseOFTWithFee.sendFrom(address,uint16,bytes32,uint256,uint256,ICommonOFT.LzCallParams).\_dstChainId (OFTWithFee.sol#1363) is not in mixedCase

Parameter

BaseOFTWithFee.sendFrom(address,uint16,bytes32,uint256,uint256,ICommonOFT.LzCallParams).\_toAddress (OFTWithFee.sol#1363) is not in mixedCase

Parameter

BaseOFTWithFee.sendFrom(address,uint16,bytes32,uint256,uint256,ICommonOFT.LzCallParams).\_amount (OFTWithFee.sol#1363) is not in mixedCase

Parameter

BaseOFTWithFee.sendFrom(address,uint16,bytes32,uint256,uint256,ICommonOFT.LzCallParams).\_minAmount (OFTWithFee.sol#1363) is not in mixedCase

Parameter

BaseOFTWithFee.sendFrom(address,uint16,bytes32,uint256,uint256,ICommonOFT.LzCallParams).\_callParams (OFTWithFee.sol#1363) is not in mixedCase

Parameter

BaseOFTWithFee.sendAndCall(address,uint16,bytes32,uint256,uint256,bytes,uint64,ICommonOFT.LzCallParams).\_from (OFTWithFee.sol#1369) is not in mixedCase

Parameter

BaseOFTWithFee.sendAndCall(address,uint16,bytes32,uint256,uint256,bytes,uint64,ICommonOFT.LzCallParams).\_dstChainId (OFTWithFee.sol#1369) is not in mixedCase

Parameter

BaseOFTWithFee.sendAndCall(address,uint16,bytes32,uint256,uint256,bytes,uint64,ICommonOFT.LzCallParams).\_toAddress (OFTWithFee.sol#1369) is not in mixedCase

Parameter

BaseOFTWithFee.sendAndCall(address,uint16,bytes32,uint256,uint256,bytes,uint64,ICommonOFT.LzCallParams).\_amount (OFTWithFee.sol#1369) is not in mixedCase

Parameter

BaseOFTWithFee.sendAndCall(address,uint16,bytes32,uint256,uint256,bytes,uint64,ICommonOFT.LzCallParams).\_minAmount (OFTWithFee.sol#1369) is not in mixedCase

Parameter

BaseOFTWithFee.sendAndCall(address,uint16,bytes32,uint256,uint256,bytes,uint64,ICommonOFT.LzCallParams).\_payload (OFTWithFee.sol#1369) is not in mixedCase

Parameter

BaseOFTWithFee.sendAndCall(address,uint16,bytes32,uint256,uint256,bytes,uint64,ICommon



OFT.LzCallParams).\_dstGasForCall (OFTWithFee.sol#1369) is not in mixedCase  
Parameter  
BaseOFTWithFee.sendAndCall(address,uint16,bytes32,uint256,uint256,bytes,uint64,ICommon  
OFT.LzCallParams).\_callParams (OFTWithFee.sol#1369) is not in mixedCase  
Parameter BaseOFTWithFee.estimateSendFee(uint16,bytes32,uint256,bool,bytes).\_dstChainId  
(OFTWithFee.sol#1379) is not in mixedCase  
Parameter BaseOFTWithFee.estimateSendFee(uint16,bytes32,uint256,bool,bytes).\_toAddress  
(OFTWithFee.sol#1379) is not in mixedCase  
Parameter BaseOFTWithFee.estimateSendFee(uint16,bytes32,uint256,bool,bytes).\_amount  
(OFTWithFee.sol#1379) is not in mixedCase  
Parameter BaseOFTWithFee.estimateSendFee(uint16,bytes32,uint256,bool,bytes).\_useZro  
(OFTWithFee.sol#1379) is not in mixedCase  
Parameter  
BaseOFTWithFee.estimateSendFee(uint16,bytes32,uint256,bool,bytes).\_adapterParams  
(OFTWithFee.sol#1379) is not in mixedCase  
Parameter  
BaseOFTWithFee.estimateSendAndCallFee(uint16,bytes32,uint256,bytes,uint64,bool,bytes).\_ds  
tChainId (OFTWithFee.sol#1383) is not in mixedCase  
Parameter  
BaseOFTWithFee.estimateSendAndCallFee(uint16,bytes32,uint256,bytes,uint64,bool,bytes).\_to  
Address (OFTWithFee.sol#1383) is not in mixedCase  
Parameter  
BaseOFTWithFee.estimateSendAndCallFee(uint16,bytes32,uint256,bytes,uint64,bool,bytes).\_a  
mount (OFTWithFee.sol#1383) is not in mixedCase  
Parameter  
BaseOFTWithFee.estimateSendAndCallFee(uint16,bytes32,uint256,bytes,uint64,bool,bytes).\_pa  
yload (OFTWithFee.sol#1383) is not in mixedCase  
Parameter  
BaseOFTWithFee.estimateSendAndCallFee(uint16,bytes32,uint256,bytes,uint64,bool,bytes).\_ds  
tGasForCall (OFTWithFee.sol#1383) is not in mixedCase  
Parameter  
BaseOFTWithFee.estimateSendAndCallFee(uint16,bytes32,uint256,bytes,uint64,bool,bytes).\_us  
eZro (OFTWithFee.sol#1383) is not in mixedCase  
Parameter  
BaseOFTWithFee.estimateSendAndCallFee(uint16,bytes32,uint256,bytes,uint64,bool,bytes).\_ad  
apterParams (OFTWithFee.sol#1383) is not in mixedCase  
Reference:  
[https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c  
onventions](https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions)  
INFO:Detectors:  
ExcessivelySafeCall.slitherConstructorConstantVariables() (OFTWithFee.sol#397-467) uses  
literals with too many digits:  
- LOW\_28\_MASK = 0x00000000ff  
(OFTWithFee.sol#398)  
Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits>  
INFO:Slither:OFTWithFee.sol analyzed (21 contracts with 93 detectors), 163 result(s) found

# Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

## OFTWithFee.sol

### Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

more

Pos: 610:8:

This on local calls:

Use of "this" for local functions: Never use "this" to call functions in the same contract, it only consumes more gas than normal local calls.

more

Pos: 1360:35:

### Similar variable names:

OFTCoreV2.\_sendAck(uint16,bytes,uint64,bytes) : Variables have very similar names "amountSD" and "amount". Note: Modifiers are currently not considered by this static analysis.

Pos: 1294:8:

### Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 1509:8:

### Data truncated:

Division of integer values yields an integer value again. That means e.g.  $10 / 100 = 0$  instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 1377:24:

## Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

### OFTWithFee.sol

```
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:864
Error message for require is too long
Pos: 9:879
Error message for require is too long
Pos: 9:904
Check result of "send" call
Pos: 9:906
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:923
Code contains empty blocks
Pos: 53:1022
Error message for require is too long
Pos: 9:1064
Error message for require is too long
Pos: 9:1084
Error message for require is too long
Pos: 9:1085
Error message for require is too long
Pos: 9:1115
Error message for require is too long
Pos: 9:1121
Explicitly ma
Code contains empty blocks
Pos: 101:1461
Error message for require is too long
Pos: 9:1470
Error message for require is too long
Pos: 9:1476
Error message for require is too long
Pos: 9:1508
```

### Software analysis result:

This software reported many false positive results and some are informational issues. So, those issues can be safely ignored.



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

**Email: [audit@EtherAuthority.io](mailto:audit@EtherAuthority.io)**