PRIVACY POOLS SMART CONTRACTS AUDIT REPORT

OXORIO

MARCH 18, 2025

1 EXECUTIVE SUMMARY

1.1 EXECUTIVE SUMMARY

This document presents the smart contracts security audit conducted by Oxorio for Privacy Pool Smart Contracts.
Privacy Pool is a blockchain protocol that enables private asset transfers. Users can deposit funds publicly and partially withdraw them privately, provided they can prove membership in an approved set of addresses.

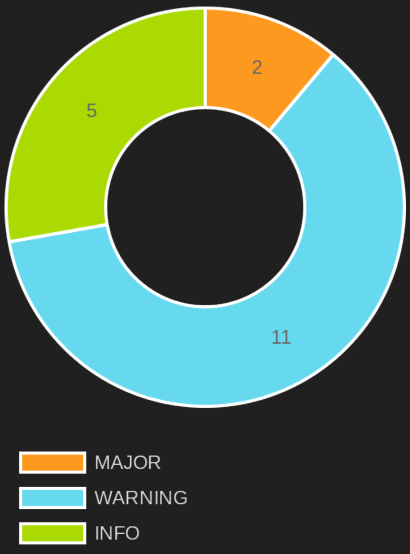
The audit process involved a comprehensive approach, including manual code review, automated analysis, and extensive testing and simulations of the curcuits to assess the project's security and functionality. The audit covered a total of 7 contracts, encompassing 408 lines of code. For an in-depth explanation of used the smart contract security audit methodology, please refer to the Security Assessment Methodology section of this document.

1.2 SUMMARY OF FINDINGS

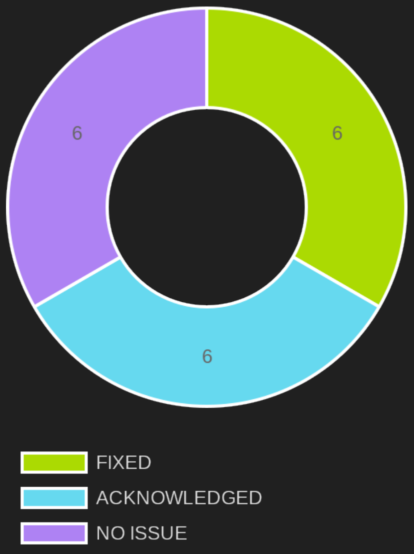
The table below provides a comprehensive summary of the audit findings, categorizing each by status and severity level. For a detailed description of the severity levels and statuses of findings, see the Findings Classification Reference section.

Detailed technical information on the audit findings, along with our recommendations for addressing them, is provided in the Findings Report section for further reference.

Severity	TOTAL	NEW	FIXED	ACKNOWLEDGED	NO ISSUE
CRITICAL	**0**	0	0	0	0
MAJOR	**2**	0	1	1	0
WARNING	**11**	0	3	3	5
INFO	**5**	0	2	2	1
TOTAL	**18**	**0**	**6**	**6**	**6**



Issue distribution by severity



Issue distribution by status

2 AUDIT OVERVIEW

CONTENTS

2.1 DISCLAIMER

At the request of the client, Oxorio consents to the public release of this audit report. The information contained herein is provided "as is" without any representations or warranties of any kind. Oxorio disclaims all liability for any damages arising from or related to the use of this audit report. Oxorio retains copyright over the contents of this report.

This report is based on the scope of materials and documentation provided to Oxorio for the security audit as detailed in the Executive Summary and Audited Files sections. The findings presented in this report may not encompass all potential vulnerabilities. Oxorio delivers this report and its findings on an as-is basis, and any reliance on this report is undertaken at the user’s sole risk. It is important to recognize that blockchain technology remains in a developmental stage and is subject to inherent risks and flaws.

This audit does not extend beyond the programming language of smart contracts to include areas such as the compiler layer or other components that may introduce security risks. Consequently, this report should not be interpreted as an endorsement of any project or team, nor does it guarantee the security of the project under review.

THE CONTENT OF THIS REPORT, INCLUDING ITS ACCESS AND/OR USE, AS WELL AS ANY ASSOCIATED SERVICES OR MATERIALS, MUST NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE. Third parties should not rely on this report for making any decisions, including the purchase or sale of any product, service, or asset. Oxorio expressly disclaims any liability related to the report, its contents, and any associated services, including, but not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Oxorio does not warrant, endorse, or take responsibility for any product or service referenced or linked within this report.

For any decisions related to financial, legal, regulatory, or other professional advice, users are strongly encouraged to consult with qualified professionals.

2.2 PROJECT BRIEF

Title	Description
Client	Privacy Pools
Project name	Privacy Pools smart contracts
Category	privacy, asset management
Repository	github.com/0xbow-io/privacy-pools-core
Documentation	README.md
Initial commit	`2d4627ba55743d17ff62a2856d93ef7cc926fc64`
Final commit	`532eaa8ed151f06697249d400926d17adf442d8e`
Languages	Solidity
Lead Auditor	Alexander Mazaletskiy - am@oxor.io
Project Manager	Elena Kozmiryuk - elena@oxor.io

2.3 PROJECT TIMELINE

The key events and milestones of the project are outlined below.

Date	Event
February 23, 2025	Client approached Oxorio requesting an audit.
February 25, 2025	The audit team commenced work on the project.
March 3, 2025	Submission of the comprehensive report.
March 11, 2025	Client feedback on the report was received.
March 18, 2025	Submission of the final report incorporating client’s verified fixes.

2.4 AUDITED FILES

The following table contains a list of the audited files. The scc tool was used to count the number of lines and assess complexity of the files.

	File	Lines	Blanks	Comments	Code	Complexity
1	packages/contracts/src/contracts/Entrypoint.sol	391	71	150	**170**	31%
2	packages/contracts/src/contracts/implementations/PrivacyPoolComplex.sol	66	10	33	**23**	22%
3	packages/contracts/src/contracts/implementations/PrivacyPoolSimple.sol	55	6	31	**18**	17%
4	packages/contracts/src/contracts/lib/Constants.sol	9	2	1	**6**	0%
5	packages/contracts/src/contracts/lib/ProofLib.sol	167	16	100	**51**	0%
6	packages/contracts/src/contracts/PrivacyPool.sol	184	38	78	**68**	28%
7	packages/contracts/src/contracts/State.sol	185	32	81	**72**	25%
	Total	**408**	**175**	**474**	**408**	**24%**

Lines: The total number of lines in each file. This provides a quick overview of the file size and its contents.

Blanks: The count of blank lines in the file.

Comments: This column shows the number of lines that are comments.

Code: The count of lines that actually contain executable code. This metric is essential for understanding how much of the file is dedicated to operational elements rather than comments or whitespace.

Complexity: This column shows the file complexity per line of code. It is calculated by dividing the file's total complexity (an approximation of cyclomatic complexity that estimates logical depth and decision points like loops and conditional branches) by the number of executable lines of code. A higher value suggests greater complexity per line, indicating areas with concentrated logic.

2.5 PROJECT OVERVIEW

The protocol enables users to deposit assets publicly and withdraw them privately, provided they can prove membership in an approved set of addresses. Each supported asset (native or ERC20) has its own dedicated pool contract that inherits from a common `PrivacyPool` implementation.

Deposit Flow

When a user deposits funds, they:

1. Generate commitment parameters (nullifier and secret)
2. Send the deposit transaction through the Entrypoint
3. The Entrypoint routes the deposit to the appropriate pool
4. The pool records the commitment in its state tree
5. The depositor receives a deposit identifier (label) and a commitment hash

Withdrawal Flow

To withdraw funds privately, users:

1. Generate a zero-knowledge proof demonstrating:
2. Ownership of a valid deposit commitment
3. Membership in the approved address set
4. Correctness of the withdrawal amount
5. Submit the withdrawal transaction through a relayer
6. The pool verifies the proof and processes the withdrawal
7. A new commitment is created for the remaining funds (even if it is zero)

Emergency Exit (`ragequit`)

The protocol implements a ragequit mechanism that allows original depositors to withdraw their funds directly for non ASP approved funds. This process:

1. Requires the original deposit label
2. Bypasses the approved address set verification
3. Can only be executed by the original depositor
4. Withdraws the full commitment amount

Core Contracts

`State.sol`
The base contract implementing fundamental state management:

- Manages the Merkle tree state using LeanIMT
- Tracks tree roots with a sliding window (30 latest roots)

- Records used nullifiers to prevent double spending
- Maps deposit labels to original depositors
- Implements tree operations

`PrivacyPool.sol`
An abstract contract inheriting from State.sol that implements the core protocol logic:

Standard Operations:

- Deposit processing (through Entrypoint only)
- Withdrawal verification and processing
- Wind down mechanism for pool deprecation
- Ragequit mechanism for non-approved withdrawals
- Abstract methods for asset transfers

Pool Implementations

`PrivacyPoolSimple.sol`
Implements `PrivacyPool` for native asset:

- Handles native asset deposits through `payable` functions
- Implements native asset transfer logic
- Validates transaction values

`PrivacyPoolComplex.sol`
Implements `PrivacyPool` for ERC20 tokens:

- Manages token approvals and transfers
- Implements safe ERC20 operations

Protocol Coordination

`Entrypoint.sol`
Manages protocol-wide operations:

- Routes deposits to appropriate pools
- Maintains the approved address set (ASP)
- Processes withdrawal relays
- Handles fee collection and distribution
- Manages pool registration and removal
- Controls protocol upgrades and access control

Supporting Libraries

`ProofLib.sol`
Handles accessing a proof signals values.

2.6 FINDINGS BREAKDOWN BY FILE

This table provides an overview of the findings across the audited files, categorized by severity level. It serves as a useful tool for identifying areas that may require attention, helping to prioritize remediation efforts, and provides a clear summary of the audit results.

File	TOTAL	CRITICAL	MAJOR	WARNING	INFO
packages/contracts/src/contracts/Entrypoint.sol	**13**	0	1	7	5
packages/contracts/src/contracts/PrivacyPool.sol	**5**	0	1	4	0

2.7 CONCLUSION

A comprehensive audit was conducted on 7 contracts, revealing no critical and 2 major issues. However, several warnings and informational notes were identified. The audit identified vulnerabilities, including potential fund loss, privacy breaches, and inefficiencies in fee calculations and code optimization.

Following our initial audit, Privacy Pools worked closely with our team to address the identified issues. The proposed changes aim to strengthen protocol security, improve efficiency, and ensure seamless user experience. Key recommendations include adding validation checks, optimizing code, and ensuring compatibility between deposited and withdrawn values to enhance security and maintain user privacy.

As a result, the project has passed our audit. Our auditors have verified that the Privacy Pools Smart Contracts, as of audited commit 532eaa8ed151f06697249d400926d17adf442d8e, operates as intended within the defined scope, based on the information and code provided at the time of evaluation. The robustness of the codebase has been significantly improved, meeting the necessary security and functionality requirements established for this audit.

3 FINDINGS REPORT

OXORIO

3.1 CRITICAL

No critical issues found.

3.2 MAJOR

M-01	Sending ETH or tokens to the zero address in `Entrypoint`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
Entrypoint.sol	contract `Entrypoint` > function `relay`	157-160

Description

In the function `relay` of the `Entrypoint` contract, it is possible to pass withdrawal data with empty values for `recipient = address(0)` and `feeRecipient = address(0)`. In this case, ETH or tokens would be sent to the zero address, while the `nullifierHash` is marked as spent, resulting in the permanent loss of funds.

Recommendation

We recommend adding a check to ensure that `recipient` and `feeRecipient` are not equal to `address(0)`.

Update

Fixed at `016a949f53f0493388a6877529f28774ef054a8e`

M-02	No option to withdraw privately in an edge case in `PrivacyPool`
Severity	**MAJOR**
Status	• ACKNOWLEDGED

Location

File	Location	Line
PrivacyPool.sol	contract `PrivacyPool` > function `withdraw`	119

Description

In the `withdraw` function of the `PrivacyPool` contract, there is no way to perform a withdrawal if `_merkleTree.depth` reaches `MAX_TREE_DEPTH`. This results in a scenario where users cannot withdraw funds privately from the pool. Once `_merkleTree.depth` reaches `MAX_TREE_DEPTH`, the only available option to withdraw funds is `ragequit` function. However, a `ragequit` withdrawal can only be made to the original depositor's address, thereby breaking privacy.

Recommendation

We recommend adding a separate `fullWithdraw` function that can only be called when `_merkleTree.depth` has reached `MAX_TREE_DEPTH`. This function should allow users to withdraw funds similarly to `withdraw`, but without adding a new commitment to the Merkle tree.

Update

Privacy Pools Response

This is practically impossible. It requires 2**32 deposits and Ethereum L1 has a total of 2.7 billion txs.

3.3 WARNING

W-01	Potential deanonymization in `PrivacyPool`
Severity	**WARNING**
Status	• NO ISSUE

Location

File	Location	Line
PrivacyPool.sol	contract `PrivacyPool` > function `ragequit`	139

Description

In the `ragequit` function of the `PrivacyPool` contract, there is a scenario where a link between the `depositor` and the `recipient` can be established:

- The user deposits 1 ETH by calling the `deposit` function.
- The user withdraws part of the funds (0.5 ETH) by calling the `withdraw` function.
- For some reason, the user withdraws the remaining funds (0.5 ETH) by calling the `ragequit` function with a `commitmentHash` that was previously publicly created during the `withdraw` process.

As a result, the shared `commitmentHash` links the calls to the `withdraw` and `ragequit` functions, while the shared `label` links the calls to the `deposit` and `ragequit` functions. Consequently, the `depositor` address used in the `deposit` function and the `recipient` address used in the `withdraw` function become linked.

Recommendation

We recommend analyzing this situation and performing the necessary code refactoring.

Update

Privacy Pools Response

Intended by the ASP operator.

W-02	Missing relay data checks in `Entrypoint`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
Entrypoint.sol	contract `Entrypoint` > function `relay`	130
Entrypoint.sol	contract `Entrypoint` > function `relay`	155

Description

In the `relay` function of the `Entrypoint` contract, there are missing checks:

- There is no validation to ensure that `feeRecipient` is not equal to `recipient`. This could lead to funds being sent to a `recipient` address that matches the relayer's `feeRecipient` address.
- There is no validation for the `relayFeeBPS` parameter, which could allow scenarios where `relayFeeBPS` is set to 100%, causing all funds to be fully deducted.

Recommendation

We recommend adding the missing relay data checks mentioned above.

Update

Fixed at `016a949f53f0493388a6877529f28774ef054a8e`

W-03	Missing new pool checks in `Entrypoint`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
Entrypoint.sol	contract `Entrypoint` > function `registerPool`	191

Description

In the `registerPool` function of the `Entrypoint` contract, there are missing checks:

- There is no validation to ensure that the pool is not dead, which could allow the addition of a pool that was previously deactivated.
- There is no validation for the `pool.ENTRYPOINT` parameter when adding a new pool. This could result in a pool being added with a different `entrypoint`.

Recommendation

We recommend adding the missing new pool checks mentioned above.

Update

Partly fixed at `016a949f53f0493388a6877529f28774ef054a8e` - validation to ensure that the pool is not dead is added.

Privacy Pools Response

This isn't a vulnerability (missing validation for the `pool.ENTRYPOINT` parameter) because it wouldn't possibly incur in user losing funds, just a misconfiguration that can be updated. Nevertheless, added a new check for the pool's configured Entrypoint address. PR for this is here.

W-04	Token decimals not considered in fee calculation in `Entrypoint`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
Entrypoint.sol	contract `Entrypoint` > function `_deductFee`	359

Description

In the `_deductFee` function of the `Entrypoint` contract, the token's `decimals` value is not considered when calculating relayer fee value. The current maximum fee BPS is `10_000`, which represents 100%. However, some tokens, such as GUSD, have a low `decimals` value. For such tokens, with the current BPS structure, the relayer fee may round to zero for small transfer amounts:

- A user wants to withdraw 1 GUSD (`decimals = 2`).
- The relayer fee is set to 0.1% (`_feeBPS = 10`).
- Fee calculation: `1 * 10**2 * 10 / 10_000 = 0`.

Recommendation

We recommend considering token decimals when calculating relay fees and reviewing the BPS handling logic.

Update

Privacy Pools Response

Tokens with less than 6 decimals won’t be used. And the relayer can drop any withdrawal request if it’s not profitable.

W-05	Shared `ASP` tree across all pools in `Entrypoint`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
[Entrypoint.sol]()	contract `Entrypoint`	52

Description

In the `Entrypoint` contract, a single `associationSets` is used for all registered pools. Since the `ASP` tree is limited by `MAX_TREE_DEPTH`, this could potentially cause the `ASP` tree to reach `MAX_TREE_DEPTH` more quickly than if each pool had its own `ASP` tree.

Recommendation

We recommend adding a separate `associationSets` for each pool.

Update

Privacy Pools Response

2**32 will suffice for 0xbow's Pools in this version of the protocol (and probably next ones).

W-06	Missing validation for `postman` and `owner` in `Entrypoint`
Severity	**WARNING**
Status	• NO ISSUE

Location

File	Location	Line
Entrypoint.sol	contract `Entrypoint` > function `initialize`	68-69

Description

In the `initialize` function of the `Entrypoint` contract, there is no validation to ensure that `postman` and `owner` are not equal to each other or to `msg.sender`. This could allow the deployer (`msg.sender`) to retain full control or make `postman` the `owner` as well. Additionally, the `owner` has the ability to update the `implementation` of the proxy contract.

Recommendation

We recommend adding checks to ensure that `postman` and `owner` are not equal to each other or to `msg.sender`.

Update

Privacy Pools Response

It'd be a valid configuration if desired.

W-07	Missing minimum amount for withdrawals in `PrivacyPool`
Severity	**WARNING**
Status	• NO ISSUE

Location

File	Location	Line
PrivacyPool.sol	contract `PrivacyPool` > function `withdraw`	122

Description

In the `withdraw` function of the `PrivacyPool` contract, there is no minimum withdrawal amount limitation. This could potentially lead to an attack where an attacker fills the commitment Merkle tree with dust commitments. As a result, the tree could reach `MAX_TREE_DEPTH`, preventing further deposits into the pool.

Recommendation

We recommend adding a minimum withdrawal amount limitation to prevent the tree from being filled with dust commitments.

Update

Privacy Pools Response

Minimums are enforced onchain only in deposits. When using the relay method, the only check is for amount equal zero. The relayer is the one choosing which requests to relay, thus choosing his minimum. Direct withdrawals are supposed to not have a minimum so users can rotate their commitment secrets in case of leakage.

W-08	Missing check that `processooor` is not equal to `depositor` in `PrivacyPool`
Severity	**WARNING**
Status	• NO ISSUE

Location

File	Location	Line
PrivacyPool.sol	contract `PrivacyPool` > modifier `validWithdrawal`	43

Description

In the `validWithdrawal` modifier of the `PrivacyPool` contract, there is no validation to ensure that `processooor` is not equal to `depositor`. This could lead to deanonymization. Moreover, there is a separate function, `ragequit`, specifically for withdrawals to the `depositor` address.

Recommendation

We recommend adding a check to ensure that `processooor` is not equal to `depositor`.

Update

Privacy Pools Response

Unnecessarily restrictive.

W-09	Only the latest `ASP` root is used for `ASP` inclusion proof verification in `PrivacyPool`
Severity	**WARNING**
Status	• NO ISSUE

Location

File	Location	Line
PrivacyPool.sol	contract `PrivacyPool` > modifier `validWithdrawal`	57

Description

In the `validWithdrawal` modifier of the `PrivacyPool` contract, the proof verification checks that only the latest `ASP` root is used:

```
if (_proof.ASPRoot() != ENTRYPOINT.latestRoot()) revert IncorrectASPRoot();
```

However, a proof might have been generated for the `ASP` root at `index-1` just minutes before a new `associationSets` is added. This would result in the user's transaction reverting, even though the `associationSets` root at `index-1` still exists in the `Entrypoint` contract.

Recommendation

We recommend adding a check to ensure that `_proof.ASPRoot()` is included in the set of known `associationSets`.

Update

Privacy Pools Response

The main feature of the protocol is to be able to filter out funds that may be associated to malicious activity. Funds that were approved at some point, may be flagged later, and need to be removed from the approved deposits set. As the association set is not append-only, the last root must be used only, as the previous ones may include malicious deposits.

W-10	Deposit fee can be unexpected in `Entrypoint`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
Entrypoint.sol	contract `Entrypoint` > function `updatePoolConfiguration`	226-239

Description

In the `updatePoolConfiguration` function of the `Entrypoint` contract, the value of the `vettingFeeBPS` parameter is changed:

```
    // Update pool configuration with validation
    _setPoolConfiguration(_config, _minimumDepositAmount, _vettingFeeBPS);
```

However, this function can be used to execute a front-running attack on a user's deposit (intentionally or accidentally):

- A user submits a deposit of 100 ETH, and at the time of signing/sending the transaction, the `vettingFeeBPS` for the ETH pool is 1%.
- The owner front-runs the user's transaction by calling `updatePoolConfiguration` and setting `vettingFeeBPS` for the ETH pool to 5%.
- The user's deposit transaction is executed with an unexpected and higher fee than anticipated.

Recommendation

We recommend adding an additional parameter to the `deposit` function that limits the maximum fee percentage allowed when the deposit is executed.

Update

Privacy Pools Response

This goes in the same owner-goes-rogue category like upgrading a contract. We assume trust in the owner.

W-11	Missing check that recipient is not the zero address in `Entrypoint`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
Entrypoint.sol	contract `Entrypoint` > function `withdrawFees`	255

Description

In the `withdrawFees` function of the `Entrypoint` contract, assets stored on the `Entrypoint` contract are transferred to the `_recipient` address. However, there is no validation to ensure that `_recipient` is not the zero address, which could lead to asset loss.

Recommendation

We recommend adding a check to ensure that `_recipient` is not the zero address.

Update

Fixed at `016a949f53f0493388a6877529f28774ef054a8e`

3.4 INFO

I-01	Unable to call `relay` after pool removal in `Entrypoint`
Severity	**INFO**
Status	• NO ISSUE

Location

File	Location	Line
Entrypoint.sol	contract `Entrypoint` > function `removePool`	207

Description

In the `Entrypoint` contract, if a pool is removed using the `removePool` function, there is no longer a way to withdraw funds via the `relay` function. The only option left is to withdraw directly from the pool using `withdraw`.

Recommendation

We recommend moving the `relay` function to the pool contract itself.

Update

Privacy Pools Response

That's the purpose of removing the pool. Users can still withdraw directly.

I-02	Expression is evaluated twice in `Entrypoint`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
Entrypoint.sol	contract `Entrypoint` > function `relay`	160-166

Description

In the `relay` function of the `Entrypoint` contract, the expression `_withdrawnAmount - _amountAfterFees` is evaluated twice:

- When calling `_transfer`.
- When calling `emit WithdrawalRelayed`.

For optimization purposes, it would be better to store `_withdrawnAmount - _amountAfterFees` in a separate variable.

Recommendation

We recommend using a separate variable to store `_withdrawnAmount - _amountAfterFees` wherever applicable.

Update

Fixed at `016a949f53f0493388a6877529f28774ef054a8e`

I-03	Confusing constant value definitions in `Entrypoint`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
Entrypoint.sol	contract `Entrypoint`	40-43

Description

In the `Entrypoint` contract, constants are defined in the next way:

```
  // @notice keccak256('OWNER_ROLE')
  bytes32 internal constant _OWNER_ROLE =
0x6270edb7c868f86fda4adedba75108201087268ea345934db8bad688e1feb91b;
  // @notice keccak256('ASP_POSTMAN')
  bytes32 internal constant _ASP_POSTMAN =
0xfc84ade01695dae2ade01aa4226dc40bdceaf9d5dbd3bf8630b1dd5af195bbc5;
```

That makes it difficult to verify whether their values match the corresponding comments.

Recommendation

We recommend modifying the constant definitions to use `keccak256` directly in the code to ensure clarity and correctness:

```
  bytes32 internal constant _OWNER_ROLE = keccak256('OWNER_ROLE');
  bytes32 internal constant _ASP_POSTMAN = keccak256('ASP_POSTMAN');
```

Update

Fixed at `016a949f53f0493388a6877529f28774ef054a8e`

I-04	Redundant function parameter in `Entrypoint`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
Entrypoint.sol	contract `Entrypoint` > function `registerPool`	175

Description

In the `registerPool` function of the `Entrypoint` contract, the `_asset` parameter is first passed as a function parameter but is later retrieved from the pool contract and checked to be equal to the same value:

```
if (_asset != IERC20(_pool.ASSET())) revert AssetMismatch();
```

Since the asset value is already available within the pool contract, the `_asset` parameter can be removed, and the retrieved value can be used directly.

Recommendation

We recommend removing the `_asset` parameter and using the value obtained from the pool contract instead.

I-05	Hardcoded number in `Entrypoint`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
Entrypoint.sol	contract `Entrypoint` > function `_deductFee`	359

Description

In the `_deductFee` function of the `Entrypoint` contract, the hardcoded number `10_000` is used for fee calculation:

```
_afterFees = _amount - (_amount * _feeBPS / 10_000);
```

Recommendation

We recommend defining a constant with a meaningful name and assigning it the value `10_000`, then using this constant throughout the contract instead of the hardcoded number.

Update

Privacy Pools Response

We use the term BPS which by definition is a fraction of 10_000.

4. APPENDIX

OXORIO

4.1 SECURITY ASSESSMENT METHODOLOGY

Oxorio's smart contract security audit methodology is designed to ensure the security, reliability, and compliance of curcuits throughout their development lifecycle. Our process integrates the Smart Contract Security Verification Standard (SCSVS) with our advanced techniques to address complex security challenges. For a detailed look at our approach, please refer to the full version of our methodology. Here is a concise overview of our auditing process:

1. Project Architecture Review

All necessary information about the smart contract is gathered, including its intended functionality and dependencies. This stage sets the foundation by reviewing documentation, business logic, and initial code analysis.

2. Vulnerability Assessment

This phase involves a deep dive into the smart contract's code to identify security vulnerabilities. Rigorous testing and review processes are applied to ensure robustness against potential attacks.

This stage is focused on identifying specific vulnerabilities within the smart contract code. It involves scanning and testing the code for known security weaknesses and patterns that could potentially be exploited by malicious actors.

3. Security Model Evaluation

The smart contract’s architecture is assessed to ensure it aligns with security best practices and does not introduce potential vulnerabilities. This includes reviewing how the contract integrates with external systems, its compliance with security best practices, and whether the overall design supports a secure operational environment.

This phase involves a analysis of the project's documentation, the consistency of business logic as documented versus implemented in the code, and any assumptions made during the design and development phases. It assesses if the contract's architectural design adequately addresses potential threats and integrates necessary security controls.

4. Cross-Verification by Multiple Auditors

Typically, the project is assessed by multiple auditors to ensure a diverse range of insights and thorough coverage. Findings from individual auditors are cross-checked to verify accuracy and completeness.

5. Report Consolidation

Findings from all auditors are consolidated into a single, comprehensive express audit. This report outlines potential vulnerabilities, areas for improvement, and an overall assessment of the smart contract's security posture.

6. Reaudit of Revised Submissions

Post-review modifications made by the client are reassessed to ensure that all previously identified issues have been adequately addressed. This stage helps validate the effectiveness of the fixes applied.

7. Final express audit Publication

The final version of the express audit is delivered to the client and published on Oxorio's official website. This report includes detailed findings, recommendations for improvement, and an executive summary of the smart contract's security status.

4.2 FINDINGS CLASSIFICATION REFERENCE

4.2.1 Severity Level Reference

The following severity levels were assigned to the issues described in the report:

Title	Description
CRITICAL	Issues that pose immediate and significant risks, potentially leading to asset theft, inaccessible funds, unauthorized transactions, or other substantial financial losses. These vulnerabilities represent serious flaws that could be exploited to compromise or control the entire contract. They require immediate attention and remediation to secure the system and prevent further exploitation.
MAJOR	Issues that could cause a significant failure in the contract's functionality, potentially necessitating manual intervention to modify or replace the contract. These vulnerabilities may result in data corruption, malfunctioning logic, or prolonged downtime, requiring substantial operational changes to restore normal performance. While these issues do not immediately lead to financial losses, they compromise the reliability and security of the contract, demanding prioritized attention and remediation.
WARNING	Issues that might disrupt the contract's intended logic, affecting its correct functioning or making it vulnerable to Denial of Service (DDoS) attacks. These problems may result in the unintended triggering of conditions, edge cases, or interactions that could degrade the user experience or impede specific operations. While they do not pose immediate critical risks, they could impact contract reliability and require attention to prevent future vulnerabilities or disruptions.
INFO	Issues that do not impact the security of the project but are reported to the client's team for improvement. They include recommendations related to code quality, gas optimization, and other minor adjustments that could enhance the project's overall performance and maintainability.

4.2.2 Status Level Reference

Based on the feedback received from the client's team regarding the list of findings discovered by the contractor, the following statuses were assigned to the findings:

Title	Description
NEW	Waiting for the project team's feedback.

Title	Description
FIXED	Recommended fixes have been applied to the project code and the identified issue no longer affects the project's security.
ACKNOWLEDGED	The project team is aware of this finding and acknowledges the associated risks. This finding may affect the overall security of the project; however, based on the risk assessment, the team will decide whether to address it or leave it unchanged.
NO ISSUE	Finding does not affect the overall security of the project and does not violate the logic of its work.

4.3 ABOUT OXORIO

OXORIO is a blockchain security firm that specializes in curcuits, zk-SNARK solutions, and security consulting. With a decade of blockchain development and five years in smart contract auditing, our expert team delivers premier security services for projects at any stage of maturity and development.

Since 2021, we've conducted key security audits for notable DeFi projects like Lido, 1Inch, Rarible, and deBridge, prioritizing excellence and long-term client relationships. Our co-founders, recognized by the Ethereum and Web3 Foundations, lead our continuous research to address new threats in the blockchain industry. Committed to the industry's trust and advancement, we contribute significantly to security standards and practices through our research and education work.

Our contacts:

- oxor.io
- ping@oxor.io
- Github
- Linkedin
- Twitter

THANK YOU FOR CHOOSING

OXORIO