PRIVACY POOLS SMART CONTRACTS AUDIT REPORT

OXORIO

MAY 20, 2025

1 EXECUTIVE SUMMARY

OXORIO

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.

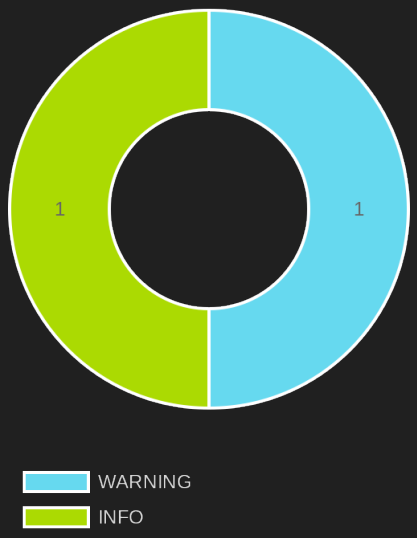
The audit process involved a comprehensive approach, including manual code review, automated analysis, and extensive testing and simulations of the curcuits to assess the project's security and functionality. The audit covered a total of 2 contracts, encompassing 794 lines of code. For an in-depth explanation of used the smart contract security audit methodology, please refer to the Security Assessment Methodology section of this document.

1.2 SUMMARY OF FINDINGS

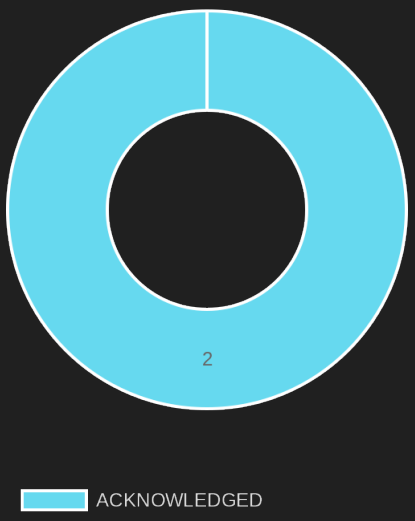
The table below provides a comprehensive summary of the audit findings, categorizing each by status and severity level. For a detailed description of the severity levels and statuses of findings, see the Findings Classification Reference section.

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	**0**	0	0	0	0
WARNING	**1**	0	0	1	0
INFO	**1**	0	0	1	0
TOTAL	**2**	**0**	**0**	**2**	**0**



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	`238e3594053becde68aa40e1e4cef6c0c46e68da`
Final commit	`238e3594053becde68aa40e1e4cef6c0c46e68da`
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
May 15, 2025	Client approached Oxorio requesting an audit.
May 18, 2025	The audit team commenced work on the project.
May 18, 2025	Submission of the comprehensive report.

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	403	78	142	**183**	33%
2	packages/contracts/src/interfaces/IEntrypoint.sol	391	54	242	**95**	0%
	Total	**794**	**132**	**384**	**278**	**22%**

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)

Pull Request Description

This pull request introduces changes to the to-be-upgraded `Entrypoint` contract. With this upgrade, a `usedPrecommitments` mapping is added to the contract, which tracks all used precommitments and prevents users from depositing using an already used precommitment, thus preventing stuck funds.

Introduced changes:

- state variable and checks added to Entrypoint
- added unit tests for the new revert case
- added an upgrade test using a forked environment from Ethereum Mainnet

An issue of this kind could theoretically happen to the secrets used on withdrawals, though it's way less probable because of the nature of the commitment system on withdrawals:

1. When withdrawing, partially or completely, a previous existing commitment is spent and a new one is created for the remaining value (even if it's zero). Because of this, when generating the withdrawal proof, the user must provide the nullifier of the spending commitment and a new nullifier for the new commitment. The `withdraw.circom` circuit does take this in count and will never produce a withdrawal proof with a new commitment with the same nullifier as the just spent one.
2. The secrets for deposits and withdrawals are generated differently. Both are created deterministically based on a `master_nullifier` and `master_secret`, but the data used as pre-image of the ultimate secret values is inherently different. For deposits, the secrets are image of `poseidon(master_key, pool_scope, deposit_index)`, while the ones of withdrawals are the image of `poseidon(master_key, deposit_label, withdrawal_index)`. This makes the chances of collision of deposit and withdrawal secrets almost non-existent.
3. When withdrawing, since a commitment is spent, its nullifier is marked as spent and that is checked for all further commitments to avoid double-spending. If a user were to submit two partial withdrawals of the same origin deposit quickly with an outdated state, one withdrawal transaction would be successful and the second one would revert, as both withdrawals would be trying to spend the same commitment, which can only be spent once.

This issue can not be used by a third actor in a malicious way whatsoever. Another account can see the chain and use your same pre-commitment for a deposit, but only the user who owns the master keys generated by the seed-phrase is the one that will be able to later spend the commitment.

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	**2**	0	0	1	1

2.7 CONCLUSION

A comprehensive audit was conducted on 2 contracts, revealing no critical and major issues. However, several warnings and informational notes were identified. The audit identified vulnerability with already used nullifier 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 `238e3594053becde68aa40e1e4cef6c0c46e68da`, 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

No major issues found.

3.3 WARNING

W-01	Potential nullifier reuse vulnerability in precommitment tracking in `EntryPoint`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
Entrypoint.sol	function `_handleDeposit`	317

Description

In the `_handleDeposit` function of the `EntryPoint` contract, a precommitment tracking mechanism has been implemented:

```
// Check if the `_precommitment` has already been used
if (usedPrecommitments[_precommitment]) revert PrecommitmentAlreadyUsed();
// Mark it as used
usedPrecommitments[_precommitment] = true;
```

This implementation only partially mitigates precommitment reuse risk. The vulnerability arises because a precommitment is computed as `hash(nullifier, secret)`. During withdrawal, the system marks the nullifier as spent. However, an attacker could use the same nullifier with different secrets to generate distinct precommitment hashes, effectively bypassing the precommitment reuse check.

This vulnerability creates a scenario where multiple deposits could be made using the same underlying nullifier, but only one could be successfully withdrawn (as the nullifier would be marked spent after the first withdrawal). The remaining deposits would become unwithdrawable, resulting in permanently locked funds.

Recommendation

We recommend reconsidering the deposit logic to address the fundamental nullifier reuse vulnerability. One possible solution is to implement a cryptographic mechanism to verify nullifier uniqueness without revealing it at deposit time.

Update

Client's response

To use the same nullifier as other commitment, the attacker must have access to the user's Privacy Pool seedphrase, which gives access to spending all user's commitments. On top of it, nullifiers and secrets are generated based on a random key-pair and more pseudo-random values like pool scope and deposit label, making it impossible for a nullifier to be generated twice accidentally. We won't take any action regarding this issue.

3.4 INFO

I-01	Non-optimal gas usage due to check order in `Entrypoint`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
Entrypoint.sol	contract `Entrypoint` > function `deposit`	113
Entrypoint.sol	contract `Entrypoint` > function `deposit`	125

Description

In the function `_handleDeposit` of contract `Entrypoint`, a check for the presence of `_precommitment` in the `usedPrecommitments` mapping was added. This function is invoked when calling the `deposit` functions for both native and non-native assets.

However, the `_precommitment` check is performed after storage is read to verify the `_pool` address:

```
function _handleDeposit(IERC20 _asset, uint256 _value, uint256 _precommitment) internal
returns (uint256 _commitment) {
  AssetConfig memory _config = assetConfig[_asset];
  IPrivacyPool _pool = _config.pool;
  if (address(_pool) == address(0)) revert PoolNotFound();

  if (usedPrecommitments[_precommitment]) revert PrecommitmentAlreadyUsed();
    // ...
```

and the asset transfer is executed in the `deposit` function:

```
function deposit(
  IERC20 _asset,
```

```
    uint256 _value,
    uint256 _precommitment
  ) external nonReentrant returns (uint256 _commitment) {
    _asset.safeTransferFrom(msg.sender, address(this), _value);
    _commitment = _handleDeposit(_asset, _value, _precommitment);
  }
```

This leads to unnecessary gas consumption if the `_precommitment` already exists.

Recommendation

We recommend considering moving the `_precommitment` check to the very beginning of the `deposit` function logic to optimize gas usage.

Update

Client's response

Will keep it as it is.

4. APPENDIX

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