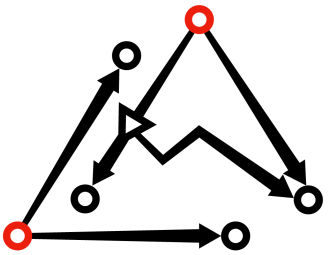

Least Authority

PRIVACY MATTERS

Zkopru zk-SNARK Circuits + Smart Contracts

Security Audit Report

Ethereum Foundation

Final Report Version: 22 June 2021

Table of Contents

Overview

Background

Ethereum Foundation has requested that Least Authority perform a security audit of the Zkopru (zk-optimistic-rollup) zk-SNARK Circuits and Smart Contracts.

Zkopru, a Zcash-Style privacy solution, is a layer-2 scaling solution for private transactions using zk-SNARK and optimistic rollup on the Ethereum Blockchain. It supports private transfer and private atomic swap within the layer-2 network between ETH, ERC-20, and ERC-721 at a low cost. In addition, with the pay-in-advance feature, it aims to allow users to withdraw assets from the layer-2 before the finalization.

Project Dates

- **January 11 - February 22**: Code review *(Completed)*
- **February 25**: Delivery of Initial Audit Report *(Completed)*
- **June 17 - 21:** Verification *(Completed)*
- **June 22:** Delivery of Final Audit Report *(Completed)*

Review Team

- Dominc Tarr, Security Researcher and Engineer
- Nathan Ginnever, Security Researcher and Engineer
- Mirco Richter, Cryptography Researcher and Engineer
- Jan Winkelmann, Cryptography Researcher and Engineer
- Rai Yang, Security Researcher and Engineer
- Steve Thakur, Cryptography Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Zkopru (zk-optimistic-rollup) zk-SNARK Circuits and Smart Contracts followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

The following code repositories are considered in-scope for the review:

- zk-SNARK Circuits and Smart Contracts: https://github.com/zkopru-network/zkopru/releases/tag/audit-v1

Specifically, we examined the Git revisions for our initial review:

`4236fc8a5cbf73b7f3860d87a1a447eea8d7abd4`

For the verification, we examined the Git revision:

`1e0883138b7348b7a230fb07f753ffac5ba7adae`

For the review, this repository was cloned for use during the audit and for reference in this report:

https://github.com/LeastAuthority/zkopru

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third party code, unless specifically mentioned as in-scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Zkopru Documentation: https://docs.zkopru.network/v/burrito/
- Zkopru (zk-optimistic-rollup) for Private Transactions: https://ethresear.ch/t/zkopru-zk-optimistic-rollup-for-private-transactions/7717

In addition, this audit report references the following documents:

- T. Kim, and R. Barbulescu, 2015, "Extended Tower Number Field Sieve: A New Complexity for the Medium Prime Case." *IACR Cryptol. ePrint Arch 2015/1027* [KB15]
- A. Menezes, P. Sakar, and S. Singh, 2016, "Challenges with Assessing the Impact of NFS Advances on the Security of Pairing-based Cryptography." *IACR Cryptol. ePrint Arch 2016/1102* [MSS16]
- R. Barbulescu, and S. Duquesne, 2017, "Updating key size estimations for pairings." *IACR Cryptol. ePrint Arch 2017/334* [BD17]
- T. Perrin, 2016, "Curves for pairings." [P16]
- Y. Sakemi, Ed. Lepidum, T. Kobayashi, T. Saito, NTT, R. Wahby, 2020, "Pairing-Friendly Curves." [SLK+20]
- E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, 2014, "Zerocash: Decentralized Anonymous Payments from Bitcoin (extended version)." [BCG+14]
- S. Bowe, K. Gurkan, and E. Tromer, 2018, "ZKProof Standards. Implementation Track Proceedings." [BGT18]
- D. Hopwood, 2018 "Zcon0 Circuit Optimisation handout." [H18]
- D. Hopwood, 2019 "Designing Efficient R1CS Circuits." [H19]
- J. Groth, 2016, "On the Size of Pairing-based Non-interactive Arguments." *IACR Cryptol. ePrint Arch 2016/260* [G16]
- E. Baker, 2020 "Recommendation for Key Management: Part 1 – General." *NIST Special Publication 800-57* [B20, Table 4]

Areas of Concern

Our investigation focused on the following areas:

- General
  - Correctness of the implementation;
  - Resistance to Distributed Denial of Service (DDoS), Reentrance, and similar attacks;
  - Inappropriate permissions and excess authority;
  - Performance problems or other potential impacts on performance;
  - Data privacy, data leaking, and information integrity;
  - Confidentiality of which coins have been spent or created;
- zk-SNARK Circuits
  - Common and case-specific implementation errors in the circuit code;
  - Requirement for the private key of a coin in order to spend it;
  - Possibility to construct a transaction pair where either both or neither transactions can be executed;
  - Limitation that coins cannot be created and can only be deposited;
  - Limitation that coins cannot be destroyed and can only be withdrawn;
- Smart Contracts

    - Adversarial actions and other attacks on the smart contracts;
    - Potential misuse and gaming of the smart contracts;
    - Attacks that impacts funds, such as the draining or the manipulation of funds;
    - Mismanagement of funds via transactions;
    - Alignment of incentive mechanisms to help prevent unwanted or unexpected behavior;
    - Vulnerabilities in the smart contracts code as well as secure interaction between the contracts and with related network components;
    - Proper management of encryption and signing keys;
        - Coins deposited by a user may only be moved by the holder of that user's private key;
        - Coins withdrawn from the circuit by a user may only be moved by the holder of that user's private key;
    - Protection against malicious attacks and other ways to exploit contracts;
    - For any invalid state transition, a fraud proof can be constructed that reverts that state transition;
    - Impossible to construct a state transition that does not reveal the state of the whole system after that state transition; and
- Anything else as identified during the initial analysis phase.

Findings

zk-SNARK Circuits

System Design

SNARK Statement

The Zkopru team's initial approach to the design of the zk-SNARK circuits was absent of a SNARK statement, which is a critical part of circuit design best practices and methodologies and should be a prerequisite to the coded implementation. Furthermore, a comprehensive and rigorous statement should adhere to the conventions of proper statement design, as recommended by [H19]. In this particular instance, the SNARK statement that was developed after the beginning of the audit contributed significantly to our understanding of how the system works. Without a proper statement, propositions would need to be extracted from the code, which results in the high likelihood for concluding incorrect assumptions about the circuit's function and increased difficulty for reviewers of the code. These factors introduce a significant risk that vulnerabilities are missed due to misunderstandings of the code's intention. We commend the Zkopru team for their quick development of the SNARK statement.

We recommend the Zkopru team further improve their statement to provide a link to the trusted setup computation, which clearly describes how the used common reference string is computed (e.g., version identifier, protocol used, number of participants, rounds, etc.). An accompanying theorem should also be written that proves the desired properties of the SNARK (e.g. [BCG+14], Section 4). Overall, more attention should be given to the accuracy of the specifications, as well as the mathematical and proofing aspects of the system design. As a result, we suggest that an extensive and rigorous statement definition be written describing the reasoning behind the design of the zk-SNARK circuits (Issue B) and providing a comprehensive list of security assumptions (Issue D).

A statement description should also specify design decisions and the assumptions made by the system (e.g., what information is revealed, what is hidden, and what information an UTXO owner needs to know in order to spend the UTXO or to generate a SNARK proof). As a best practice, the implementation of the code should follow the completion of the statement design or, at a minimum, be done in parallel. It is also critical that the documentation is maintained and updated regularly as an accurate point of reference for

the coded implementation, as inconsistency in the documentation and the implementation could result in confusion or errors. We encourage the Zkopru team to consider starting with the statement description in future SNARK circuits work.

EdDSA Signature Checks

During our review, we identified a code update that disabled EdDSA signature checks, which were previously enabled. This change likely resulted from a coding error accompanying a larger code refactoring commit. Disabled EdDSA signature checks allow for attackers to forge valid SNARK proofs, thus posing a critical security vulnerability. We recommend re-enabling signature checks, in addition to introducing tests to prevent errors in the existing code upon the introduction of code updates (Issue C).

BN254 Curve

The BN254 curve used by Zkopru is not sufficiently secure. Due to more recent advances in number theory, the BN254 curve is no longer considered to provide the security we have come to expect from cryptographic algorithms. As a result, we recommend that the Zkopru team implement the curve pairing-friendly curve BLS12-381 (Issue A).

Input Variables

The circuit contains several public and private input variables that are currently unused and intended for future atomic swap features. However, we do not foresee any negative security implications since these features are solely necessary binders for the trusted setup phase. This is particularly important since it avoids the need for a new trusted setup ceremony once the circuit has been updated to implement atomic swap features.

Dependencies

All of the circuit dependencies are widely utilized circom gadgets. The use of trusted and well maintained dependencies minimizes the potential security risks and malicious code introduced as a result of utilizing third-party code. We commend the Zkopru team for their selection of gadgets with security in mind and recommend that they continue to maintain and update dependencies, in addition to checking that they have been sufficiently audited.

Code Quality

The zk-SNARK circuits code base is organized and the code is well written and easy to read, which can be attributed to the adherence to development best practices and to the exceptional readability of the Circom programming language. In addition, the zk-SNARK circuits include some test coverage, which demonstrate the intended use for the circuits. However, additional tests to account for potential edge cases would help to further protect against malicious actions and unexpected behavior (Suggestion 2).

It should be noted that because Circom has no type-safety, we type checked the code during our review. Since this burden of type checking is transferred to reviewers of the code, it should be accounted for in the planning of future audits. Given that Circom is still in the early stages of development, coding best practices and style guides are still emerging and have not been firmly established. As a result, we did not evaluate the code against SNARK development standards for Circom.

Code Comments

We identified some comments in `zk_transaction.circom`, however, all other parts of the SNARK code base were absent of, and would greatly benefit from, comments describing their intended functionality, particularly for the low level gadgets. We recommend that additional comments be incorporated into the code base to ensure sufficient coverage, which would greatly improve the readability of the gadgets (Suggestion 1).

Scope

The scope of the zk-SNARK circuit audit provided sufficient coverage, encompassing all circuit components of the Zkopru system, with a few exceptions as suggested in this report. The zk-SNARK circuits implement a multi-asset commitment-nullifier scheme to achieve privacy in transactions. In order to verify the correctness of the implementation, we recommended that a SNARK statement be provided by the Zkopru team to compare the zk-SNARK circuits code against. With some consultation, the current SNARK statement was created and was subsequently included in the scope of the audit. This facilitated a successful review by our team and demonstrates their commitment to the security of the protocol.

In addition to performing the SNARK statement to code comparison, we tested the zk-SNARK against fringe cases and found no discrepancies.

However, validation of the SNARK statement was out of scope for this review. An audit to validate the SNARK statement can only be done in an accompanying mathematical paper containing a proper proof for all required properties. We recommend that the Zkopru team prepare a mathematical paper for all required properties and have it validated by a qualified team. In addition, while checking the correctness of the commitment scheme verifier implementation was out of scope for this audit, our team found it to be a useful reference in building a better understanding of the circuits' functions. In reviewing it for reference and understanding, we identified a critical vulnerability such that the uniqueness of UTXO nullifiers is not checked, which may potentially result in loss of funds (Issue J). As a result, we recommend an audit of the commitment scheme verifier (Suggestion 8).

Smart Contracts

System Design

The current centralized way of updating the consensus rules allows the operator to have complete control over verifications, opening the possibility of malicious state updates. We recommend replacing the centralized approach with a democratic and transparent solution, so that this attack surface will be closed to a malicious or compromised operator (Issue P).

The Zkopru smart contracts act as a verification judge on the main Ethereum chain for a Layer 2 UTXO chain using a combination of zero-knowledge and fraud proofs to provide security against double spends and privacy. As a result, the smart contracts are required to support challenges and zero-knowledge verification functions. The combination of systems leads to an extensive verification library containing both zero knowledge pairing checks and merkle inclusion proofs for challenges. In addition, it results in a more complex and costly verification process. A potential approach that can be further explored by the Zkopru team is to aggregate proofs, thus making verification more compact and less expensive. However, this is an area of active research and remains significantly under-developed.

Potential Attack Vectors

We identified several opportunities for front running attacks (Issue F, Issue G and Issue N). While a general remediation that can be easily implemented by the Zkopru team does not exist for front running in Ethereum, we provide specific recommendations for remediating the front running of fees in withdraw transactions (Issue G) and for front running burn auction bids (Issue N).

Additionally, we have found that a coordinator's URL/IP address is exposed to all other coordinators in the auction process, which can be exploited with a DDoS attack. Therefore, we suggest removing the requirement for coordinators to share their URL/IP information to join an auction, and using an anonymous routing protocol to keep this data confidential (Issue R).

Caller Fees

Currently, the caller fee is only rewarded on withdrawals that contain ETH assets. The withdrawal fee is not rewarded for non-ETH assets such as ERC-20 tokens and NFTs. As a result, we recommend adding a fee transfer to the caller when `eth=0` in order to resolve this error (Issue M).

Dependencies

The code uses inherited libraries clearly and we found it easy to navigate through the dependencies. The code is well written with minimal linting errors. We found some instances of duplicate code, which can lead to increased costs and human error when writing or reviewing code, which should be removed (Suggestion 4).

Code Quality

We found the code base to be organized and well written. In addition to manual review of the code, we examined the code using Solhint, an open source linter for Solidity code and identified no security critical issues in the results.

We found that the smart contracts generally adhere to Solidity best practices, however, we make note of several exceptions where deviations exist, as detailed below.

The check effects interaction pattern is not being adhered to, which may lead to re-entrancy attacks resulting in loss of funds (Issue H). We recommend addressing re-entrancy code patterns in several locations of the code, which results in the contract state being updated after, as opposed to before, calling external functions (Issue I).

Another potential attack surface is the missing uniqueness check of commitment nullifiers, which should be added when the proposer generates blocks. Without this, an attacker can claim the slash reward by submitting a double spend transaction, which results in the block proposer getting slashed if there is a redundant nullifier, causing a double spend when the transaction is challenged (Issue J).

At present, Zkopru smart contracts make use of outdated ERC-20 token interfaces, which are missing a safe transfer wrapper to anticipate the event that a token interface does not adhere to the standard of reverting in the case of failure. This can lead to older tokens being locked on the smart contract without the ability for Zkopru to interact with the tokens to make withdrawals (Issue O).

Finally, we recommend increasing the compiler version to at least 0.7.0 to incorporate newer fixes (Suggestion 5), adding address input sanitization (Suggestion 6), and addressing outstanding TODO comments (Suggestion 7).

Tests

The smart contracts include test coverage that is limited to success cases, and does not incorporate failures and errors. Furthermore, each validator should have corresponding tests, which are currently insufficient. We recommend increasing test coverage to help detect and prevent unintended behavior and edge cases, which may result in potential vulnerabilities (Suggestion 2).

Code Comments

The code is well commented and follows NatSpec guidelines for Solidity comments, which helps code reviewers and users easily understand the inputs and functionality of every method present, and therefore aid in identifying any potential issues. However, there are several functions that are difficult to follow and do not have comments describing their functionality or inputs. For example, `BurnAuction.sol` is insufficiently commented and, as a result, the Burn Auction consensus mechanism required considerable time and effort to examine and understand in depth. Increasing code comment coverage would help to

explain the intended functionality and facilitate easier review and more efficient review of the code (Suggestion 1).

Documentation

The project documentation provides an overview of the Zkopru system and the intended functionality for the smart contracts. However, the smart contracts interact with many off-chain components of the circuit system (e.g. coordinator, validator, SNARK verifier, wallet client, fullnode, etc.) and these interactions are not comprehensively documented. Furthermore, build and test documentation is missing and would provide a clear description of the setup process, which would avoid the potential for mistakes. As a result, we recommend creating detailed documentation of how the entire system interacts, which would provide a deeper appreciation and understanding of Zkopru. In addition, providing build and test documentation would prevent the potential for issues to arise during the initial setup phase (Suggestion 3).

Scope

The scope of the smart contracts audit included all security critical components of the smart contracts system. However, during our review of the smart contracts and their integration with the Layer 1 blockchain, we discovered issues in the client code, which was out of scope for this audit. In particular, we found no withdrawal note validity check when a prepayer receives an instant withdrawal request, which could result in loss of funds for the prepayer. This can be remediated by having the coordinator or prepayer add a validity check for the withdrawal note when receiving an instant withdrawal request (Issue K).

In addition, SNARK validation, which validates the SNARK proof in the transaction, is missed in on-chain/off-chain validation. This can result in invalid transactions passing the validation, thus allowing an attacker to generate a transaction resulting in the loss of funds in the smart contracts (Issue L). Since Zkopru is a Layer 2 blockchain, the security of the Layer 1 integration points and off-chain components, should be considered a priority. While we did not uncover any dependency concerns in the smart contracts, we recommend that off-chain components (i.e., coordinator, wallet client, validator, SNARK verifier, and synchronizer) be followed up with a security audit (Suggestion 8).

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
zk-SNARKS Circuits	
Issue A: The BN254 Curve Provides Insufficient Security	Unresolved
Issue B: Write a Proper Accompanying zk-SNARK Statement	Resolved
Issue C: Previously Correct Ownership Proof Disabled via Code Changes	Resolved
Issue D: Define the Desired zk-SNARK Properties and Write Proper Proofs	Partially Resolved
Issue E: Circuit Does Not Check the ERC-20 Sum Correctly (Known Issue)	Resolved
Suggestion 1: Increase Code Comments	Resolved

Smart Contracts	
Issue F: Front Running of Challenge Transactions	Unresolved
Issue G: Front Running of Fees in Withdraw Transactions	Unresolved
Issue H: Reentrancy Attacks	Resolved
Issue I: Fix Reentrancy Code Patterns	Resolved
Issue J: No Nullifier Uniqueness Check	Resolved
Issue K: No Withdrawal Note Validity Check (Out of Scope)	Resolved
Issue L: SNARK Validation Missed in On-chain/Off-chain Validation (Out of Scope)	Resolved
Issue M: No Caller Fee When Withdrawing Only Non-ETH Assets (ERC 20 Token and NFTs) (Known Issue)	Resolved
Issue N: Front Running of Burn Auction Bids	Partially Resolved
Issue O: Old ERC-20 Token Interfaces May Lead to Stuck Tokens Stuck Tokens Or Be Blocked From Use	Resolved
Issue P: Operator Has Total Authority Over State Verification	Resolved
Issue Q: Deposit Does Not Check For Registered Token	Partially Resolved
Issue R: Coordinator's URL/IP Address is Exposed in Auction Process, Which Can Be Exploited For a DDoS Attack	Partially Resolved
Suggestion 2: Increase Test Coverage	Unresolved
Suggestion 3: Expand System Documentation	Unresolved
Suggestion 4: Remove Duplicate Code	Resolved
Suggestion 5: Update Compiler Version	Resolved
Suggestion 6: Add Address Input Sanitization	Resolved
Suggestion 7: Address TODO Comments	Resolved
Suggestion 8: Audit Off-Chain Components	Unresolved

zk-SNARK Circuits

Issue A: The BN254 Curve Provides Insufficient Security

Location

`circuits/script/powers_of_tau_phase_1.sh`

Synopsis

Zkopru uses a variant of the BN254 curve, also called BN128. In 2016, advances in number theory led to a lower security estimate of that curve. Specifically, its security is now considered to be around 96 bits. This is significantly lower than the 112 bits currently required by NIST for new products.

Impact

Use of the BN254 curve reduces the security of the zk-SNARK scheme, such that the feasibility of computing valid, forged proofs cannot be ruled out. Such proofs would pass validation, yet violate the constraints imposed by the circuit. For example, a valid, forged proof could spend UTXOs arbitrarily.

Feasibility

Although the exact feasibility is difficult to estimate, the potential gains from a successful attack are high. This suggests that an attacker would have incentive to invest significant resources resulting in an increased risk that the feasibility is reasonable.

Technical Details

Attacks based on the Tower Number Field Sieve (TNFS) and the derivative exTNDS and SexTNFS have led to a reduced estimate of the security level of BN254. Consensus on the new estimate has not been reached by scholars and practitioners working on this issue, however, opinions range from 96 bits to 110 bits [KB15, MSS16, BD17, P16]. Regardless of where on this spectrum the real value falls, it is still too low. Even for applications that only need to remain secure until 2030, NIST requires a security level of at least 112 bits [B20, Table 4], which BN254 does not achieve.

Remediation

We recommend using the curve BLS12-381. According to the draft RFC on pairing-friendly curves [SLK+20], it has a security level of ~128 bits, which is above the 112 bits considered sufficient by NIST until 2030.

Status

The Berlin hard fork upgrade will no longer contain EIP-2537 and, as a result, 384 bit arithmetic in the EVM (i.e. EVM-384) is currently unavailable. Given this change, there is currently no efficient method for secure pairing based cryptography that is able to achieve at least 112 bits of security, as required by NIST for new products. Thus, a long-term remediation is not currently possible and we recommend that the Zkopru team continue to monitor developments with EIP-2537.

Verification

Unresolved.

Issue B: Write a Proper Accompanying zk-SNARK Statement

Location

`zkopru-network/protocol-specification/`

Synopsis

zk-SNARKS are short and computationally sound proofs for the existence of witnesses to given statements, able to hide parts of the witness from any verifier. In order to correctly perform a security audit on a zk-SNARK implementation, it is therefore fundamental to start with a clear and formal definition of the associated zk-SNARK statement.

Impact

A well written SNARK statement improves the security of the SNARK by making it easier to reason abstractly about the security properties of the zk-SNARK itself. Statements that are not well designed carry the risk of unintended solutions, which will enable a malicious prover to find valid proofs for unintended system behavior. Since zero-knowledge is involved, those proofs, that are valid but malicious, would be potentially very difficult to detect. In addition, statements are the foundation for writing security proofs for zk-SNARK properties.

Additionally, when lacking a statement, there is no foundation to compare the implementation against. It is insufficient to have the statement implicit in the code, as this would force a circular approach for reviewers, consisting of comparing the code against a statement that is implicit in the code.

Remediation

We recommend that the Zkopru team write an extensive and rigorous statement definition, adhering to the best practices of SNARK development as outlined for example in [BGT18], section "Correctness and Trust / Considerations", [H18], or [H19]. In addition, we suggest instituting regular documentation reviews to ensure the documentation remains up to date and consistent with the implementation. The statement should contain a list of all assumptions made by the system. In addition, a clear description of how the common reference string was or will be computed should be given, and reference all data known related to the trusted setup accordingly.

The statement documentation created by the Zkopru team during the course of this audit is helpful, and we identified no deviations in comparison to the actual functionality of the gadgets apart from Issue C and Issue E.

Status

The Zkopru team developed a specification for the protocol, which included a zk-SNARK statement. This aided their efforts in identifying and resolving new critical vulnerabilities.

Verification

Resolved.

Issue C: Previously Correct Ownership Proof Disabled via Code Changes

Location

`circuits/lib/ownership_proof.circom#L10`

Synopsis

Zkopru is supposed to check the ownership of any UTXO, by enforcing a valid EdDSA check on the BabyJubJub curve. To do so, it utilizes Circom's `EdDSAPoseidonVerifier()` gadget, which has an input variable that is used to enable or disable EdDSA signature checks during proof generation. In Zkopru, the intended mode is to always enable the signature check, but in the commit we audited, we found this variable to be set in such a way that signature checks were always disabled.

On inspection of the commit history, we discovered that previous commits correctly enabled the signature check as expected. The vulnerability was introduced in a commit, most likely due to a need for large code refactoring, since Circom and `snarkjs` both underwent prior major changes.

Impact

Severe. Without enabled signature checks, the ownership proof will always be valid, regardless of the correctness of the signature. A malicious proofer might use this vulnerability to forge valid zk-SNARK

proofs. As the signature is private data and hence not revealed to the public, this kind of behaviour would be potentially very difficult to detect.

Feasibility

Forging an ownership proof with the signature check disabled is a trivial effort. Whether this can be used to transfer value from UTXOs that are not owned by the attacker is not easy to estimate, as this depends on the ability of the attacker to provide all the other private data required to generate a valid proof.

Technical Details

In the gadget `OwnershipProof()`, the `EdDSAPoseidonVerifier()` subgadget's input signal `eddsa.enabled` is defined as the constant `0`, whereas it should be defined as `1`. We note that the correct setting is used in commits preceding the commit under investigation for this audit.

Remediation

Set `eddsa.enabled` to `1` so that the ownership proof works as intended. In addition, extend test coverage, to ensure that no future code update can re-introduce this vulnerability.

Status

The Zkopru team changed `eddsa.enabled` to `1` and added an ownership test catching the issue upon reintroduction.

Verification

Resolved.

Issue D: Define the Desired zk-SNARK Properties and Write Proper Proofs

Location

`zkopru-network/protocol-specification/`

Synopsis

In SNARK statement design, attention should be given to the accuracy of the specifications, as well as the mathematical and proofing aspects of system design. In doing so, a comprehensive list of security assumptions should be clearly specified that can then be compared against the coded implementation.

Zkopru uses the zk-SNARK Groth16 to provide a range proof while hiding the account balance. The Groth16 SNARK hinges on certain cryptographic assumptions about elliptic curve pairings and requires a trusted setup. The common reference string (CRS) is usually constructed through a secure multi-party computation (MPC). In our audit, we found that the documentation did not explicitly mention the cryptographic assumptions. Details about the construction of the CRS were also missing.

Technical Details

During our review, we first audited the zk-SNARK implementation for coding errors and then compared it to the statement that the Zkopru team provided to us upon request (see the section System Design). However, we identified four problems with the statement provided:

1. The assumptions that the system relies on are not explicit.
2. The properties that the SNARK is intended to guarantee are not explicit.
3. There are no proper proofs for the guaranteed properties.
4. There is not any reference to the common reference string that is used in the system and there is not a description about how that CRS was computed, or why it can be trusted.

In the Zkopru transaction documentation, the following is stated:

> *"Zkopru achieves privacy using the commitment-nullifier scheme. It means that a zk transaction spends a UTXO while not revealing which note has been used."*

However, from a security point of view, this is too vague. For example, exactly what is guaranteed to stay private and under what assumptions is unclear. Additionally, it is unknown to which commitment-nullifier scheme the documentation refers to and there is not a proof for the claimed properties.

Mitigation

If writing a proper proof is untenable at this point in the Zkopru project, we recommend that at least the statement description is expanded by providing a clear list of assumptions, referencing the common reference string trusted setup phase and making the claimed guarantees as explicit as possible. At minimum, some high level reasoning should be given about why it is impossible to unreveal any spent UTXO.

Remediation

In [BCG+14], the authors defined their version of what they called a "*decentralized anonymous payment scheme [DAP scheme]*", and then proved its critical properties like *ledger indistinguishability*, *transaction non-malleability* and *balance invariance* (Section 3.4, [BCG+14]).

We recommend to first make the Zkopru adaptation of a DAP scheme explicit, then derive an analog to theorem 4.1 in [BCG+14] and prove equivalent properties, which could be called something like *contract state indistinguishability*, *transaction non-malleability* and *multi-asset-balance invariance*. In contrast to [BCG+14], more than one asset type is involved, so a proper definition of something we could call a *multi-asset commitment-nullifier scheme* is needed.

Status

The Zkopru team developed a specification for the protocol, which includes a zk-SNARK statement. However, a formal specification of the SNARK properties and security proofs is still missing. The trusted setup has been completed on the Zkopru website and they have noted that a verification script will soon be published.

Verification

Partially Resolved.

Issue E: Circuit Does Not Check the ERC-20 Sum Correctly (Known Issue)

Location

`circuits/lib/zk_transaction.circom#L255`

Synopsis

The current circuit only checks the sum of ERC-20 tokens for the token addresses included in the input notes, but not for others, which could enable a malicious actor to drain funds.

Impact

Severe. An attack could result in token loss, with the attacker potentially draining funds. Moreover, an attack of this type would be potentially undetectable, since tokens would only be seen on the smart contract side. The amount of tokens that can be potentially stolen is limited to the tokens that are rolled up.

Preconditions

Some amount of tokens need to be rolled up in order to be stolen.

Technical Details

The following is an input that works as intended:

```
inputs: [ { token: DAI, amount: 10 }, {token: WETH, amount: 1}],

outputs: inputs: [ { token: DAI, amount: 5 }, { token: DAI, amount: 5 },
{token: WETH,amount: 1}],
```

However, this also returns `true` in the following, unintended case:

```
inputs: [ { token: DAI, amount: 10 }, {token: WETH, amount: 1}],

outputs: inputs: [ { token: DAI, amount: 5 }, { token: DAI, amount: 5 },
{token: WETH, amount: 1}, { token: USDC, amount: 1000000 }],
```

Remediation

Modify the circuit to enforce that outputs do not contain ERC-20 addresses that are not part of any spend note.

Status

This issue was found and resolved by the Zkopru team in writing the zk-SNARK statement during the audit.

Verification

Resolved.

Suggestion 1: Increase Code Comments

Location

Some examples, although not an exhaustive list:
`packages/circuits/lib`

`contracts/consensus/BurnAuction.sol`

Synopsis

Circuits

Code comments within the codebase are critical for developers and reviewers, as they help to define and explain the purpose of each gadget and provide a description of the intended functionality. Furthermore, code comments can highlight other key information, such as which areas are vulnerable to potential failure, which is critical in making sure the system is implemented correctly. A comprehensive description of the intended functionality of each gadget is missing. The lack of sufficient code comments hinders the readability and auditability of the code. For example, in `if_else_then.circom`, a helpful comment would say the following:

The gadget `IfTHenElse(n)` is satisfied if and only if:
((For all n: obj1[n] == obj2[n]) AND out == if_v) OR (There is a n: obj1[n] =!= obj2[n] AND out == else_v))

Smart Contracts

Additionally, the smart contracts lack sufficient comments. For example, the lack of sufficient comments on the Burn Auction contract made the consensus mechanism particularly time-consuming to examine and understand in detail. Thorough and clear code comments make the system easier to use and review.

Mitigation

Circuits

We recommend the Zkopru team expand code comment coverage so that each gadget is accompanied by a code comment describing what it is intended to prove.

Smart Contracts

Additionally, expand smart contract code comments to include comments describing the intended behavior.

Status

The Zkopru team has added descriptive code comments to the circuits and smart contracts.

Verification

Resolved.

Smart Contracts

Issue F: Front Running of Challenge Transactions

Location

`zkopru/controllers/Challengeable.sol#L48`

Synopsis

Ethereum uses gas auction to order transactions in the mempool: the transaction with the higher gas price or gas limit gets into the mempool first and will be included in the block. A front runner bot which scans the mempool of Ethereum may front run the challenge transaction with higher gas price and gas limit and, as a result, gets the slash rewards.

Impact

A front runner can steal the slash rewards from original validators, which is ⅔ of the proposer staked (32 ETH) in the burn auction, and weakens the incentivization of performing the validation work.

Preconditions

Anyone can front run a transaction using a higher gas price/limit as long as the profit it gains is higher than gas cost.

Feasibility

A front runner bot that constantly scans the mempool looking for profitable transactions can front run the challenge transaction. These kinds of bots are known to exist and have carried out attacks of this type in the past.

Technical Details

The challenger sends a challenge transaction to the `Zkopru.sol` contract, then it redirects the call to the fallback function of `Challengeable.sol`, which redirects it to the validators contract for validation. If

the challenge is successful, it will trigger the slash function, the bad block proposer will be slashed and the challenger will be rewarded. The front runner who front runs the challenge transaction will get the reward instead of the real challenger.

Mitigation

There is no known mitigation, for such front running attacks, at this point. Since front running attacks are the natural consequence of the Ethereum mempool, the possibility of such attacks is a known disadvantage of optimistic rollups. We recommend that the Zkopru team monitor the progress of front running research for future potential mitigation strategies. In the interim, we also suggest that the validators be warned of this possible attack.

Remediation

There is no known effective protection against front running. Sending the transaction to a trustworthy miner which includes it in the block without broadcasting it to the network would block the attack, however, this is not practically implementable since mining is currently too costly to make this worthwhile.

Status

The Zkopru team has acknowledged and responded to the front running concern on slashing rewards. At present, they have chosen to keep the system as simple as possible. Our team considers this to be a reasonable choice, particularly given the nature of this issue and that introducing complexity at this stage may introduce additional harm. In addition, we note that this could reduce the incentives for validation. Nonetheless, the issue remains unresolved at the time of this verification.

Verification

Unresolved.

Issue G: Front Running of Fees in Withdraw Transactions

Location

`zkopru/controllers/UserInteractable.sol#L256`

Synopsis

An attacker can front run the withdraw transaction and steal the caller fee.

Impact

If the fee is being front run, the withdraw caller would lose the incentive to withdraw for other users.

Preconditions

When a withdraw caller calls `withdraw()` for another user and there is a caller fee in the withdrawal note, a front runner can front run the fee in the transaction. When user A does not have sufficient ETH to pay the transaction fee to withdraw, user B can call withdraw for user A. In this case, user A would attach a fee for the withdrawal caller (user B).

Feasibility

The incentive of a profit is difficult to estimate, and would depend on how high the fees are in comparison to the gas costs.

Technical Details

When the attacker front runs the `withdraw` transaction, the fee will be transferred to the message sender (the front runner) as the front runner is not the note owner.

```
(bool success, ) = msg.sender.call{ value: fee }("");
```

Remediation

Although there is no known effective protection against front running in general, this issue can be remediated by removing the use of `msg.sender`, and explicitly using signed state as the receiver of the fee payment.

Status

The Zkopru team has acknowledged and responded to the front running concern on withdrawal fees and, at present, have chosen to keep the system as simple as possible. Our team considers this to be a reasonable choice, particularly given the nature of this issue and that introducing complexity at this stage may introduce additional harm. Nonetheless, the issue remains unresolved at the time of this verification.

Verification

Unresolved.

Issue H: Reentrancy Attacks

Location

[zkopru/controllers/UserInteractable.sol#L269]

[zkopru/controllers/Coordinatable.sol#L183]

Synopsis

An attacker can commit a reentrancy attack using the `withdraw()` function to drain all the Ether funds from the contract. We also found the reward `withdraw` function will transfer before decrementing the award state and may also be reentered.

Impact

The attacker could drain all the Ether funds from the contract.

Preconditions

The attacker can make a contract as the destination address of `to.call( value: eth)` and call back into the `withdraw` function in its fallback function, which would withdraw all the funds in the contract.

Feasibility

Highly feasible, as it's easy to carry out and highly profitable since the only costs for the attacker are gas costs.

Technical Details

The withdrawn state update (set to true) is done after the fund transfer call, the attacker can call the `withdraw()` function repeatedly to withdraw all funds before the withdrawn state is updated.

Remediation

Use the check effects interaction pattern. Update the state right after the state check:

```
require(!Storage.chain.withdrawn[withdrawalHash], "Already withdrawn");

Storage.chain.withdrawn[withdrawalHash] = true;
```

Status

The Zkporu team has moved the state that marks the withdrawal boolean to before the transfer function, correcting the possible reentrancy attack.

Verification

Resolved.

Issue I: Fix Reentrancy Code Patterns

Location

zkopru/controllers/Coordinatable.sol#L184

zkopru/controllers/Coordinatable.sol#L59

Synopsis

There are a few other locations where the contract state is only updated after calling external functions, which make the system susceptible to reentrancy attacks in which an attacker repeatedly calls certain functions to extract profit.

Impact

The different invocations of the function may change contract data in destructive ways, which could expose other vulnerabilities.

Preconditions

A call to an external function before finishing internal work, e.g. state change, in the contract.

Feasibility

Highly feasible as long as an attacker can gain profits from the external calls (e.g., withdrawing funds repeatedly).

Technical Details

The attacker can call the involved functions repeatedly, before the first invocation of the function is finished. This may cause the different invocations of the function to interact in destructive ways. For example:

```
payable(proposerAddr).transfer(amount);

proposer.reward -= amount;
```

The attacker can use an external contract as the proposer address to call back the involved function which repeatedly transfers funds, before the reward is deducted. Only here the attack is not exploitable as the **transfer** function is only forwarded with a limited amount of gas (2300 wei), the call back is not possible with this amount of gas.

Remediation

Call external functions before finishing internal work (e.g. state updates), update the state right after state check.

```
require(proposer.reward >= amount, "You can't withdraw more than you
have");
```

```
proposer.reward -= amount;

payable(proposerAddr).transfer(amount);
```

Status

The Zkopru team has issued a commit that contains a list of smart contracts and functions where the correct check effects interaction pattern has been applied, eliminating the possibility for reentrance attacks on these functions.

Verification

Resolved.

Issue J: No Nullifier Uniqueness Check

Location

middlewares/default/block-generator.ts#L44

Synopsis

The Zkopru SNARK implements a commitment nullifier scheme that requires each revealed nullifier to be unique, in order to prevent double spending. It is therefore the burden of any verifier to not only check the correctness of the Groth16 proof [G16], but also the uniqueness of any revealed nullifier. Since the proposer includes the transactions to generate blocks, each spent UTXO should therefore have its unique nullifier checked.

Impact

The block proposer will get slashed if there is a redundant nullifier, causing a double spend issue when someone challenges the transaction. An attacker who submits a double spend transaction can challenge the block which includes the invalid transaction later to claim the slash reward.

Preconditions

The block producer/coordinator does not check the uniqueness of the spending note.

Feasibility

Highly feasible, as there is very high profit from this attack. An attacker can double spend if no one challenges the invalidate transaction or get a slash reward by challenging the bad block, since they know it's invalid.

Remediation

Add a transaction nullifier uniqueness check when the proposer generates blocks.

Status

The Zkopru team has responded noting that this is not an issue as they have client code that will ensure that the identifier is unique in a dry run.

Verification

Resolved.

Issue K: No Withdrawal Note Validity Check (Out of Scope)

Location

coordinator/src/api.ts#L111

zkopru/controllers/UserInteractable.sol#L92

Synopsis

There is no withdrawal note validity check when the prepayer receives an instant withdraw request. If the withdrawal note is invalid and the block which includes it is slashed later by a validator challenge, the bad block will not be finalized and the withdrawal note becomes non-withdrawable. This would result in the prepayer who bought this note taking the loss.

Impact

The prepayer will lose the funds they prepaid to the user who requests instant withdrawal with an invalid withdrawal note, which could be caught by a challenger before it is finalized, as they cannot withdraw.

Preconditions

This attack is possible if the withdrawal note is not valid, the prepayer does not verify it and pays the instant withdrawal. In this case, the note is transferred to the prepayer, the block which includes the invalid withdrawal note gets challenged later and cannot be finalized, and the prepayer cannot withdraw the prepaid withdrawal note they bought.

Feasibility

This is feasible as an attacker can forge an invalid withdrawal note and request instant withdrawal from a prepayer as long as the prepayer doesn't check the validity of the withdrawal note.

Remediation

When receiving an instant withdraw request, we recommend that the coordinator or prepayer should add a validity check for the withdrawal note that it is correctly included in the block.

Status

The Zkopru team has introduced verification checks in the API that may resolve this issue. However, we recommend that this out of scope code be further reviewed and evaluated, as it has not been thoroughly reviewed by our team.

Verification

Resolved.

Issue L: SNARK Validation Missed in On-chain/Off-chain Validation (Out of Scope)

Location

src/validator/validator.ts#L398

Synopsis

When the validator validates a transaction, it runs a couple of validations on-chain/off-chain. One of the validations is SNARK validation, which validates the SNARK proof in the transaction and it is missed in the called validations.

Impact

Invalid transactions can pass the validation, and an attacker can generate a transaction which outputs an invalid note of arbitrage amount, then withdraw it to L1, draining all the funds in the contract.

Feasibility

Highly feasible, as an attacker can gain huge profit just by making an invalid transaction.

Technical Details

The `validateTx()` function of the validator, returns an array of on-chain/off-chain validator functions in the array `fnCalls`, `validateSNARKCalls` (which validates SNARK proofs) is missed.

Remediation

Add the missed `validateSNARKCalls` in the array (`fnCalls`).

Status

This issue was reported to the Zkopru team and fixed during the audit and prior to the delivery of the Initial Audit Report.

Verification

Resolved.

Issue M: No Caller Fee When Withdrawing Only Non-ETH Assets (ERC-20 Token and NFTs) (Known Issue)

Location

[zkopru/controllers/UserInteractable.sol#L246]

Synopsis

When withdrawing L2 assets to L1 for other users, the withdraw caller can get a caller fee as a reward. When there are only ERC-20 tokens and NFTs (and no ETH assets) in the withdrawal, the caller will not get a caller fee.

Impact

Withdraw caller will not get the caller fee.

Preconditions

The withdraw note only contains ERC-20 tokens and NFTs and the withdrawal fee is not zero.

Technical Details

In the `withdraw` function, when `eth` is not `0`, the contract transfers the fee to the caller (whether the caller is the owner or not). When `eth=0`, there is no fee transfer logic, only the ERC-20 tokens and NFTs are transferred to the owner.

Remediation

Add fee transfer to the caller when `eth=0`.

Status

The addition of the fee logic has been moved out of the conditional statement, thus resolving this issue.

Verification

Resolved.

Issue N: Front Running of Burn Auction Bids

Location

contracts/consensus/BurnAuction.sol

Synopsis

Staked coordinators may claim block proposal rights by winning a "burn auction". They make bids that must increase by 10% each accepted bid, until a time limit. But a front runner can submit the same bid with a slightly higher gas price to get their transaction moved to the start of the next block and considered the first bid, and thus winning that price.

Impact

A front runner can win burn auctions and force honest bidders to increase bids, possibly making coordinating Zkopru unprofitable for honest participants.

Preconditions

An honest coordinator is making a burn auction bid to gain block submission rights and then make a profit from transaction fees.

Technical Details

For coordinating to be profitable, the amount spent in burn auctions must be less than the average amount gained in fees. This means that the burn auction is a competition about who can most accurately model the distribution of fees. The requirement that a bid be 10% more than the previous bid actually makes coordinating quite profitable. Assuming accurate expected fees are known, then it is possible to make a bid slightly below the average fee, such that the next bid will be above it. For example, if 109 in fees is expected, and then 100 is bid, the next bid would be 110, which would lose 1 unit. Since fees are public information, it's expected that all coordinators know the averages. However, a front runner can submit the same bid, but at a slightly higher gas price, this forces the honest coordinator to make a bid at least 10% more, which may be unprofitable.

Mitigation

Bids should be submitted with very high gas prices as close to the end of the block as possible, to reduce the chance that a front runner will have time to get their transaction in.

Remediation

Bids could be made with a commit-and-reveal pattern, or use a different consensus mechanism that is not threatened by front running, such as having all coordinators take turns.

Status

The 10% increment step has been removed, making the profitable bid less obvious for front runners. However, since front running may still occur, we consider the issues partially resolved. We recommend implementing a different auction or consensus mechanism not threatened by front running to fully resolve the issue.

Verification

Partially Resolved.

Issue O: Old ERC-20 Token Interfaces May Lead to Stuck Tokens or Be Blocked from Use

Location

zkopru/controllers/UserInteractable.sol#L263-L266

zkopru/controllers/UserInteractable.sol#L135

Synopsis

All instances of ERC-20 token transfer and `transferFrom` calls do not have a wrapper to anticipate the event that a token interface does not adhere to the standard of reverting in the case of failure. A missing return value bug may arise, leading to older tokens becoming locked on the contract when Zkopru needs to interact with them to withdraw. The Zkopru contract uses `IERC20.sol`, and if it interacts with a token that does not match IERC20 return values, there is a potential for the tokens to be locked in the contract.

To combat this possibility, the registration of a token in Zkopru must return a successful transfer trial. This can add a small amount of extra gas consumption and will block any older tokens that have a bad standard interface.

Impact

Tokens may become locked in the Zkopru contract and will be destroyed if this is not prevented in a standard way. Ensuring that tokens must transfer before registering them should reduce the possibility of this incident happening but will block bad interface tokens.

Preconditions

A token that does not conform to the standard that Zkopru expects is deposited and is unable to be withdrawn after. This token must get registered by passing the transfer checks so this may just result in the token being blocked from use in Zkopru.

Feasibility

Some tokens that are in use still adhere to a standard that does not return a value. A list of them is provided in the documentation on the missing return value bug.

Remediation

We recommend using a standard safe transfer wrapper, such as the one used by Uniswap or provided by OpenZeppelin, which will anticipate these interfaces with no return value and allow them to still be used.

Status

The safe transfer wrapper has been added to the `Coordinatable.sol` controller, thus eliminating the possibility of bad tokens being stuck.

Verification

Resolved.

Issue P: Operator Has Total Authority Over State Verification

Location

contracts/zkopru/Zkopru.sol#L31

Synopsis

In the current implementation, the Zkopru contracts allow for the centrally owned operator to replace the SNARK circuit or the associated common reference string at any time. This SNARK circuit is used to verify shielded transactions. If this verification circuit is replaced with a malicious one, it will allow state updates that are malicious.

Impact

Severe. If the operator of the SNARK circuit becomes malicious or compromised, they are able to control all state updates. This can lead to the users losing all funds in the system.

Feasibility

In the early stages of the Layer 2 chain, it is against the self interest of the owner to commit fraud. However, this is a single point of failure that could become compromised through other attack methods.

Mitigation

While the SNARK circuits are still being finalized, it is favorable to be able to switch a broken circuit out quickly. Once the SNARK circuits have been finalized, the update function should be immediately switched to one controlled by a `multisig` contract.

Remediation

A democratic and transparent approach to the update of the consensus rules would be an ideal alternative to the currently centralized design. One way to do this would be to place a proposed circuit update on-chain where it could be voted on by the users of the system, or by those qualified to ensure that the new circuit is correct. SNARK circuits are complicated protocols and not easily reviewed by most people, so votes should be only placed on circuits that have been thoroughly audited by reliable and ethical sources.

Status

The setup phase of the smart contracts now renounces ownership on the OpenZepplen ownership smart contract.

Verification

Resolved.

Issue Q: Deposit Does Not Check For Registered Token

Location

`zkopru/controllers/UserInteractable.sol#L146`

Synopsis

There is a registration process for tokens where they must be able to transfer tokens. This is a precaution for tokens that want to use Zkopru but do not adhere to the ERC-20 standard interface. The deposit function here should check that the tokens entering the Layer 2 chain have been registered and conform to ERC-20 standards.

Impact

Tokens deposited with an incorrect standard may become stuck on the contract.

Feasibility

This is possible if tokens do not adhere to standards and this has happened in other systems.

Remediation

Add a requirement that the token being deposited, is in the registered tokens storage.

```
require(Storage.chain.registeredERC20s[token]  != address(0));
```

Status

The tokens entering now use the safe transfer wrapper to ensure that there is no return value bug and that the transfer interface exists on the tokens. However, the fix now introduces redundant code, which should be removed:

zkopru/controllers/UserInteractable.sol#L185-L186

zkopru/controllers/UserInteractable.sol#L192-L193

As `_checkNoteFields` already performs the check.

Verification

Partially Resolved.

Issue R: Coordinator's URL/IP Address is Exposed in Auction Process, Which Can Be Exploited For a DDoS Attack

Location

contracts/consensus/BurnAuction.sol#L108

coordinator/src/auction-monitor.ts#L207

Synopsis

In a burn auction, coordinators are required to set the URL when they join the auction. This URL/IP information is open to all coordinators. This enables a possible Distributed Denial of Service attack, in which a malicious coordinator can prevent the legitimate block proposer from proposing blocks and snatch the block proposing right without winning the burn auction.

Impact

A malicious coordinator can gain the block proposing right without being the winner in the burn auction or an attacker can prevent coordinators from proposing blocks to disrupt the system.

Preconditions

Assuming there are not many coordinators at the beginning of the system launch, an attacker can launch a DDoS attack against all coordinators during the block proposing round.

Feasibility

Highly likely, as winning the block proposal rights can be very competitive. The feasibility increases when there are fewer coordinators in an auction.

Technical Details

The coordinator is required to set their URL/IP with the auction smart contract when they join an auction. The smart contract will emit an event `UrlUpdate` once the coordinator sets the URL, and all coordinators

who join the auction will have access to the coordinator URL/IP by subscribing to the `UrlUpdate` event. A malicious coordinator can use this information to launch a DDoS attack against the winning coordinator during the block proposing round (10 minutes at 15 sec/block), thus preventing them from proposing blocks. After half of the round passes then any other coordinator can propose, but an attacker could launch a DDoS attack against the other coordinators, especially when there are not many of them. As a result, the malicious coordinator gains the proposal rights without winning the auction.

Mitigation

Implement an incentive mechanism that can attract enough coordinators to participate in the system, and have multiple coordinators propose the block during the block proposing round to ensure that, if one block proposer is attacked, another coordinator can still proceed with proposing the block. After half of the round has passed, any other coordinator can propose a block (since it is harder for the attacker to attack all other proposers). Additionally, we recommend that the Zkopru team investigate possible DDoS protections for coordinators to implement.

Remediation

Remove the URL/IP information requirements for coordinators to join the auction, using some anonymous routing protocol (Tor, I2P, Raven e.g.) to protect coordinator network metadata privacy (IP and its link to on-chain ID) from deanonymization attacks.

Status

The Zkopru team has responded stating that a browser wallet will be deployed on a subdomain of zkopru.network and the coordinators will only accept HTTP requests from the whitelisted domains. However, DDos attacks can still send non HTTP requests/data (e.g. SYN) to flood the coordinator's bandwidth. As a result, we consider this issue to be partially resolved.

Verification

Partially Resolved.

Suggestion 2: Increase Test Coverage

Location

`packages/circuits/tests`

`test/validators/tx-validator.soltest.js#L93-L97`

`test-cases/test/validators`

Synopsis

Circuits

The zk-SNARK circuits include some test coverage, which demonstrate the intended use for the circuits. However, additional tests to account for potential edge cases would help to further protect against malicious actions and unexpected behavior

Smart Contracts

The smart contracts include test coverage that is limited to success cases (when the slash condition is false), and does not incorporate failures and errors. Furthermore, each validator should have corresponding tests (e.g. `withdrawtreeValidator`, `NullifierTreeValidator`, `DepositValidator` and `UtxoTreeValidator`), which are currently insufficient.

Increasing test coverage with specific attention to fail cases will help identify simple errors and prevent functionality from breaking when new code changes are introduced. In addition, tests help to detect and prevent unintended behavior and edge cases, which may result in the potential vulnerabilities.

Mitigation

Circuits

We recommend increasing test coverage to account for potential edge cases and unexpected behavior.

Smart Contracts

We recommend increasing test coverage for fail cases (e.g. slashable condition) and for all validator contracts.

Status

The Zkopru team has responded that they have provided additional tests and are currently onboarding new developers so they expect test coverage to increase in the near term. At the time of this verification, test coverage continues to be insufficient.

Verification

Unresolved.

Suggestion 3: Expand System Documentation

Location

https://docs.zkopru.network/v/burrito

Synopsis

The project documentation provides an overview of the Zkopru system and the intended functionality for the smart contracts. However, the smart contracts interact with many off-chain components of the circuit system (e.g. coordinator, validator, SNARK verifier, wallet client, fullnode, etc.) and these interactions are not comprehensively documented.

Furthermore, build and test documentation is missing and would provide a clear description of the setup process, which would avoid the potential for mistakes.

Insufficient documentation of this nature leads to difficulty understanding, reviewing, and using the system effectively.

Mitigation

We recommend creating detailed documentation of how the entire system interacts, which would provide a deeper appreciation and understanding of Zkopru. In addition, we recommend providing build and test documentation, which would prevent the potential for issues to arise during the initial setup phase.

Status

The Zkopru team has responded that the documentation would be updated before their launch to mainnet. At the time of this verification, this suggestion remains unresolved.

Verification

Unresolved.

Suggestion 4: Remove Duplicate Code

Location

Example 1:
zkopru/controllers/Challengeable.sol#L69

zkopru/controllers/Challengeable.sol#L66

Example 2:
zkopru/libraries/MerkleTree.sol#L265

zkopru/libraries/MerkleTree.sol#L289

Synopsis

Example 1 demonstrates an instance of code duplication, where two identical lines of code exist in the system. Example 2 demonstrates an instance of two lines of code that are different but perform an identical function.

Code duplication should be avoided, as it increases storage costs and can lead to misunderstandings when making changes to and reviewing code.

Mitigation

We recommend performing a code review to identify and remove all duplicated lines of code.

Status

The Zkopru team has removed the duplicate code.

Verification

Resolved.

Suggestion 5: Update Compiler Version

Location

zkopru/controllers/Challengeable.sol

Synopsis

The compiler version is set to version 0.6.12, which does not incorporate newer compiler fixes and updates.

Mitigation

We recommend updating the compiler version to 0.7.0–0.7.4. While more recent versions exist, we advise against their use as they may contain a higher probability of unknown issues.

Status

The compiler has been updated to v0.7.4. Since the time of the original report, there has been more time for compiler versions to settle. We recommend going as high as 0.8.0 if desired to achieve the new features such as EVM safe math and to regularly update the compiler to up to date, stable versions.

Verification

Resolved.

Suggestion 6: Add Address Input Sanitization

Location

zkopru/controllers/Coordinatable.sol#L41

Synopsis

There are two ways to enter stake to the coordinator contract. If calling `register()`, the address will be hardcoded to be `msg.sender` and there will be no issue. However, if `stake()` is called directly, the address input is never examined for format correctness.

Mitigation

We recommend adding a check to the `address coordinator` input for at least `NULL` address with `require(coordinator != address(0));`.

Status

The Zkopru team has issued an update and a check for the zero address supplied for the coordinator is now present.

Verification

Resolved.

Suggestion 7: Address TODO Comments

Location

zkopru/controllers/UserInteractable.sol#L166

zkopru/controllers/Coordinatable.sol#L192

zkopru/libraries/Types.sol#L370

contracts/consensus/BurnAuction.sol#L170

Synopsis

The TODOs listed in the code may impact the security of the system. In the event that they are not planned to be implemented, a comment associated with each TODO explaining the rationale would benefit from security review.

Mitigation

Complete TODOs or add documentation as to why they are unnecessary.

Status

The Zkopru team has responded that most of the TODO comments were outdated and have removed them where necessary.

Verification

Resolved.

Suggestion 8: Audit Off-Chain Components

Location

packages/coordinator

packages/core

packages/account

packages/transaction

packages/zk-wizard

Synopsis

Off-chain components (coordinator, wallet client, validator, SNARK verifier, synchronizer, transaction builder) were out of scope for this audit. To check the security of the system as a whole, these would require a security audit.

Mitigation

We recommend that audits of the off-chain components, including the coordinator, wallet client, validator, SNARK verifier, transaction builder and synchronizer, are undertaken to check for potential vulnerabilities and to ensure that the interaction of the components function as intended.

Status

The Zkopru team has responded that, given that Zkopru is an optimistic rollup protocol, everything should be guaranteed on-chain. Off-chain components are only the implementation and they do not have any plans to have them audited.

Verification

Unresolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. Although we are a small team, we believe that we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

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 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 whitebox penetration testing. We look at the project's web site 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. We hypothesize what vulnerabilities may be present, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

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, 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.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.